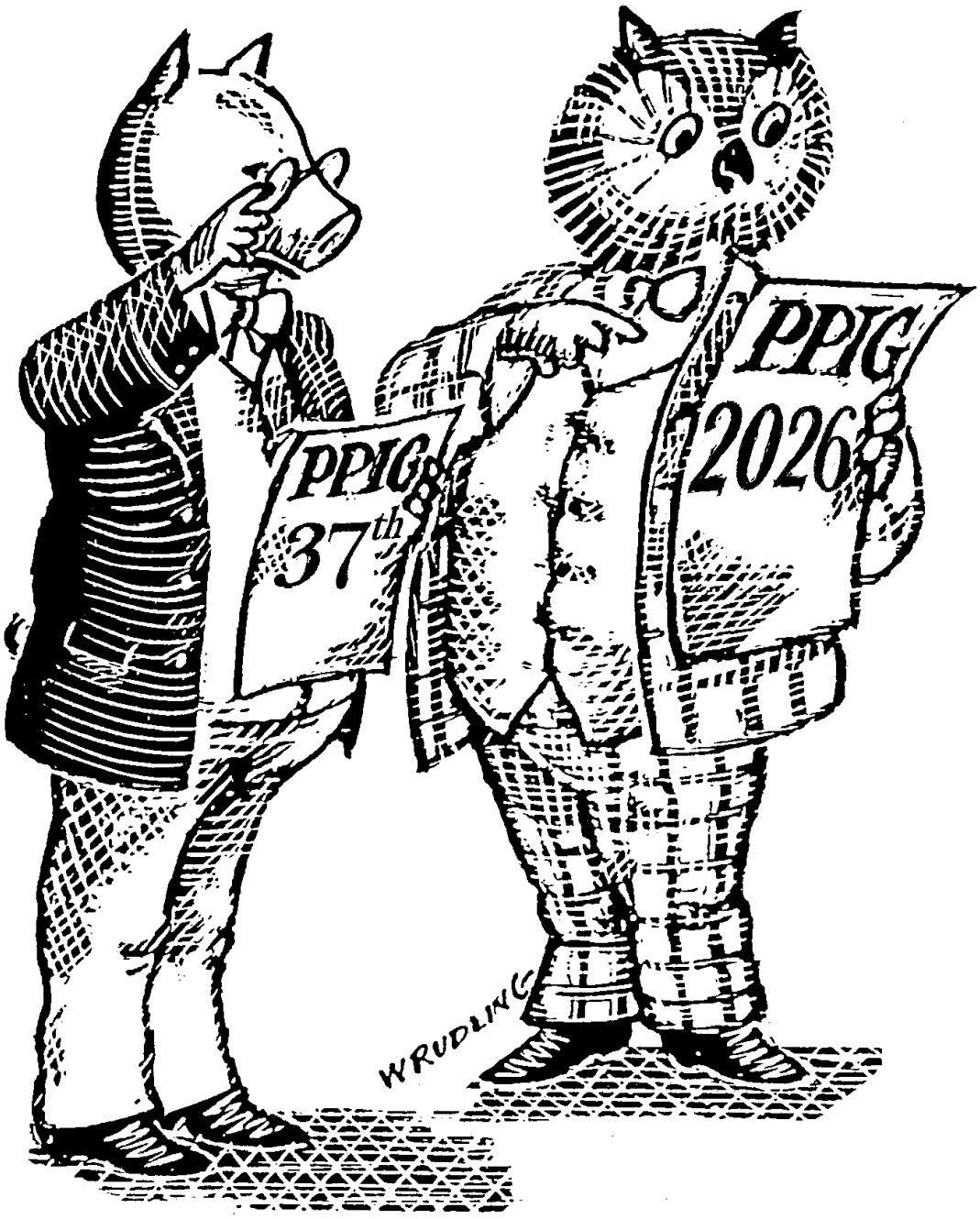
PPIG 2026

Proceedings of the 37th Annual Workshop of the Psychology of Programming Interest Group

27-29 July 2026

UCLIC, London, UK



Edited by
Nancy Xia, Luke Church, and Mariana Marasoiu

Editors' preface

This year's workshop brought together our international community to reflect on, question, and expand how people engage with code, computation, and emerging technologies. Over three full days, the programme bridged theoretical and philosophical inquiry with tangible, playful experimentation.

PPIG was generously welcomed at the UCL Interaction Centre (UCLIC) in central London by the centre's director Nadia Berthouze. We enjoyed keynote addresses from Katie Seaborn, exploring the nuances of kawaii vocal aesthetics and affective computing, and Marianna Obrist, guiding us beyond the screen into multisensory design. Alongside paper sessions spanning rapid-changing developer practice, accessibility, cognitive models, inclusive design, and developer reasoning, our discussions also took hands-on form through immersive demonstrations - from directing robots in VR to dynamic explorations of agentic pair-programming and vibe-coding.

PPIG has always been more of an interest group than a traditional conference - a forum where ideas are nurtured through constructive, generous dialogue rather than merely broadcast.

We want to extend a particular and heartfelt thank you to all our authors who took the time and care to substantially revise their papers following the workshop. Seeing authors incorporate questions raised during talk sessions, insights from coffee breaks, and perspectives shared across the room into their final manuscripts demonstrates the real value and spirit of the PPIG community.

We also would like to thank the Doctoral Consortium mentors and students for energising the start of the week, our peer reviewers for their thoughtful and constructive feedback, and every attendee - both in London and online - for their warmth and curiosity. We hope you find reading these revised works as stimulating and generative as the conversations that inspired them.

The Editors

Nancy Xia, Luke Church, and Mariana Mărășoiu

London, 2026

PPIG 2026
Programme & Proceedings Index

Monday, 27 July

12:00 - 13:00	*Lunch & registration*	
13:00 - 13:30	Welcome!	
13:30 - 14:30	*Keynote:* **From Anime to the Vocal Arts: Programming Kawaii Vocalics** Katie Seaborn, University of Cambridge	6
14:30 - 15:00	*Coffee break*	
15:00 - 16:30	*Talks*	
	Toward Adaptive Programming Environments for Blind Developers: A Large-Scale Analysis of Community Challenges Niloofar Azizi and Alan Blackwell	7
	Agency, Expertise, and the Limits of Augmentation in LLM-Assisted Programming Workflows Gabriel Hughes and Yvonne Rogers	21
	Supervision Theatre: Human-on-the-Loop as the Spear Carrier in the Agentic Play Joana Vieira, Carlos Silva and Menno Cramer	32
16:30 - 17:15	*Hands-on demo:* **Just tell the robot what to do, in VR** Anahide Silahli and Thiago Rocha Silva	45
18:30 onwards	*PPIG Conference Dinner* The Union Tavern, 52 Lloyd Baker Street, London, WC1X 9AA	

Tuesday, 28 July

9:00 - 9:05	*DC Minute Madness*	
9:05 - 10:30	*Talks*	
	NodeToJoy: Understanding Probabilistic State through Interactive Visualisation Charlotte Cox and Alan Blackwell	47
	Conceptualizing Agentic Programming with Scratch: Enhancing Computational Thinking on a Higher Level of Abstraction Orit Hazzan and Noa Ragonis	61
	What if Offloading Is Not the Problem? Ilya Zakharov	71

10:30 - 11:00	*Coffee break*	
11:00 - 12:00	*Keynote:* **Beyond the Screen: Taste, Smell, and the Untapped Sensorium of Human-Computer Interaction** Marianna Obrist, University College London	83
12:00 - 13:00	*Lunch*	
13:00 - 14:30	*Talks*	
	Talking as Theory Reconstruction: Eliciting Architectural Decisions from Developers via Interview and LLM-Based Synthesis Bernhard Mayr, Alexander Feiglstorfer and Dominik Angerer	84
	Investigation into the Factors Which Influence Students to Study Computer Science between Second and Third-Level Education – A Working Study Miriam Harte, Keith Quille, Rajesh Jaiswal and Keith Nolan	91
	Autonotism: the choices we could have made in AI-assisted programming Advait Sarkar	96
14:30 - 15:00	*Coffee break*	
15:00 - 16:30	*Talks*	
	Designing for Persistence: AI Chatbot Architecture, Gender Bias, and Inclusive Design for Computing Higher Education Alina Berry	112
	Narrative Structures for Process-Level Reasoning in Software Engineering Md. Hasan Tareque and John Anvik	123
	Can Gamification Encourage Debugger Usage? Teresa Dreyer and Christoph Bockisch	131
16:30 - 17:00	*Walk (optional)*	
17:00 - 18:00	*Novelty Automation (optional)*	

Wednesday, 29 July

9:00 - 10:30	*Talks*	
	A Text Foraging Perspective on Processing Code Teresa Dreyer, Jan Tünnermann and Christoph Bockisch	137
	Towards Supporting Novice Programmers' Web Search Skills for Using Unfamiliar Libraries and APIs Mehmet Arif Demirtaş, Jinyoung Hur, Alex Atcheson, Claire Zheng and Kathryn Cunningham	142

	Evaluating and Adapting the ATTA for Computational Thinking Skills Assessment in Higher Education: A Pilot Study Frances Sheridan, Paul Stynes, Pramod Pathak and Keith Quille	150
10:30 - 11:00	*Coffee break*	
11:00 - 12:00	*Workshop:* **Vibe Coding in Pairs** Gabriel Hughes	181
12:00 - 13:00	*Lunch*	
13:00 - 14:30	*Talks*	
	Certifiably Curious: Why do Second-Level Teachers Engage in Industry AI Certification? Miriam Harte, William Clifford, Roisin Faherty, Nikhitha Guttula, Rajesh Jaiswal, Musfira Jilani, Na Li, Joyce Mahon, Keith Nolan and Keith Quille	183
	MVC-ARENA: A Card-Based Representation for Reasoning About MVC Architecture Md Jamiur Rahman Rifat and John Anvik	199
	Talents, Competencies and Techniques of Business Analysts: Talented Individuals to Dream Teams Anna Bobkowska	205
14:30 - 15:00	*Coffee break*	
15:00 - 16:30	*Talks*	
	Interruptive Thinking: An Empirical Study of Operator-Side Control in AI Interaction Sudhanshu Garg	212
	Exploring the impact of a single TPL session on Primary School Teachers' Domain-Specific Self-Esteem Amanda O'Farrell, Keith Quille and Keith Nolan	222
	Reversing the Mirror: The Moral Programming Language as Primary Notation for Agentic AI Varad Vishwarupe and Alan F. Blackwell	236
16:30 - 17:00	*PPIG Prizes & close*	

PPIG 2026 Doctoral Consortium

Programme & Proceedings Index

Monday, 27 July

9:00 - 12:00 *Doctoral Consortium*

Designing Purpose-first Instruction for Supporting End-user Programmer Learning Goals 251
Mehmet Arif Demirtaş

From Abstract to Concrete: Programming the Running System via Direct Manipulation and Live Feedback 255
Eva Krebs

Beyond prompt engineering: Rethinking agency in LLM interactions -
Gabriel Hughes

How Is AI Reshaping Computer Science Students' Cognition of Programming? 260
Ruolin Li

From Anime to the Vocal Arts: Programming Kawaii Vocalics

Katie Seaborn
University of Cambridge

Keynote abstract

Elements of Japanese culture, like anime characters and kawaii, the Japanese instantiation of cuteness, have crossed cultural borders and entered digital spaces outside of Japan. In this talk, I introduce the science of kawaii vocalics, a field of study I coined. Kawaii vocalics focuses on how the vocal medium and paralanguage of cute voices can be studied and applied, especially programmatically within computer applications. I cover the underlying theories, from cuteness as baby schema to Nittono's two-layer model of kawaii, as well as methods by which to manipulate the kawaii-ness of voice, including the "super kawaii vocalics" method myself and my collaborators created. I touch on interlinking factors like Japanese first-person pronouns (of which there are many that connect to various facets of social identity, relations, and power structures) to demonstrate how character archetypes like "bokukko" can be elicited through voice and text choices in computer agents. I demonstrate subtle and full-on applications of kawaii vocalics in a game interface and collaborative soundboard drum circle.

Speaker bio

Katie Seaborn is an Associate Professor of Human-Computer Interaction and Critical Computing in the Department of Computer Science and Technology (CST) at the University of Cambridge. They are also an Associate Fellow in the Centre for Human Inspired Artificial Intelligence (CHIA) and Visiting University Associate Professor at Institute of Science Tokyo (formerly: Tokyo Institute of Technology). They are Director of the Aspirational Computing Lab (established April 2020). They received their Ph.D. in Mechanical and Industrial Engineering from the University of Toronto (2016) with a Major in Human Factors and a Collaborative Specialization in Knowledge Media Design. They also hold B.A. and M.Sc. degrees from the School of Interactive Arts and Technology (SIAT) at Simon Fraser University.

Toward Adaptive Programming Environments for Blind Developers: A Large-Scale Analysis of Community Challenges

Niloofar Azizi[1] and Alan Blackwell[2]

[1] Imperial College London
n.azizi@imperial.ac.uk
[2] University of Cambridge
afb21@cam.ac.uk

Abstract. We aim to make programming more adaptable and accessible for blind users by identifying the key challenges and unmet needs that require new ideas and innovation. Our main research question is: *How can we better adapt programming environments, tools, and workflows to the needs of blind programmers?* To investigate this question, we conducted a large-scale analysis of the Blind Programming mailing-list archive (Program-L), examining approximately 10,000 emails exchanged between March 2022 and March 2026. Using an LLM-assisted systematic thematic analysis framework, we reconstructed discussion threads, extracted keywords and accessibility-related issues, and generated candidate codes and themes grounded in the original discussions. Our analysis identifies recurring accessibility barriers, unresolved technical challenges, and common user frustrations across programming workflows, development environments, accessibility tools, and assistive technologies. The resulting findings provide an evidence-driven overview of the domains where new adaptive tools, research directions, and accessibility-focused innovations are most urgently needed to improve programming accessibility for blind individuals.

Keywords: Blind programming, Accessibility, Assistive technology, Inclusive software development, Programming accessibility, Visual impairment, Accessible coding environments

1 Introduction

Programming environments have become increasingly complex, collaborative, and dependent on rapidly evolving development tools and workflows. While accessibility research has improved support for blind and low-vision users in many domains, blind programmers still face substantial barriers in areas such as IDE interaction, debugging, terminal workflows, collaborative development, and accessibility of modern web-development ecosystems. These challenges often emerge not from a single inaccessible interface, but from the interaction between multiple tools, frameworks, configuration systems, and assistive technologies within real-world software-development workflows.

To better understand these challenges, we analyzed approximately 10,000 emails from the Blind Programming mailing-list archive (Program-L), covering discussions between March 2022 and March 2026. Using an LLM-assisted thematic-analysis pipeline combined with semantic clustering, we identified recurring accessibility barriers, adaptive practices, and unresolved technical concerns discussed by blind programmers in naturally occurring community interactions. Our findings provide an evidence-driven overview of the areas where new adaptive programming tools, accessibility-aware workflows, and future research directions may be most urgently needed. This paper makes three contributions:

- a large-scale reconstruction of blind-programming community discussions,
- an LLM-assisted thematic-analysis framework for analyzing accessibility-related programming discussions,
- an empirical characterization of recurring accessibility barriers, adaptive practices, and unmet workflow needs in non-visual programming environments.
- a longitudinal analysis of how generative coding assistants entered these workflows, and of whether they mitigated or relocated existing barriers.

2 Related Work

Prior work has shown that blind and low-vision programmers face accessibility barriers across software-development workflows, including code navigation, debugging, collaboration, and interaction with professional development environments (Mealin & Murphy-Hill, 2012; Pandey, Kameswaran, Rao, O'Modhrain, & Oney, 2021). Previous studies have also investigated accessibility challenges in web development, Visual Studio Code, UI frameworks, and collaborative programming environments (Ferrari & Hurst, 2021; Seo & Rogge, 2023; Pandey, Bondre, O'Modhrain, & Oney, 2022; Potluri, Pandey, Begel, Barnett, & Reitherman, 2022; Cha et al., 2024) .

More recently, large language models have been explored as tools for supporting thematic analysis and large-scale qualitative data analysis (Naeem, Smith, & Thomas, 2025; Nguyen-Trung, 2025; Wang, Erqsous, Barner, & Mauriello, 2025). However, most prior studies have focused on interviews, controlled studies, or relatively small qualitative datasets. In contrast, our work analyzes several years of naturally occurring community discussions from a large-scale blind-programming archive. By combining LLM-assisted thematic analysis with semantic clustering, we aim to provide a broader evidence-driven view of recurring accessibility challenges, adaptive practices, and unmet needs in real-world non-visual programming workflows.

A recent literature treats generative coding assistants as an accessibility problem in their own right. (Flores-Saviaga, Hanrahan, Imteyaz, Clarke, & Savage, 2025) reports that such assistants both empowered visually impaired developers and exacerbated existing barriers, introducing suggestion overload and context-switching costs; (Chen, Qiu, Wang, Wang, & Yang, 2026) finds efficiency gains alongside difficulty conveying intent and maintaining awareness across multiple views; (Mowar, Peng, Steinfeld, & Bigham, 2024; Mowar, Peng, Wu, Steinfeld, & Bigham, 2025) shows that assistants degrade the accessibility of the code they generate. (Cha, Jackson, Shu, Branham, & van der Hoek, 2026) identifies a verification asymmetry in which blind professionals are least able to detect the hallucinations to which they are most exposed; 16 of their 39 participants were recruited through Program-L, the archive we analyse. All of this work is interview- or task-based. We complement it with longitudinal evidence from four years of naturally occurring discussion, which lets us ask when these concerns emerged and how their composition changed.

3 Method

We analyzed our data following the LLM-assisted thematic analysis framework proposed by (Naeem et al., 2025). This framework consists of six main stages:

- (1) transcription, familiarization with data, and the selection of quotations,
- (2) selection of keywords,
- (3) coding,
- (4) theme development,
- (5) conceptualization through the interpretation of keywords, codes, and themes, and
- (6) development of a conceptual model.

In this study, the stages of keyword extraction, coding, and theme development were performed on discussion threads collected from March 2022 to March 2026.

Transcription, familiarization with data, and the selection of quotations We collected and processed discussion threads from the Blind Programming mailing-list archive (Program-L). The raw archive pages were cleaned and reconstructed into individual discussion threads by grouping and merging messages according to the discussion subject identifiers appearing in the archive URLs. For each thread, metadata such as authors, timestamps, and message bodies were extracted. Duplicated quoted content, quoted reply chains, and archive boilerplate were removed during preprocessing. Representative quotations were retained in order to preserve traceability between the generated analysis and the original discussion content.

Selection of keywords For each reconstructed discussion thread, we used an LLM-assisted prompt to extract candidate keywords grounded in the original thread content. The extracted keywords were generated together with supporting evidence, including the main problem discussed in the thread, unresolved issues, representative quotations, and proposed solutions where available. This ensured that the extracted keywords remained connected to the original semantic context of each discussion thread.

Coding In the coding stage, each discussion thread was analyzed using the extracted keywords, main problems, unresolved issues, and supporting quotations to generate candidate qualitative codes. These codes were intended to summarize the central accessibility-related issue represented by each thread. The generated codes were iteratively reviewed for consistency with the semantic content of the corresponding discussion thread.

Theme development In the theme development stage, related candidate codes were grouped into broader candidate themes representing recurring accessibility concerns and shared user experiences across discussion threads. These themes were developed by examining repeated semantic patterns and supporting textual evidence across multiple threads, rather than relying solely on keyword frequency.

Conceptual clustering. To move from candidate codes and themes toward higher-level conceptual categories, we embedded each thread-level thematic summary using a sentence-transformer model. For each thread, the embedding input combined the generated accessibility-related code, candidate theme, and selected supporting evidence from the original discussion. The resulting semantic embeddings were normalized and reduced using UMAP with a cosine distance metric. We then clustered the reduced representations into ten conceptual groups,

Table 1. Representative examples of thread-level thematic analysis.

Thread topic	Representative quotation	Candidate code	Candidate theme
Platform-specific accessibility tools	"The majority of blind/VI individuals would not be able to afford iOS devices."	Platform exclusivity and affordability barriers	Economic and ecosystem barriers in accessible software
Python and NVDA	"NVDA is written in Python and many blind people start learning it with the idea of contributing."	Assistive technologies shaping language adoption	Influence of accessibility ecosystems on programming practices
AI-assisted coding	"The regular coding is where coders learn their skills."	Concerns about AI reducing skill acquisition	Challenges introduced by AI-assisted development
Blind web development	"Even with this you need a sighted person to look and see that it looks good."	Reliance on sighted assistance for visual layout validation	Tooling gaps in accessible front-end development
Communicating layouts to designers	"How could I talk about layouts and screen reader navigation in a way they understand?"	Difficulty communicating visual structure non-visually	Bridging visual and non-visual accessibility perspectives
IntelliJ accessibility	"Jaws still does not read all the access keys correctly in the JAB."	Screen-reader incompatibilities in IDE environments	Accessibility limitations in professional developer tools
WebAccessibilizer	"The extension can now be installed in just a few clicks."	Reducing installation complexity for accessibility tools	Improving usability of assistive technologies
BIOS accessibility	"You might find a HDMI capture card useful."	Hardware-assisted access to inaccessible interfaces	Technical adaptations for inaccessible systems
Accessible Arduino development	"Getting that accessible kit will not be available anytime soon in India."	Geographic and economic barriers to accessible hardware	Accessibility inequalities in educational hardware ecosystems

which were used as an intermediate analytic layer between theme development and conceptual interpretation. For each cluster, representative threads were selected according to their cosine similarity to the cluster centroid, allowing us to inspect central examples and generate cluster-level interpretations grounded in the original evidence. Representative examples of the conceptual clusters are shown in Table 6. The complete set of clusters is provided in the Appendix.

4 Dataset overview

Table 2 summarizes the main statistics of the reconstructed corpus. The dataset exhibits a highly skewed thread structure, where most threads are short and contain only a small number of posts. Specifically, 692 threads contain a single post, while 319 and 328 threads contain two and three posts, respectively. Nevertheless, the corpus also includes several longer discussions, with the largest thread containing 63 posts and 10 identified users.

User participation is similarly imbalanced. A small number of highly active contributors account for a substantial portion of the activity. For example, Lewis Wood contributed 804 posts across 488 threads, Jacob Kruger contributed 586 posts across 349 threads, and Jim Homme contributed 286 posts across 205 threads. In contrast, many users appear only once or twice throughout the corpus, indicating a long-tailed participation distribution.

At the interaction level, most threads involve only a small number of identified users: 845 threads contain posts from a single identified user, 570 contain two users, and 311 contain three users. Only a limited number of threads develop into larger multi-user discussions, with up to 14 identified users in a single thread. Additionally, 242 threads contain no detected known user, likely due to missing or inconsistently parsed author metadata rather than genuinely anonymous participation.

5 Findings and Discussion

In this section, we interpret the conceptual clusters obtained during the previous stage of the analysis. For each cluster, we selected up to 20 representative discussion threads whose semantic embeddings were closest to the

Table 2. Summary statistics of the reconstructed corpus.

Statistic	Value
Total reconstructed threads	2,321
Longest thread length	63 posts
Maximum users in one thread	14 users
Threads with 1 post	692
Threads with 2 posts	319
Threads with 3 posts	328
Threads with 1 identified user	845
Threads with 2 identified users	570
Threads with 3 identified users	311
Threads with no identified user	242
Most active contributor	Lewis Wood (804 posts / 488 threads)
Second most active contributor	Jacob Kruger (586 posts / 349 threads)
Third most active contributor	Jim Homme (286 posts / 205 threads)

Table 3. Summary statistics of the semantic clusters. For each cluster, we report the number of discussion threads (N1), the number of unique contributors (N2), the total number of posts (N3), and the median number of words per contributed message (N4, excluding quoted replies).

Conceptual cluster	Threads (N1)	Unique contributors (N2)	Posts (N3)	Median words/post (N4)
C0	36	24	158	75.0
C1	81	29	142	114.5
C2	63	66	283	89.0
C3	97	72	317	68.0
C4	82	72	350	61.0
C5	149	103	667	79.0
C6	51	54	130	46.5
C7	41	66	194	92.0
C8	124	78	501	81.0
C9	127	124	553	92.0

cluster centroid. This allowed us to inspect semantically central examples while reducing the influence of outlier discussions. The selected threads were then manually reviewed and iteratively interpreted to identify recurring accessibility barriers, development practices, and adaptive strategies discussed by blind programmers.

3 summarizes the quantitative characteristics of each conceptual cluster, including the number of discussion threads (N1), unique contributors (N2), total contributed messages (N3), and the median number of words per contributed message (N4), excluding quoted replies. These statistics provide additional context regarding the relative level of community engagement represented by each cluster before presenting the qualitative interpretations below.

The resulting conceptual clusters were subsequently synthesized into a smaller set of higher-level findings capturing broader patterns that emerged across multiple clusters. The subsection titles presented below therefore represent interpretive themes derived from representative cluster samples rather than automatically generated categories. We first present clusters directly related to programming and software-development workflows, followed by clusters reflecting broader community and accessibility-related discussions.

5.1 Programming and software-development related topics

Cluster 0: Tool interoperability and configuration barriers in web-development environments This cluster contains 36 discussion threads and captures recurring problems encountered by blind developers when configuring and connecting modern web-development tools, particularly Visual Studio or VS Code, Node-based frameworks, local servers, and deployment or containerization tools. Several representative threads concern local execution and deployment issues, including localhost setup, IIS publishing, Azure-related publishing requirements, and launch-configuration errors (threads 1, 5, 6, 7, 9, 10, 11, 12, 14, 23, 25, 27). A second group of threads concerns the accessibility of frontend frameworks and their interaction with screen readers such as JAWS and NVDA, especially in React, Vue, and React Native projects (threads 1, 8, 13, 26, 28, 29, 30). Other examples focus on dependency-management and environment-configuration failures involving Babel, ESLint, npm imports, Docker integration, and file-system inconsistencies (threads 16, 17, 18, 19, 20, 31, 33, 35, 36). Overall, the cluster suggests that accessibility barriers in web-development environments often emerge not from a single inaccessible

interface, but from the difficulty of making multiple tools, frameworks, configuration files, and deployment targets work together predictably within non-visual workflows.

Cluster 2: Auditory accessibility and configurable non-visual feedback in development environments This cluster contains 15 discussion threads and captures the role of auditory feedback, audio cues, tones, and spoken notifications in non-visual programming workflows. Many representative threads discuss audio cues integrated into Visual Studio and VS Code for communicating errors, warnings, breakpoints, progress status, debugging information, and editor events (threads 1, 2, 3, 4, 6, 7, 11, 13, 14, 15, 16, 19, 20). Several discussions specifically focus on the need for configurable auditory feedback, including requests to control audio cue timing, reduce excessive spoken alerts, prioritize sounds differently, or disable intrusive notifications (threads 3, 6, 7, 12, 14, 16, 19). Other threads explore broader forms of non-visual interaction, including sonification, audio-based debugging workflows, audio breakpoint systems, and accessible sound-based development tools (threads 10, 18, 20). The cluster also includes examples of accessibility limitations in surrounding audio ecosystems, such as inaccessible headphone software and limitations of VoiceOver-based workflows on macOS (threads 8 and 17). Overall, the cluster suggests that accessibility in programming environments extends beyond screen-reader compatibility alone and increasingly depends on configurable, low-latency, and workflow-aware auditory interaction mechanisms that support non-visual software development.

Cluster 3: Accessible onboarding and workflow adaptation in modern IDEs This cluster contains 15 discussion threads and captures the challenges blind developers face when learning, configuring, navigating, and adapting to modern IDE workflows, particularly in VS Code and Visual Studio. Several representative threads focus on onboarding and discoverability barriers, including difficulties creating files, locating commands, configuring extensions, understanding keyboard workflows, and learning debugging features (threads 3, 4, 5, 6, 17, 19, 20). Other discussions concern accessibility regressions, inconsistent screen-reader behavior, inaccessible extensions, and platform-specific usability problems in IDE environments (threads 1, 7, 8, 10, 11). The cluster also contains examples of community-driven accessibility learning through tutorials, workshops, feature requests, GitHub discussions, and peer support (threads 3, 6, 12, 14, 15). Overall, the cluster suggests that accessibility in modern IDEs depends not only on screen-reader compatibility, but also on the discoverability, learnability, and workflow integration of development features within non-visual programming environments.

Cluster 4: Screen-reader interoperability in professional developer tools This cluster contains 20 discussion threads and captures interoperability problems between screen readers such as JAWS and NVDA and professional development tools including Visual Studio, VS Code, IntelliJ, SPSS, SSMS, JMeter, and GitHub Copilot. Many representative threads describe situations in which IDE features, debugging interfaces, menus, access keys, navigation controls, or editor indicators are either partially inaccessible or incorrectly exposed to screen readers (threads 1, 2, 4, 5, 6, 7, 8, 10, 11, 13, 17, 19, 20). Several discussions specifically concern failures in non-visual navigation and interaction, including inaccessible menus, unreadable debug information, broken cursor routing, inconsistent auditory feedback, and screen readers failing to announce the currently navigated line or interface state (threads 1, 6, 8, 11, 13, 17, 20). Other threads focus on the use of JAWS scripting, custom accessibility scripts, and manual workaround techniques to compensate for missing or unreliable accessibility support in developer tools (threads 9, 14, 16, 18). Overall, the cluster suggests that accessibility in professional programming environments depends not only on whether screen readers technically support a tool, but also on the stability, completeness, and predictability of interaction between assistive technologies and complex software-development interfaces.

Cluster 5: Accessibility barriers in terminal and console-based development workflows This cluster contains 20 discussion threads and captures accessibility barriers encountered by blind programmers when interacting with terminal environments, command-line interfaces, integrated IDE consoles, and debugger output systems. Many representative threads describe situations in which screen readers fail to correctly read terminal output, command-line interaction, debugger information, or dynamic console content in environments such as PowerShell, VS Code terminals, Docker containers, Rust tooling, and Jupyter notebooks (threads 1, 2, 3, 4, 5, 6, 8, 13, 20). Several discussions focus on navigation and interaction failures in integrated terminal workflows, including inaccessible debugger interaction, broken keyboard navigation, focus-management problems, and unreadable output regions in IDE-based consoles (threads 2, 6, 10, 13, 15, 17). Other threads concern broader technical limitations involving dynamic UI updates, event interception, WebView accessibility, unreliable screen-reader detection, and inconsistent assistive-technology behavior in interactive interfaces (threads 7, 11, 12, 19). The cluster also contains repeated examples of adaptive workaround strategies, including switching to Linux console environments, modifying NVDA timing settings, avoiding built-in terminals, or relying on external visual assistance tools such as Be My Eyes (threads 6, 10, 14, 18, 20). Overall, the cluster suggests that terminal and console-based workflows remain a major source of accessibility friction in modern software-development environments, particularly when dynamic output, interactive state changes, and integrated debugging interfaces are not reliably exposed to non-visual interaction technologies.

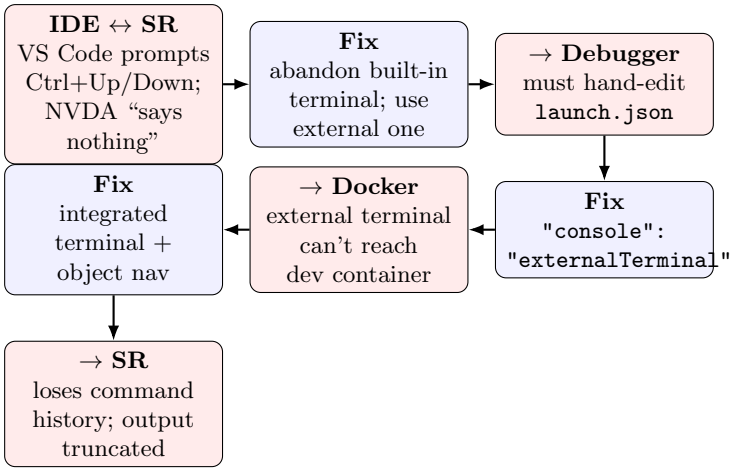


Fig. 1. Each accessibility fix relocates the barrier into the next tool. No single component is inaccessible; the failure is in their composition. Red: interoperability failure. Blue: workaround. Source: Program-L thread *"how to navigate terminal in vscode with nvda?"* (Jan. 2023).

Illustrative examples: tools that do not interoperate. The claim that accessibility depends on tools working *together* is best shown by cases in which no single component is inaccessible, yet their composition fails. Figure 1 traces one such case (11 posts, Jan. 2023): VS Code instructs the user to press Ctrl+Up/Down to read terminal output, but the screen reader "says nothing or says listItem"; the community fix is to abandon the built-in terminal ("don't use built-in terminal, it's not accessible"), which then requires editing `launch.json` to reconnect the debugger, which in turn breaks under Docker dev containers, forcing a return to object navigation that loses command history and truncates output. Each remedy pushes the barrier one tool deeper.

The same pattern recurs across the corpus. In a second thread (4 posts, Nov. 2024), a developer reports that "NVDA isn't reading the text inside the VS Code's Console, but Be My Eyes confirms it's there": the output is present and rendered, so the failure lies squarely in the console–screen-reader interaction rather than in missing content. A third thread (8 posts, Nov. 2022) shows the failure spanning a shared team toolchain: the standard API client is inaccessible ("that app is not accessible"), and the accessible substitute does not slot into the existing workflow, costing the developer parity with sighted colleagues. Finally, a fourth thread (5 posts, June 2023) illustrates the end state of repeated interoperability failure—developers exit the integrated environment altogether for a terminal-native stack that composes reliably: "I use vim and terminal which has always been accessible." Together these cases indicate that the barrier is systemic: it emerges from the seams between the IDE, terminal, debugger, configuration files, containers, and assistive technology, not from any one inaccessible interface.

Cluster 7: Accessibility inclusion versus specialized development tooling This cluster contains 20 discussion threads and captures broader discussions about how accessibility should be integrated into modern software-development ecosystems for blind and low-vision programmers. Several representative threads explicitly critique the idea of separate accessibility-specific development tools and instead argue for improving the accessibility of mainstream IDEs, frameworks, and workflows used by sighted developers as well (threads 4, 5, 10). Other discussions focus on the relationship between accessibility, usability, and workflow parity, emphasizing that technically accessible tools may still remain cumbersome, inefficient, or socially isolating in professional programming environments (threads 1, 3, 15, 18, 19). The cluster also contains repeated examples of inaccessible UI frameworks, limited accessibility documentation, inaccessible GUI tooling, and challenges adopting alternative development ecosystems (threads 12, 16, 17, 18). In parallel, several threads concern ongoing research efforts investigating accessible programming workspaces, UI development environments, block-based programming systems, and accessibility support for developers with visual impairments (threads 2, 6, 7, 13). Overall, the cluster reflects broader community-level discussions about inclusion, usability, and long-term participation of blind developers within mainstream software-development ecosystems rather than focusing on a single technical accessibility barrier.

Cluster 9: Educational and professional inclusion of blind developers in technical ecosystems This cluster contains 20 discussion threads and captures broader accessibility concerns related to programming education, workforce participation, technical training, and long-term inclusion of blind and low-vision developers within software-development ecosystems. Several representative threads focus on accessible learning materials, programming roadmaps, technical training opportunities, and the accessibility of advanced domains such as machine learning, system programming, and web development for visually impaired developers (threads 1, 7,

13, 14, 18). Other discussions concern research studies and community initiatives investigating the experiences, workflows, and accessibility needs of blind software professionals (threads 2, 3, 4, 6, 8, 11). The cluster also includes examples related to employment opportunities, accessibility-focused workforce training, and collaborative development environments involving blind developers (threads 12, 15, 20). Overall, the cluster reflects broader concerns about educational access, career participation, and long-term inclusion within technical professions rather than focusing on a single accessibility barrier in programming tools themselves.

5.2 AI-assisted development: mitigation and relocation

The clusters above are fitted over the whole corpus and are period-agnostic. Because generative coding assistants became widely available part-way through the window, we additionally split the corpus at March 2024, identifying mentions four ways: by a validated lexicon at thread and at post level, by the `tools_mentioned` field our extraction stage already produces, and by whether a thread's assigned `code` or `theme` names an AI tool. Section 6.7 gives the construction; the result does not depend on the choice.

The trend is large and diffuse. At thread level, 227 threads (9.6%) mention an AI coding tool: 3.77% before March 2024 against 19.21% after, a factor of 5.1 ($\chi^2 = 151.7$, $p < 10^{-34}$). At post level, 494 of 11,343 posts: 1.45% against 9.17%, a factor of 6.3 ($\chi^2 = 378.8$, $p < 10^{-83}$). The post-level proportion is necessarily the lower, since a flagged thread contains many unflagged messages; both series appear in Table 9. The remaining two measures give 5.3 and 4.2 (Table 9). We report proportions because posting volume falls by roughly half across the window, from 3,039 posts in 2022 to 1,779 in 2025.

The rise is community-wide rather than the interest of a few prolific contributors: the share of active authors writing at least one AI-mentioning post rose from 14% (42 of 309) to 34% (91 of 267), and excluding the three most active authors leaves the post-level contrast slightly larger (1.7% against 9.9%).

What changed was not that AI arrived, but where it arrived. Composition is more informative than volume (Table 4). First-half discussion is almost entirely about general-purpose chatbots used outside the development environment; Gemini, Claude and agentic command-line tools are effectively absent. In the second half these appear from a base of almost nothing and Copilot mentions rise more than fourfold. The tools moved from a browser tab into the editor, the terminal and the debugger — precisely the composition points that Figure 1 identifies as the locus of accessibility failure.

Mitigation and new barriers appear in equal measure. Because our coding stage assigns each thread a code and a theme, the direction of these discussions can be read from our own analytic output rather than imposed. Of the 68 dated threads whose code or theme names an AI tool, themes indicating that the AI tool is itself inaccessible rise from 3 to 11 across the two halves; themes treating AI as an accessibility *solution* rise from 3 to 9; themes concerning reliability or deskilling are near-static, 4 against 5. The generated themes state both poles plainly: against *Inaccessible AI development tools* and *Screen reader incompatibility with AI chat interfaces* stand *Leveraging AI for visual accessibility workarounds* and *AI enables non-visual UI/UX creation.* Cell counts are small and the grouping is keyword-derived; we report it as a first pass and release the 74 threads concerned for manual verification.

The mitigation cases are concrete, and one closes a loop within our own data. In a thread on the Playwright Inspector (Nov. 2024, 5 posts) a developer sidesteps the inaccessible inspector by handing the HAR file to a language model. A 36-post thread on BIOS access (Nov. 2024) returns to the problem behind the BIOS row of Table 1, where the community's answer had been an HDMI capture card, and now weighs that hardware workaround against AI visual description: within one corpus, one class of workaround displacing another.

Relocation: the same failure, one tool generation later. Against this, the second half documents new interoperability failures with exactly the structure of Figure 1. A thread on comparing Git branches (Oct. 2024) reports that focus transfer into the diff editor is nondeterministic — sometimes focus moves and it works, sometimes a suggestion is announced that the user cannot then locate. A thread whose subject line states the problem in full, "Gemini provides code in code windows but cannot get to them" (Sep. 2025, 16 posts), describes an interactive code pane that is rendered, persistent, editable and unreachable. Two further threads (Sep. and Dec. 2025) report that VS Code's dedicated accessible view for Copilot output renders HTML entities literally, so that operators and quotation marks in generated code arrive as `<`, `>` and `"`.

In none of these is the model output itself inaccessible: the content is present and correct throughout, and the failure lies at the seam between assistant, editor pane, accessible view and screen reader. Several defects are moreover introduced *by* accessibility features rather than by their absence. Adaptation follows the same pattern as elsewhere in the corpus, now applied to a new class of tool: one developer asks the assistant to prefix every response with an HTML heading so the screen reader can navigate between turns (Dec. 2025); another discovers the `alt+F2` accessible-view shortcut by prompting Gemini for screen-reader-friendly instructions (Feb. 2026), the assistant serving as the discovery mechanism for its own accessibility affordance.

Table 4. AI-tool mentions before and after March 2024 (posts, quoted text excluded; a post may mention more than one tool). A further 16 and 85 posts respectively refer to an unnamed model or AI practice. General-purpose chatbots dominate the first half; editor-resident and command-line agentic tools appear almost entirely in the second.

Tool	H1 (2022-03–2024-02)	H2 (2024-03–2026-03)
ChatGPT / OpenAI	62	104
GitHub Copilot	24	110
Gemini	0	118
Claude	1	86
CLI / agentic tools	0	48
Cursor	0	3
Posts mentioning any AI tool	103	391
Total posts in period	7,081	4,262

Table 5. Resolution status of the 1,585 filtered threads, from the `solutions` and `unsolved_issues` fields of the extraction stage.

Outcome	Threads	Share
Resolved (solution, no residual issue)	245	15.5%
Partial / workaround (solution + open residue)	871	55.0%
No solution proposed	469	29.6%
Any solution proposed	*1,116*	*70.4%*

What this cannot establish. The analysis measures salience, not net benefit. A mailing list is complaint-selected: a workflow that works generates no thread, so successful mitigation is under-reported. The fall in the share of clustered threads in the terminal and console cluster across the two halves (20.3% to 14.0%) therefore cannot be read as terminals becoming accessible, particularly as the professional-tools interoperability cluster rises over the same period (7.4% to 14.0%). Nor are the four measures equally unbiased: our relevance filter discards AI-mentioning threads at an increasing rate over time, making the code-based factor of 4.2 a lower bound and the lexicon's 5.1 the less biased estimate.

5.3 How often are issues resolved?

Our extraction stage records a `solutions` and an `unsolved_issues` field per thread, which lets us ask how often a discussed barrier is resolved. We do not attempt the related question of whether an issue is novice- or expert-typical: the archive records participation but not expertise, and 26.9% of thread openers are unattributed, so any such split would rest on a weak proxy. Whether a known community answer exists is instead observable in the thread.

Across the 1,585 filtered threads, 70.4% surface at least one proposed solution, but only 15.5% reach clean resolution with no residual open issue; 55.0% offer a fix that leaves an open problem behind, and 29.6% record no solution at all (Table 5). Engagement is high but closure is rare, and the dominant outcome is a fix that relocates the barrier rather than removing it—the interoperability pattern of Figure 1.

Two measures independent of the LLM fields bound this rate. Explicit in-thread confirmation (*"that was it"*, *"I figured it out"*), with negations filtered, appears in 94 of 2,378 reconstructed threads (4.0%)—a strict lower bound, since confirmed fixes are rarely stated. Independently, 692 threads (29.1%) receive no reply at all, matching the 29.6% no-solution rate. Both bracket a low true resolution rate; the complaint selection of Section 5.2 makes $\approx$16% a floor.

5.4 Broader community and accessibility-related discussions

Cluster 1: Accessible STEM publishing and technical communication This cluster contains 82 discussion threads and captures community-led efforts around accessible STEM publishing, TeX/LaTeX workflows, arXiv accessibility, and technical-document communication. Many representative threads are TeX Hour announcements or updates in which accessibility, typography, technical documentation, PDF accessibility, LaTeX workflows, and STEM communication are recurring themes (threads 1, 2, 3, 4, 6, 7, 10, 12, 15). These interpretations are grounded in repeated references to accessibility-related TeX discussions, technical-document workflows, and community-organized events focused on STEM accessibility (threads 1, 2, 4, 8, 9, 11, 13, 18).

A second group of threads focuses more directly on accessible scholarly infrastructure, particularly arXiv accessibility, accessible scientific papers, and the future of TeX in STEM publishing (threads 1, 5, 13, 14, 16, 19,

20). Several discussions specifically mention accessible outputs for arXiv papers, accessible HTML and tagged PDF generation, software reprocessing of TeX documents, and user research related to scientific accessibility (threads 11, 14, 16, 19, 20).

Additional threads concern accessible mathematical and technical content, including accessible tables, mathematical notation, STEM learning materials, and accessible online mathematics resources (threads 8, 9, 11, 17, 18). These discussions frequently address accessibility challenges in mathematical notation, scientific learning materials, and technical-document presentation for blind users (threads 8, 9, 17, 18).

Overall, the cluster suggests that accessibility concerns in programming-adjacent technical communities extend beyond code editors and development environments to include the production, transformation, dissemination, and long-term accessibility of scientific and mathematical knowledge. However, because this cluster is less directly centered on programming workflows themselves and focuses more broadly on scholarly communication and STEM publishing infrastructure, we report it separately from the central findings of the paper.

Cluster 6: Community feedback and accessibility support practices This cluster contains 20 discussion threads and captures community-driven accessibility feedback, support coordination, and communication practices among blind and low-vision developers. Many representative threads concern the collection and exchange of accessibility feedback through surveys, issue reporting, direct developer outreach, and user discussions about accessibility barriers in tools and platforms (threads 1, 3, 4, 5, 10). Other discussions focus on community-mediated support practices, including help requests, off-list coordination, troubleshooting exchanges, and collaborative problem resolution within developer communities (threads 7, 13, 14, 16, 17). Several threads also concern communication accessibility itself, including accessible notifications, clearer tool descriptions, and usability evaluations of platforms such as Google Colab and other development-related services (threads 6, 12, 15, 18, 19, 20). Overall, the cluster suggests that accessibility in programming communities depends not only on technical tool support, but also on continuous feedback exchange, collaborative support practices, and active participation of blind developers in evaluating and shaping accessible software ecosystems.

Cluster 8: Community-driven programming meetups for blind and low-vision developers This cluster contains 20 discussion threads and primarily consists of recurring announcements and discussions related to the "Touching Python" Zoom meetup group for blind and low-vision programmers. The threads emphasize peer learning, community support, professional networking, accessibility discussions, and knowledge sharing across programming topics including Python, AI, C++, Kotlin, and accessibility technologies (threads 1–20). Unlike several other clusters that focus on concrete accessibility barriers in programming workflows, this cluster primarily reflects the importance of community-driven technical spaces for collaboration, learning, and mutual support among visually impaired developers. The cluster therefore highlights the social and community dimensions of accessibility within programming ecosystems.

The complete set of conceptual clusters is shown in Table 6.

6 Analysis Pipeline and Reproducibility

This section summarizes the main processing stages used in the analysis pipeline together with the corresponding intermediate outputs and dataset sizes. The purpose of this appendix is to provide transparency regarding the transformations applied to the original mailing-list archive prior to the final thematic and conceptual analysis.

6.1 Pipeline Overview

The analysis pipeline consisted of the following major stages:

1. Reconstruction and preprocessing of discussion threads from the Program-L archive.
2. LLM-assisted filtering of threads relevant to programming accessibility and developer tooling.
3. LLM-assisted extraction of accessibility-related codes, themes, and supporting evidence.
4. Semantic embedding of thematic summaries using a sentence-transformer model.
5. Dimensionality reduction using UMAP.
6. Conceptual clustering using HDBSCAN.
7. Manual interpretation of representative cluster examples.

6.2 Worked example: tracing one thread through the pipeline

To make the operations above concrete, we trace a single discussion thread from the raw archive to its final conceptual cluster. We use the thread *"IntelliJ with JAWS"* (thread id `IntelliJ-with-JAWS`), which is also the source of the IntelliJ row in Table 1.

Cluster	Conceptual cluster title	Representative themes
C0	Accessible web development and deployment workflows	Development environment configuration challenges; local deployment challenges; IDE and tooling accessibility challenges; framework onboarding challenges
C1	Accessible STEM publishing and technical communication	Promoting accessibility in STEM publishing; accessible mathematical content; community-led technical accessibility discussions; scholarly accessibility initiatives
C2	Auditory accessibility in development environments	Auditory accessibility in IDEs; accessibility feedback customization; audio debugging workflows; accessible non-visual feedback
C3	Accessible workflows and onboarding in VS Code	IDE accessibility improvements; onboarding barriers; accessible debugging workflows; AI-assisted development accessibility
C4	Screen-reader interoperability in professional developer tools	Accessibility gaps in IDEs; assistive technology scripting challenges; debugging accessibility; screen-reader compatibility issues
C5	Accessibility barriers in terminal and console-based workflows	Integrated terminal accessibility; screen-reader interaction failures; console output accessibility; command-line workflow adaptations
C6	Community feedback, support, and accessibility reporting	User feedback and support; accessibility surveys; communication practices; accessibility issue reporting
C7	Accessibility inclusion versus specialized tooling	Accessibility in mainstream development tools; accessible programming workspaces; debates on specialized versus mainstream tools; adaptive programming workflows
C8	Community-driven programming meetups and peer learning	Accessible tech community building and learning; inclusive professional networking and support; peer learning among visually impaired developers; community-driven programming collaboration
C9	Accessibility support and inclusive participation in technical ecosystems	Community support for programming accessibility; inclusive development practices; accessibility-oriented technical collaboration; improving accessibility in software-development environments

Table 6. Complete set of higher-level conceptual clusters identified during semantic clustering. For each semantic cluster, representative discussion threads were selected according to cosine similarity to the cluster centroid. These representative examples were manually reviewed in order to derive higher-level conceptual interpretations and recurring accessibility concerns grounded in the original discussion evidence.

Stage 1 — Reconstruction. The archive stores this discussion as 40 separate message pages sharing the subject identifier `IntelliJ-with-JAWS`. Grouping by that identifier and stripping quoted reply chains and archive boilerplate yields one thread record with 40 posts, 7 identified contributors, and a 15,828-character `discussion_text` field. The thread opens with a direct question: *"In your experience, how accessible is the latest IntelliJ with JAWS out of the box?"*

Stage 2 — Relevance-bearing extraction (LLM pass 1). The reconstructed thread text is inserted into the extraction prompt and sent to Gemini 2.5 Flash, which returns structured JSON containing a summary, the main problem, proposed solutions, unresolved issues, keywords, tools mentioned, screen readers mentioned, a topic phrase, and a confidence score. For this thread the summary records that NVDA is generally preferred over JAWS for IntelliJ because of better Java Access Bridge support, and the extracted evidence includes the verbatim sentence *"The only two main problems I have found with jaws is the review cursor doesn't work in the error or help lists."*

Stage 3 — Coding (LLM pass 2). The extraction output is passed to a second Gemini call that condenses it into a single qualitative code, a broader theme, and one or two supporting quotations. For this thread the result is:

> **code:** JAWS Java Access Bridge (JAB) support issues in IntelliJ
> **theme:** Accessibility Gaps in Developer Tools
> **evidence:** "...the review cursor doesn't work in the error or help lists." / "That is because Jaws still does not read all the access keys correctly in the JAB."

Table 7. The thread *"IntelliJ with JAWS"* at each pipeline stage.

Stage	State of the item
Raw archive	40 message pages sharing one subject identifier
Reconstruction	1 thread; 40 posts; 7 contributors; 15,828 characters
Extraction (LLM 1)	JSON: summary, main problem, solutions, unresolved issues, keywords, tools, screen readers
Coding (LLM 2)	1 code + 1 theme + 2 verbatim quotations
Filtering (LLM 3)	*keep* — core problem is IDE screen-reader interaction
Embedding	768-dim unit vector from code + theme + evidence
UMAP	15-dim vector, neighbourhood structure preserved
HDBSCAN	cluster C4 (of 10); not noise
Representative selection	distance to centroid 0.174; rank 1 of 82
Interpretation	one of 20 threads read manually to author the C4 narrative

Stage 4 — Relevance filtering. A further LLM classification stage judges each coded thread against the corpus goal, returning *keep*, *maybe*, or *drop*. This thread is retained because its central problem concerns screen-reader interaction with an IDE. Threads whose core topic lies elsewhere — for example disability law or mailing-list hosting — are dropped even when they mention a development tool in passing. This stage reduces the corpus from 2,344 coded threads to 1,585.

Stage 5 — Embedding. The code, theme, and up to two evidence quotations are concatenated into a short block of text, which is the unit that is actually encoded — the original 40 posts are not re-read at this point. The `all-mpnet-base-v2` sentence-transformer maps this block to a 768-dimensional vector, normalised to unit length, so that threads expressing similar accessibility problems in different words come to lie close together.

Stage 6 — Dimensionality reduction. UMAP, a non-linear dimensionality-reduction method that compresses vectors while preserving which points are neighbours of which, reduces the 768 dimensions to 15 using a cosine metric (`n_neighbors` = 30).

Stage 7 — Clustering. HDBSCAN, a density-based algorithm that finds groups of mutually close points and leaves sparse points unassigned, groups the reduced vectors (`min_cluster_size` = 15, `min_samples` = 5). Our thread is assigned to cluster C4 (82 threads). Had it been semantically isolated it would have received the label −1 and been excluded, as happened to 730 of the 1,585 threads.

Stage 8 — Representative selection and interpretation. Within C4 the cosine distance from this thread's embedding to the cluster centroid is 0.174 — the smallest in the cluster, making it the rank-1 representative. It is therefore among the 20 centroid-nearest threads we read manually to write the interpretation of C4 in Section 5. The narrative for that cluster — screen-reader interoperability in professional developer tools — is thus grounded in this thread and 19 others like it, not in the full 82.

Table 7 summarises the transformation at each stage.

6.3 Intermediate Outputs and Corpus Sizes

Table 8 summarizes the number of threads remaining after each stage of the processing pipeline.

Pipeline stage	Intermediate output	Count
Reconstructed discussion threads	`threads_for_llm_all.json`	2378
Successful Gemini outputs	`gemini_outputs_all/`	2344
Gemini failures / parsing failures	`gemini_failed_all/`	100
Filtered accessibility-related thematic outputs	`gemini_thema_analysis_out/`	1585
Valid thematic files loaded for clustering	semantic clustering input	1583
Threads assigned to semantic clusters	cluster $\neq -1$	851
Noise / unassigned threads	cluster $= -1$	732

Table 8. Intermediate outputs and corpus sizes throughout the analysis pipeline.

Table 9. The March 2024 contrast under four operationalisations. Levels differ because the unit and denominator differ — posts are finer than threads, and the coded set contains only threads surviving relevance filtering — but direction and approximate magnitude do not.

Measure	Unit	H1		H2		H2/H1
Lexicon (primary)	thread	56/1,485	(3.77%)	171/890	(19.21%)	5.1
Lexicon	post	103/7,081	(1.45%)	391/4,262	(9.17%)	6.3
Extraction `tools_mentioned`	thread	51/1,432	(3.56%)	159/844	(18.84%)	5.3
Assigned `code`/`theme`	thread	20/979	(2.04%)	48/553	(8.68%)	4.2

6.4 LLM-Assisted Corpus Filtering

Prior to thematic coding and semantic clustering, the corpus was filtered using an LLM-assisted relevance classification stage. The filtering process retained only threads containing substantial evidence related to accessible programming workflows, developer tooling accessibility, non-visual coding practices, screen-reader interaction with programming environments, and AI-assisted coding accessibility.

Threads whose primary focus was unrelated to programming-tool accessibility (e.g., general social discussions, legal topics, or unrelated accessibility discussions) were excluded from subsequent thematic analysis.

6.5 Thematic Coding Pipeline

For each retained discussion thread, an LLM-generated thematic summary was produced containing:

- a candidate accessibility-related code,
- a candidate theme,
- supporting evidence quotations extracted from the thread.

These thread-level summaries were subsequently embedded and clustered in order to identify higher-level conceptual patterns across the corpus.

6.6 Semantic Clustering Parameters

The conceptual clustering stage used the following configuration:

- Sentence embedding model: `all-mpnet-base-v2`
- Embedding normalization: cosine normalization
- UMAP neighbors: 30
- UMAP components: 15
- UMAP metric: cosine distance
- HDBSCAN minimum cluster size: 15
- HDBSCAN minimum samples: 5
- Representative examples per cluster: 20

Threads assigned the HDBSCAN label -1 were treated as semantically heterogeneous or weakly connected outliers and were therefore excluded from the main conceptual clusters.

6.7 Measuring AI-tool mentions

Section 5.2 uses four operationalisations at two units of analysis (Table 9): a lexicon of tool names and AI-assisted development practices applied at thread level over subject lines and message bodies — the primary measure, consistent with Tables 2 and 3; the same lexicon at post level over bodies alone; the `tools_mentioned` field produced by the extraction stage of Section 3; and whether the assigned `code` or `theme` names an AI tool.

7 Conclusion

We set out to ask how programming environments can be better adapted to the needs of blind developers, and analysed four years of the Program-L archive to answer it. The specific findings are laid out above; here we state what we take to be the single issue underlying them.

Across the corpus, the barrier is almost never a single inaccessible interface. Only 15.5% of discussed issues reach clean resolution, while 55.0% end in a fix that leaves an open problem behind, and the recurring shape of those fixes—abandon the built-in terminal, hand-edit `launch.json`, drop to a Linux console, paste the output

into a language model—is not a set of solutions but a set of *relocations*: each remedy pushes the barrier one tool deeper (Figure 1). The AI-assisted period reproduces the same structure one tool generation later, with rendered-but-unreachable code panes and accessible views that corrupt the very output they exist to expose. The consistent thread is that the information the developer needs already exists, in correct and structured form, and is lost at the boundary between tools rather than being absent from any one of them.

This locates the problem more precisely than "tools are inaccessible." A screen reader can only voice what an application publishes to the platform accessibility tree; when a terminal renders to a canvas, or a debugger panel or an AI code pane is never exposed as accessible text, the information is dropped at the *provider* stage of the pipeline, before any assistive technology can reach it. The screen reader is not at fault and cannot be the site of repair, because it is downstream of the loss. Yet in nearly every case the same information survives upstream of the inaccessible rendering—in the pseudo-terminal buffer, the Debug Adapter Protocol stream, the raw model response—which is exactly why the community's most durable workarounds are, in effect, ad hoc taps into those upstream sources. The central unmet need this archive reveals is therefore not better screen-reader compatibility for individual tools, nor better description of what has already been lost, but a principled way to recover structured information at its source and route it through an accessible channel that bypasses the broken rendering entirely.

This reframing also situates our findings against prior work. Recent studies characterise the AI-accessibility problem as one of verification (Cha et al., 2026), interaction overload (Flores-Saviaga et al., 2025; Qiu et al., 2026), or generated-code quality (Mowar et al., 2025). Each documents that information fails to reach the blind developer, but treats the accessibility layer as opaque and does not ask where, mechanically, the information is lost or in what form it still exists. Our contribution is to name that stage and to show, from naturally occurring evidence at scale, that it is the common cause behind barriers previously treated separately.

Two limitations bound these claims. A mailing list is complaint-selected: a workflow that works generates no thread, so the resolution rate is a floor and successful adaptation is systematically under-reported. Participation is also skewed towards a small number of highly active contributors and a Windows/NVDA/VS Code milieu, so the specific tools named are not a neutral sample of the field. Neither limitation affects the structural finding, which concerns where in the pipeline information is lost, not how often a given tool is mentioned.

We therefore read this archive not as a catalogue of broken tools but as evidence for a single design principle: for the dead-end cases blind developers report—terminal, debugger, AI output—the needed information is recoverable upstream of the rendering that loses it, and the most promising direction for adaptive programming environments is to intercept it there. The debugger is the cleanest first test of this principle, since the Debug Adapter Protocol already carries, as structured messages, everything the inaccessible debug panel displays. We leave its construction to future work, but the case for building outside the translation chain, rather than repairing links within it, is the clearest conclusion this analysis supports.

References

Cha, Y., Figueira, I., Ayala, J., Edwards, E. J., Garcia, J., Van Der Hoek, A., & Branham, S. M. (2024). Do You Want Me to Participate or Not?": Investigating the Accessibility of Software Development Meetings for Blind and Low Vision Professionals. In *Proceedings of the 2024 chi conference on human factors in computing systems.*

Cha, Y., Jackson, V., Shu, L., Branham, S. M., & van der Hoek, A. (2026). "game changer" or "overenthusiastic drunk acquaintance"? generative AI use by blind and low vision software professionals in the workplace. In *Proceedings of the 48th ieee/acm international conference on software engineering (icse).* ACM. doi: 10.1145/3744916.3773172

Chen, N., Qiu, L. K., Wang, A. Z., Wang, Z., & Yang, Y. (2026). Screen reader programmers in the vibe coding era: Adaptation, empowerment, and new accessibility landscape. In *Proceedings of the 2026 chi conference on human factors in computing systems.* ACM. (Preprint: arXiv:2506.13270) doi: 10.1145/3772318.3790726

Ferrari, C., & Hurst, A. (2021). Accessible web development: Opportunities to Improve the Education and Practice of Web Development with a Screen Reader. *TACCESS*.

Flores-Saviaga, C., Hanrahan, B. V., Imteyaz, K., Clarke, S., & Savage, S. (2025). The impact of generative AI coding assistants on developers who are visually impaired. In *Proceedings of the 2025 chi conference on human factors in computing systems.* ACM. doi: 10.1145/3706598.3714008

Mealin, S., & Murphy-Hill, E. (2012). An Exploratory Study of Blind Software Developers. In *vl/hcc.*

Mowar, P., Peng, Y.-H., Steinfeld, A., & Bigham, J. P. (2024). Tab to autocomplete: The effects of AI coding assistants on web accessibility. In *Proceedings of the 26th international acm sigaccess conference on computers and accessibility (assets)*. ACM. doi: 10.1145/3663548.3688513

Mowar, P., Peng, Y.-H., Wu, J., Steinfeld, A., & Bigham, J. P. (2025). CodeA11y: Making AI coding assistants useful for accessible web development. In *Proceedings of the 2025 chi conference on human factors in computing systems*. ACM. doi: 10.1145/3706598.3713335

Naeem, M., Smith, T., & Thomas, L. (2025). Thematic Analysis and Artificial Intelligence: A Step-by-Step Process for using ChatGPT in Thematic Analysis. *International Journal of Qualitative Methods*.

Nguyen-Trung, K. (2025). ChatGPT in Thematic Analysis: Can AI Become a Research Assistant in Qualitative Research? *Quality & Quantity*.

Pandey, M., Bondre, S., O'Modhrain, S., & Oney, S. (2022). Accessibility of UI Frameworks and Libraries for Programmers with Visual Impairments. In *Vl/hcc*.

Pandey, M., Kameswaran, V., Rao, H. V., O'Modhrain, S., & Oney, S. (2021). Understanding Accessibility and Collaboration in Programming for People with Visual Impairments. *Proceedings of the ACM on Human-Computer Interaction*.

Potluri, V., Pandey, M., Begel, A., Barnett, M., & Reitherman, S. (2022). Codewalk: Facilitating Shared Awareness in Mixed-ability Collaborative Software Development. In *International acm sigaccess conference on computers and accessibility*.

Seo, J., & Rogge, M. (2023). Coding Non-visually in Visual Studio Code: Collaboration Towards Accessible Development Environment for Blind Programmers. In *International acm sigaccess conference on computers and accessibility*.

Wang, Q., Erqsous, M., Barner, K. E., & Mauriello, M. L. (2025). LATA: A Pilot Study on LLM-assisted Thematic Analysis of Online Social Network Data Generation Experiences. *Proceedings of the ACM on Human-Computer Interaction*.

Agency, Expertise, and the Limits of Augmentation in LLM-Assisted Programming Workflows

Gabriel Hughes
UCL Interaction Centre
University College London
gabriel.hughes@ucl.ac.uk

Yvonne Rogers
UCL Interaction Centre
University College London
y.rogers@ucl.ac.uk

Abstract

Large language models (LLMs) are being rapidly adopted for complex knowledge work, yet little is known about how end users are adapting these tools to their needs, why their customised workflows take the shape they do, and what those adaptations reveal about unmet needs in human-AI interaction. This paper reports findings from seventeen semi-structured interviews with knowledge workers, primarily software engineers, investigating how complementary human-AI relationships develop in practice, and whether users perceive LLMs to be genuinely augmentative. Expert software engineers had developed workflows that preserved their agency over the parts of their work they valued. These were characterised by meticulous upfront specification and iterative agent configuration, which kept strategic and creative decisions in human hands. Even within these expert workflows, however, participants described new cognitive burdens, particularly regarding context management, output evaluation, and maintaining strategic oversight, all of which current tool design does little to mitigate by default. These findings suggest that complementarity is not a default property of LLM-based tools but something users must deliberately construct, often in ways those tools do not naturally support.

1. Introduction

Large language models (LLMs), which power popular chatbots like ChatGPT and Claude, have over a billion users worldwide, yet their design has not evolved significantly past chatbot-based paradigms since they entered mainstream usage. Prompting a conversational interface via natural language remains the standard for engaging with AI, despite arguments that it is a poor substitute for genuine natural-language interaction (Kraljic & Lahav, 2024; Meredith Ringel Morris, 2024). Even interactions with LLM-powered coding tools are organised as linear chat conversations, whose structure, combined with the quantity of output LLMs create, produces significant information processing challenges for complex knowledge work like software development (Alami & Ernst, 2025; Suh et al., 2023; Tang et al., 2024). While there is substantial evidence these tools can genuinely augment complex knowledge work (Dell'Acqua et al., 2023; Mohamed et al., 2026), the ways in which they are changing the nature of work can come with significant cognitive costs (Ranganathan & Ye, 2026; Simkute et al., 2024; Tang et al., 2024). What remains under-explored is how practitioners are adapting to this tension in potential, and what their adaptations reveal about the conditions under which genuine augmentation becomes possible.

Research on end-user development has long shown that people with domain expertise will adapt tools to meet their needs when default designs fall short (Ko et al., 2011). LLM users appear to be doing exactly this. Sarkar and Drosos (2025) conducted one of the first ethnographic studies of 'vibe coding' (a popularised term for LLM-assisted programming), in which they analysed over eight hours of recorded expert coding sessions. They found that experienced developers are adapting LLM-based tools in various ways that both preserve their agency over their work and mitigate LLMs' shortcomings, notably the tendency for their quality to deteriorate as their context grows.

Each end-user adaptation reveals a potential new UX issue and, therefore, a design opportunity. Observational research like Sarkar and Drosos (2025) is therefore beneficial for understanding the details of an end-user-developed practice, but it leaves open questions regarding why those adaptations emerged and how attitudes and practices have developed over time. To that end, this research sought to answer the following research questions through seventeen semi-structured interviews with knowledge workers, primarily software engineers:

- **RQ1.** How do knowledge workers' custom adaptations of AI tools for complex workflows reveal unmet needs in human-AI collaboration?
- **RQ2.** What is the perceived potential for LLMs to augment human abilities?

The data suggest that a genuinely complementary relationship with LLM-based tools is not something the tools provide by default but something users must construct. The most experienced software engineers had developed sophisticated workflows for preserving agency over the parts of their work they valued while offloading less rewarding jobs, but these came with significant cognitive costs. Those costs, and the strategies needed to manage them, are least accessible to less experienced users, who lack both the technical skill to evaluate LLM-authored code output reliably and the metacognitive resources to manage effective delegation.

The primary contribution this research makes is to provide empirical support for the importance of promoting user agency and the enablement of intrinsically rewarding, or meaningful, work as key components of a more human-centred, or complementary, relationship with AI. While the traditional view of complementarity, centering on balancing the relative strengths and weaknesses of humans and technology, often with the goal of quantifying complementarity (Steyvers et al., 2022; Wilder et al., 2020), is still crucial, these interviews suggest that whether a relationship with AI is experienced as genuinely augmentative depends less on the balance of capabilities than on whether it allows the user to protect and advance the work they value. This paper treats complementarity in this richer sense as the condition under which LLM use is experienced as genuinely augmentative; the two concepts are distinct, but the data support their connection inductively. This perspective tends to be missing from much of the literature and technological development around LLMs, but is an important framing for future design work.

2. Methodology

2.1. Participants & Recruitment

Participants (N=17) were professional knowledge workers, mostly software engineers, with the exception of one journalist, one philosophy graduate, and a design educator. Table 1 shows details of each participant's profession and LLM expertise. Recruitment mostly followed a community-driven approach. Users of the open-source productivity software Obsidian were identified as a strong target, since a significant number already share their workflows online (e.g., on Discord, Reddit, and YouTube), and many were already using LLMs in conjunction with Obsidian. This initial group of Obsidian users contributed three participants. Two others were found on YouTube or X (f.k.a. Twitter) by searching for 'agentic coding' or 'Claude Code' (which was at the time, and still is, one of the most widely used LLM-based coding tools). These were developers who wrote about or made videos about their workflows. Nine participants were connections of other participants or UCL's network and did not publish content about their work online.

2.2. Procedure

Semi-structured interviews were chosen as the research method as a primary focus of the study was to uncover participant attitudes towards various aspects of interacting with AI, as well as their motivations for using AI tools in specific ways. These interviews lasted approximately 60 minutes each and were adapted to each participant based on a discussion guide. The discussion guide, and subsequent reflexive thematic analysis (Braun & Clarke, 2019), were largely structured around **RQ1** and **RQ2** and focused on how and why participants' LLM-assisted workflows took shape the way they did.

2.3. Analysis

The data consisted primarily of automated transcriptions via Microsoft Teams. Thirteen out of seventeen participants gave informed consent to have their interviews recorded. Data from the other four participants consisted of detailed notes taken by the interviewer. Data was analysed using reflexive thematic analysis (Braun & Clarke, 2019). This approach aims to understand implications emerging from qualitative data by identifying patterns and themes present across multiple participants. Themes were defined

PID	Profession	LLM Expertise*
P1	QA Engineer + Writer	Expert
P2	Software Engineer	Expert
P3	Software Engineer	Expert
P4	Software Engineer	Expert
P5	Journalist	Intermediate
P6	Higher Education	Proficient
P7	Software Engineer	Expert
P8	Philosophy Graduate	Intermediate
P9	Software Engineer	Expert
P10	Software Engineer	Expert
P11	Software Engineer	Proficient
P12	Software Engineer	Expert
P13	Software Engineer	Expert
P14	CS Professor	Expert
P15	CS PhD Student	Proficient
P16	ML Engineer	Expert
P17	Software Engineer	Expert

*Table 1 – This table shows each participant's profession as well as their relative expertise using LLMs. *Note: Here 'Expert' is used solely to distinguish engineers who have experience using agents to execute tasks autonomously, some of whom utilise multi-agent architectures. 'Proficient' indicates domain expertise but limited experience with agentic workflows, i.e., these users may have experience with contextually grounded tools like Claude Cowork, but they primarily interface with LLMs via a chat interface. 'Intermediate' refers to users who primarily interface with LLMs via a chat interface.*

iteratively based on the research questions summarised above. A private, local LLM (deepseek-r1 via Ollama) was used as a reflexive prompt to surface alternative readings: the LLM was prompted, for each transcript, to identify points of deviation from the themes that could point to additional ones, or observations that could add nuance to or alter themes. For each excerpt surfaced, the researcher assessed whether it represented a genuine deviation from a theme or was consistent with it and revised or retained accordingly.

3. Findings

The strategies that software engineers have developed, individually and as a community, for working with LLMs and LLM-powered agents have the potential to protect their agency over the aspects of their work they find meaningful or rewarding (such as the strategic or creative) while effectively offloading rote or repetitive work. This same offloading, combined with the speed at which LLMs produce code, also opens up the potential to glean even more reward from 'giving life' to new ideas at much greater speed. However, despite that positive potential, the interviews revealed how LLMs also heavily shift the nature of software engineering work from execution towards management or orchestration (or from production to evaluation), and this shift introduces new cognitive burdens that developers are still learning how to manage. Thus, the augmentative capacities of LLMs are counterbalanced by their potential to produce burnout. These findings are broken down in greater detail across three sections:

- §3.1: Software Engineers' Practices for Working with LLMs,
- §3.2: The Perceived Potential for Augmentation,
- §3.3: Tradeoffs between Augmentation and Burnout

3.1. Software Engineers' Practices for Working with LLMs

The most experienced software engineers in this study had developed ways of working with LLM-based agents that strived to establish complementary relationships in which the human's and the agent's roles were roughly divided between strategic decision-making and execution (e.g., code generation and debugging), respectively.

These practices were not derived from any explicit framework; varied theories still abound online regarding what exactly constitutes best practice for working with agentic coding tools. Participants instead described arriving at them through a combination of trial and error, learning through their own practice, and from tweets, blog posts, and videos shared by peers. Importantly, figuring out which tactics work for a specific context required sustained, deliberate effort that LLM-based systems don't necessarily support by default. At a high level, these practices are encapsulated in a few distinct but overlapping behaviours.

The first involves investing significant effort in specification and planning before any code is generated. This isn't a new practice for engineers, but its purpose has changed considerably in this new context. It is now as much about preserving agency as it is about defining the vision for a product. It is also a direct response to the default behaviour of LLMs, which is highly execution-driven, often at the expense of clarity and comprehensiveness (Laban et al., 2025). As P14 observed, 'These models all want to go to code as soon as possible. Sooner or later it asks, "Should we code this?" Eventually I'll say, "OK, I think we have enough."' The key word there is 'eventually.' By the time they hand anything to the agent, 'I've already gone through the major questions in the design phase, so I know what's going to happen.' Similarly, P9 described a process of extended collaborative planning with the LLM before allowing it to write a line of code: 'we are going to keep talking till we get to the level of specificity where it's appropriate for the handoff.' This front-loading keeps strategic and architectural decisions in human hands, while delegating the translation of those decisions into code to the model.

The second practice is the use of persistent agent configuration files (often referred to as AGENTS.md, CLAUDE.md, or SKILL.md files) that encode behavioural instructions, coding standards, and context that would otherwise need to be re-specified across sessions. P7 described their value: they can 'give very sloppy instructions' in the heat of a creative moment, knowing that the clarity encoded in the configuration file will compensate. Their key insight was that writing good agent instructions requires a different cognitive state than actively building something – "those are usually totally different sessions for me" – and that capturing instructions during moments of reflective clarity is what makes the system usable under less ideal conditions. This practice, similar to upfront specification, also appears to be a response to another limitation of LLMs: statelessness, or lack of actual memory (Dong et al., 2025; N. F. Liu et al., 2024). LLMs' statelessness, coupled with their frequent handling of large contexts, often forces the user to act as a memory store on their behalf, introducing new cognitive burdens (discussed more below).

The third practice is deliberate human-in-the-loop checkpointing, or structuring agent tasks so that humans review and approve frequently before an agent executes. This is enforced at varying levels and depends on the detail of a project's upfront specification work; however, some level of checkpointing is critical as a response to another limitation of LLMs, which is their tendency to 'drift.' Drifting really occurs because of their aforementioned statelessness; in other words, because they don't have memory, LLMs constantly need to summarise previous decisions and turns of conversation, and this compression (particularly the longer a session lasts) and resulting loss of detail can cause the LLM to 'drift' from its initial focused context (Laban et al., 2025; N. F. Liu et al., 2024). P10 describes responding to this behaviour: 'I usually start with "Let's make a plan," and make it actually write down a plan file and execute it step by step and confirm the steps with me as it goes along. That way it usually doesn't really drift.'

A practitioner perspective on why this matters came from P12, a prototyping architect who works with engineering teams trying to adopt AI. They described the problem with ceding too much control: 'Sometimes because you feel like it knows so well what it's doing, you end up giving it your full trust and letting

it take over... Then once it gets stuck, you're in a mess because you don't know all the things it created under the hood.' P12's preferred response of forcing the agent to work in ways they understand, even when it proposes something arguably better, reflects a broader shift they observed in how developers now make sense of their own codebases. Where the work was once to plan and then implement, it is now increasingly to specify, then verify what was built. Several other participants also described deliberately eliciting Socratic interaction (prompting the model to ask questions rather than generate answers) as a way of counteracting what P8 called the 'answer bias': LLMs' default tendency to produce confident responses rather than surface the assumptions that should be examined first. What unites these practices is that they keep the human in the strategic, creative, and architectural parts of the work while delegating execution to the model.

3.2. The Perceived Potential for Augmentation

Across participants and domains, negative perspectives on LLMs were most consistently expressed in response to the technology taking over work the user found rewarding, e.g., problem solving and strategic thinking for developers, ideation and craft for writers, critical thinking for a philosophy student. On the other hand, positive perspectives on LLMs were voiced by participants who had found ways to offload work they found tedious, repetitive, time-consuming, or cognitively burdensome (in an unrewarding way), while retaining agency and control over the parts of their work they found meaningful.

In sum, AI is experienced as augmentative when it handles jobs the user would rather not do, while leaving the user in control of the work they actually want to do. This adds nuance to traditional definitions of complementarity, i.e., a complementary relationship with AI is not one in which AI is simply delegated tasks it is 'better at.' Several participants noted that they were perfectly capable regarding the work they delegated, and in some cases could do it better than the LLM. P10, for instance, described delegating the database layer of a project: 'I know exactly how to do that; it just takes a week and it's annoying.'

When complementarity is achieved on these terms, participants described a qualitative shift in their experience of work. Participants with the most positive relationships with AI were not necessarily those producing the most output, though the ability to automate large amounts of rote work with LLMs was a factor. Where they perceived AI to be augmentative was in the renewed relationship with work it enabled. In several cases this was described as a reinvigoration of enjoyment that had been eroded by years of repetitive execution work. P3 described how encountering the same problem for the tenth time had gradually drained the joy from coding, and how LLMs had introduced the possibility of converting 'every wild idea' into a tangible proof of concept. P10, who described having 'a cemetery of dead side projects' before agentic coding tools, reported 'a renewed interest and enthusiasm' driven by being able to let the LLM handle the parts they found draining while retaining the problem-solving they find genuinely rewarding: 'my love was never the ability to write code. I think my love was for the ability to solve a problem.'

The same phenomenon explains why LLMs fail to feel augmenting in other contexts. P1, who writes and codes, was clear that they would not use AI for writing: 'I have the privilege of only doing things that I enjoy. Why would I pawn the things I enjoy off on AI?' P5 described the process of making connections between a quote and a piece of fact during journalism as something AI isn't capable of replicating: 'the logic and the connections that you make between either a quote that somebody's made or a piece of a fact that you've made – that I just think AI isn't capable of doing.' They argued that the spontaneous, chance-driven nature of ideation is fundamentally incompatible with the structured, prompt-based interface of a chatbot: 'I think a spontaneous process where you are trying to type something into the chatbot misses the point around the spontaneity of the idea process.'

P8, who experimented extensively with AI-assisted essay writing, found that it substituted a different and less satisfying kind of cognitive work for the kind they valued: 'I was spending a lot of time thinking about how to work with AI to produce a very effective essay... it always felt more scattered... everything was a lot shallower. That's not to say the output was shallower, but I wasn't responsible for anything that wasn't shallow.' This concern extended to the value of effort itself. P8 observed that intrinsic motivation

depends on recognising that work produces a commensurate reward, and that AI risks severing that connection: 'I just don't think it is realistic to expect that people can continually get the same amount of enjoyment out of doing something when they don't have to work hard... You develop intrinsic motivation by recognising that X amount of work will give X reward.' A similar intuition was expressed by P4, an early adopter of agentic tools, who, despite overall enthusiasm, noted that when everything is automated, it raises the question of where intrinsic value comes from at all: 'If everything takes no effort, does it have any value?'

P13, a games developer who had used AI extensively for web development, highlighted what happens when AI takes over too much of the rewarding work. When they extended AI use to the parts of development they genuinely loved, the work became 'very monotone... very dry... instead of being a sort of passion... it started to become more of a chore.' They described this not as a productivity problem but as a loss of craft: 'writing the code isn't the sole goal of development. There is a craftsmanship part to it that is very undervalued if you offload all of that work.'

The intrinsic reward framing also helps explain a related phenomenon: the tendency to overestimate productivity gains from LLM use. P3 described what they called a 'magic factor.' This describes the situation in which the moments of surprise at what LLMs are capable of are so significant that they overshadow the frustrating hours spent in unsuccessful 'doom loops,' which are both less memorable and often lack negative professional consequences since experimentation is encouraged. The result is that people exit interactions with LLMs perceiving them to have been more helpful than objective metrics might support. Perceptions of augmentation and actual augmentation are therefore not the same thing, and intrinsic reward is partly what conflates them. When AI handles work the user finds unrewarding and leaves them with work they find rewarding, the experience genuinely does feel like augmentation, regardless of whether output quality or quantity has measurably improved.

3.3. Tradeoffs between Augmentation and Burnout

The dark side of this perception-vs-reality phenomenon is that some software engineers described coding with agents as having become addictive, suggesting that these interactions are tapping into extrinsic reward mechanisms as well.

P7 captured this shift without being prompted: 'the work is shifting from writing the code to specifying the system. Not everybody enjoys that part of the work. I think it's actually much more mentally draining... I am thinking harder than I have in a long time because the pace and volume of everything is just so fast.' P10, despite their enthusiasm for what agentic coding has made possible, also noted: 'a lot of what I'm doing now, it genuinely feels like I'm not doing software engineering anymore. It sometimes feels like I'm babysitting agents.' P9 described the cognitive demands in terms of what managing multiple agents actually requires, likening the engineer's new role to that of a startup founder: 'The founder's capability at describing a future vision is really important because they're not going to be able to review all these decisions that need to happen with them out of the loop [i.e., the decisions made by agents].' These interviews demonstrate that the new work is not only cognitively demanding but, for many participants, less intrinsically rewarding than the production work it partially replaces, which could at least partially explain why it is so 'mentally draining.'

A related concern, expressed more tentatively across several interviews, is whether extensive LLM use is affecting skill development and output quality over time. P8 was most direct in expressing this concern: 'AI does a lot of things you would otherwise spend time doing, which means you're not spending time thinking about the material... my understanding went down. There's no doubt about it, because I was just moving through far more information more quickly.' P3 described observing a near-universal pattern among coworkers a few months into working intensively with LLMs: 'almost universally everyone starts to lower their quality bar a bit... because of the sheer amount of outputs you have to consume and validate. That's dangerous.' P13 offered the most detailed account of skill atrophy from personal experience. Having used AI heavily for web development, they found that after several months, their intuitive grasp of the codebase had faded. Tasks that were once automatic became deliberate, and de-

bugging, which had previously felt instinctive, required conscious effort where it hadn't before. They described losing what they called their *Fingerspitzengefühl*, or the feeling in one's fingertips for where problems were located (in other words the intuition that comes with expertise) (Dreyfus, 2004). They attributed this directly to having offloaded the thinking: 'you initially would probably think that... I still do the thinking, it's just like the writing, the basic code. But I think you need to sort of keep your skills sharp, just like how athletes still do the basic drills.' These observations were not the focus of this study and should be treated as preliminary, but they point to questions that longitudinal research will need to address.

Several participants also raised questions for what implications the challenges they faced could have for those still developing expertise. The strategies that allow experienced developers to approximate complementarity have to be learned in tandem with all that the inexperienced don't know about programming. As P10 observed, 'the stuff you don't know you don't know. That's the gigantic problem... a junior won't necessarily know that what [the LLM] is doing is correct... you just don't have the experience.' This dynamic was echoed by P6, a design educator who observed the same pattern in students. The result is that LLMs, in their current default design, risk replacing cognitive processes that novices need to develop rather than supporting them, which means amplifying existing gaps rather than closing them. So while LLMs have the potential to maximise rewarding work, they also have the potential to amplify existing inequalities in expertise.

4. Discussion

This paper has described how some of the strategies that professional software engineers (SWEs) have developed to work more effectively with LLM-based agents can both have augmentative, rewarding effects while at the same time producing burnout. Throughout, emphasis has been placed on the potential for LLMs to afford SWEs more time for the intrinsically rewarding, or meaningful, aspects of their work while offloading the rote or tedious. It is important to note that this was an emergent theme, not one driven by the study's research questions; however, it serves as a productive framing when considering how to design AI-powered tools that are more complementary to their users.

Such a framing echoes discourses on human flourishing, which encourage a more holistic perspective on individual and collective well-being (VanderWeele, 2017). However, there is reason to think this reorientation in perspective requires deliberate effort. Brynjolfsson (2022) argues that technology developers and businesses face structural incentives that favour automation over augmentation. Substituting existing tasks is the path of least resistance, while genuine augmentation demands more radical organisational transformation. The result is what he called the 'Turing trap,' a systematic bias toward developing AI that emulates human performance at existing tasks instead of expanding what humans are capable of. In this view, complementarity is not treated as the default.

From a 'human flourishing' perspective, complementarity looks beyond enhancing human skills with the goal of improving productive output, and towards considerations of the consequences that such a myopic focus (i.e., on productivity) might have on greater well-being. Here that means designing to protect human agency by default instead of allowing it to be something that users have to work for. As this interview sample exhibited, even those users who are best positioned to architect their workflows to protect their agency (because they have the technical skills to steer LLM behaviour beyond simple prompting) still struggle to succeed in getting agents to work for them, risk confronting burnout, and occasionally still inadvertently get locked out of decision-making. This has been observed more broadly in corporate AI adoption, where the initial productivity gains from AI use tend to give way to workload creep, cognitive fatigue, and burnout as workers self-impose higher pace and expanded scope without being asked to (Gent, 2025; Ranganathan & Ye, 2026). The fatigue and burnout are likely, at least in part, a result of the human's role in software development work shifting from a production and execution focus to one that is more evaluative and managerial (Simkute et al., 2024). Another recent study suggests that this encroachment on agency could extend far beyond software engineering. In a large-scale survey with over 300 American workers across a diverse range of professions, Ranjit et al. (2026) found that

meaningful work – tasks which workers associate with a greater sense of agency or happiness – may be disproportionately exposed to intrusion by AI, as compared to 'low-level' work.

This is not to say that high-level domains should be exclusively reserved for humans and low-level tasks delegated disproportionately to artificial agents, but rather that there is still a ways to go towards designing truly complementary AI systems: ones that are capable of maximising the human user's agency over what they find meaningful without introducing unnecessary cognitive burdens. Naturally, 'meaningful' differs from person to person, and current AI systems are not designed to uncover what that means for each user nor to structure interactions accordingly (Fischli et al., 2026). Current systems operate closer to what Fischli et al. (2026) call the 'liberal' approach to promoting autonomy in human-AI interactions, i.e., acting on what users explicitly ask for, without the 'capability-boosting' dimension that would require understanding and actively protecting each user's needs over time.

Where the flourishing framework and Brynjolfsson's case for augmentative AI highlight the importance of protecting agency at the systems level, self-determination theory (SDT) provides possible explanations for its consequence at the individual level. SDT is an empirical framework that studies how differing types of motivation predict outcomes such as psychological wellbeing, learning, and performance on creative problem-solving tasks (Deci & Ryan, 1985). Central to SDT is the distinction between autonomous motivation, in which people experience volition and self-endorsement of their actions, and controlled motivation, in which behaviour is driven by external contingencies or pressure. Importantly, autonomous motivation leads to better psychological health outcomes, greater persistence, and more effective performance on heuristic tasks, which are common in knowledge work (Deci & Ryan, 2008).

Work that is autonomously motivated is also more energising, while controlled motives drain energy, which could partially explain why multiple participants reported that AI's potential to free up their time to work on personal projects felt so invigorating (Deci & Ryan, 2008). Even though software engineering in a work context can be considered extrinsically motivated, participants' experiences indicate that it is very much possible to be motivated by, and to find enjoyment in, specific aspects of that work. This is perhaps why some react so strongly to AI 'treading on their toes,' so to speak, when it comes to those aspects referred to here as meaningful or rewarding. For many, strategic problem solving and decision-making encapsulate what they find rewarding about engineering. For some (arguably a minority), like P13, this extends to the act of writing code. However, even for P13, the importance of writing code was less about the actual act of writing code and more about the upkeep of the core skillset and intuition (the 'craftmanship') that made him a good engineer. As they said, 'It's just more fun to do things yourself.' Whether or not people like P13 are right in believing that generative AI spells doom for software engineering as a craft, the design of future AI systems could better tilt the balance towards complementing, and away from replacing, humans by becoming more adaptable to individual users' personal fulfillment. The industry-standard practice of keeping repository-level context files (e.g., 'CLAUDE.md') is evidence of, and a step towards, this. However, it is also indicative of something broken, particularly because they are not airtight solutions for addressing individual user needs due to LLMs' limitations in managing context (N. F. Liu et al., 2024), nor do they reliably improve code quality (Gloaguen et al., 2026).

The findings for expert users raise the question of how less experienced users will be impacted. Will they have the same experiences as experienced software engineers or could their challenges be compounded? Recent work suggests that offloading tasks to AI denies users the experience of productive struggle through which accurate knowledge of one's own capabilities develops, which results in reduced persistence even in the short term (Elizondo et al., 2024; Fleming & Daw, 2017; G. Liu et al., 2026; Yeung & Summerfield, 2012). Expert software engineers have already developed this self-knowledge, but novices haven't, which could explain why the benefits of LLM-assisted coding are not shared equally across skill levels (Prather et al., 2024). Complicating this picture further are the new cognitive burdens discussed above. Participants' accounts of the demanding nature of learning to work with LLMs are consistent with Simkute et al. (2024)'s analysis of the 'ironies of generative AI,' which documents

how the shift from production to evaluation introduced by automation does not simply reduce workload but redistributes it in ways that are often more taxing than anticipated. This is an effect of automation documented at least as far back as 1983 (Bainbridge, 1983). Sarkar and Drosos (2025) corroborate this phenomenon independently, describing it as programming expertise being 'redistributed rather than eliminated' in agentic workflows. Experienced developers may be better positioned to manage these new burdens, having already developed both the foundational technical skill to evaluate LLM outputs and the metacognitive strategies that effective delegation requires. Non-experts have neither.

Presented with system designs that require expert skill to protect agency and reward, novices risk offloading too much, and, in the process, lose the opportunity to learn not only important technical skill but also what they find rewarding about coding, which in turn impacts their ability to develop successful delegation strategies. Zhang (2024) goes even further, arguing that certain human activities derive their value intrinsically, from the quality of one's direct engagement, and cannot be meaningfully delegated to computational systems without loss. This again highlights the importance of exploration and the development of underlying skill, as some activity may not be able to be offloaded without impacting, at the very least, persistence and motivation. Two design directions for AI systems surface from these considerations: one that focuses on fostering skill development, and another that focuses on adaptability to what the individual user finds rewarding, ideally both part of the same system. These correspond to two of the autonomy-promoting strategies that Fischli et al. (2026) propose for personal AI agents: capability-boosting, which promotes autonomy by helping users build the skills they need to pursue their goals independently, and meta-autonomy, which gives users control over what kind of AI interaction they want across different domains, e.g., following direct instructions in one context, scaffolding skill development in another.

5. Conclusion

These findings suggest that the question of whether LLMs augment human capabilities is not a property of the technology itself, but of the relationship a user is able to construct with it. That relationship is shaped in part by the level of agency a user is able to exercise over the parts of their work they find rewarding or meaningful. Expert software engineers demonstrate that this is achievable to some extent, but only through sustained effort that current tool design neither scaffolds nor encourages. For less experienced users, the same tools that offer the promise of augmentation risk replacing the cognitive processes through which both technical skill and a sense of meaningful work develop. This reframes what it means to design for augmentation. Rather than optimising for output quality or task completion, systems that genuinely complement their users must account for what those users find rewarding, and actively work to preserve it. Two design directions follow from this work: systems that scaffold the skill development required for meaningful delegation, and systems that adapt over time to individual users' sense of what constitutes rewarding work.

References

Alami, A., & Ernst, N. A. (2025, January). Human and Machine: How Software Engineers Perceive and Engage with AI-Assisted Code Reviews Compared to Their Peers. https://doi.org/10.48550/arXiv.2501.02092

Bainbridge, L. (1983). Ironies of automation. In *Analysis, design and evaluation of man–machine systems* (pp. 129–135). Elsevier.

Braun, V., & Clarke, V. (2019). Reflecting on reflexive thematic analysis. *Qualitative Research in Sport, Exercise and Health*, *11*(4), 589–597. https://doi.org/10.1080/2159676X.2019.1628806

Brynjolfsson, E. (2022). The Turing Trap: The Promise & Peril of Human-Like Artificial Intelligence. *Daedalus*, *151*(2), 272–287. https://doi.org/10.1162/daed_a_01915

Deci, E. L., & Ryan, R. M. (1985). *Intrinsic Motivation and Self-Determination in Human Behavior*. Springer US. https://doi.org/10.1007/978-1-4899-2271-7

Deci, E. L., & Ryan, R. M. (2008). Self-determination theory: A macrotheory of human motivation, development, and health. *Canadian Psychology / Psychologie canadienne*, *49*(3), 182–185. https://doi.org/10.1037/a0012801

Dell'Acqua, F., McFowland, E., Mollick, E. R., Lifshitz-Assaf, H., Kellogg, K., Rajendran, S., Krayer, L., Candelon, F., & Lakhani, K. R. (2023). Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.4573321

Dong, C. V., Lu, Q., Norman, K. A., & Michelmann, S. (2025). Towards large language models with human-like episodic memory. *Trends in Cognitive Sciences*, *29*(10), 928–941. https://doi.org/10.1016/j.tics.2025.06.016

Dreyfus, S. E. (2004). The Five-Stage Model of Adult Skill Acquisition. *Bulletin of Science, Technology & Society*, *24*(3), 177–181. https://doi.org/10.1177/0270467604264992

Elizondo, K., Valenzuela, R., Pestana, J. V., & Codina, N. (2024). Self-regulation and procrastination in college students: A tale of motivation, strategy, and perseverance. *Psychology in the Schools*, *61*(3), 887–902. https://doi.org/10.1002/pits.23088

Fischli, R., Franklin, M., Manzini, A., & Gabriel, I. (2026). Agents, Alignment, and the Many Faces of Autonomy. *Minds and Machines*, *36*(3), 34. https://doi.org/10.1007/s11023-026-09786-9

Fleming, S. M., & Daw, N. D. (2017). Self-Evaluation of Decision-Making: A General Bayesian Framework for Metacognitive Computation. *Psychological Review*, *124*(1), 91–114. https://doi.org/10.1037/rev0000045

Gent, E. (2025). AI coding is now everywhere. But not everyone is convinced. *MIT Technology Review*.

Gloaguen, T., Mündler, N., Müller, M., Raychev, V., & Vechev, M. (2026, February). Evaluating AGENTS.md: Are Repository-Level Context Files Helpful for Coding Agents? [arXiv:2602.11988 [cs]]. https://doi.org/10.48550/arXiv.2602.11988

Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M. B., Rothermel, G., Shaw, M., & Wiedenbeck, S. (2011). The state of the art in end-user software engineering. *ACM Computing Surveys*, *43*(3), 1–44. https://doi.org/10.1145/1922649.1922658

Kraljic, T., & Lahav, M. (2024). From Prompt Engineering to Collaborating: A Human-Centered Approach to AI Interfaces. *Interactions*, *31*(3), 30–35. https://doi.org/10.1145/3652622

Laban, P., Hayashi, H., Zhou, Y., & Neville, J. (2025, May). LLMs Get Lost In Multi-Turn Conversation [arXiv:2505.06120 [cs]]. https://doi.org/10.48550/arXiv.2505.06120

Liu, G., Christian, B., Dumbalska, T., Bakker, M. A., & Dubey, R. (2026, April). AI Assistance Reduces Persistence and Hurts Independent Performance [arXiv:2604.04721 [cs]]. https://doi.org/10.48550/arXiv.2604.04721

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2024). Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*, *12*, 157–173. https://doi.org/10.1162/tacl_a_00638

Meredith Ringel Morris. (2024, October). Prompting Considered Harmful – Communications of the ACM. Retrieved February 9, 2026, from https://cacm.acm.org/opinion/prompting-considered-harmful/

Mohamed, A., Assi, M., & Guizani, M. (2026). The Impact of LLM-Assistants on Software Developer Productivity: A Systematic Review and Mapping Study [Just Accepted]. *ACM Trans. Softw. Eng. Methodol.* https://doi.org/10.1145/3809494

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

Ranganathan, A., & Ye, X. M. (2026). AI doesn't reduce work—it intensifies it. *Harvard Business Review*. Retrieved March 12, 2026, from https://hbr.org/2026/02/ai-doesnt-reduce-work-it-intensifies-it

Ranjit, J., Zhou, K., Swayamdipta, S., & Quercia, D. (2026, March). Are We Automating the Joy Out of Work? Designing AI to Augment Work, Not Meaning.

Sarkar, A., & Drosos, I. (2025, October). Vibe coding: Programming through conversation with artificial intelligence [arXiv:2506.23253 [cs]]. https://doi.org/10.48550/arXiv.2506.23253

Simkute, A., Tankelevitch, L., Kewenig, V., Scott, A. E., Sellen, A., & Rintel, S. (2024). Ironies of Generative AI: Understanding and Mitigating Productivity Loss in Human-AI Interaction. *International Journal of Human-Computer Interaction*. https://doi.org/10.1080/10447318.2024.2405782

Steyvers, M., Tejeda, H., Kerrigan, G., & Smyth, P. (2022). Bayesian modeling of human–ai complementarity. *Proceedings of the National Academy of Sciences*, *119*(11), e2111547119.

Suh, S., Min, B., Palani, S., & Xia, H. (2023). Sensecape: Enabling Multilevel Exploration and Sensemaking with Large Language Models [arXiv: 2305.11483]. *UIST 2023 - Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. https://doi.org/10.1145/3586183.3606756

Tang, N., Chen, M., Ning, Z., Bansal, A., Huang, Y., McMillan, C., & Li, T. J.-J. (2024, May). A Study on Developer Behaviors for Validating and Repairing LLM-Generated Code Using Eye Tracking and IDE Actions. https://doi.org/10.48550/arXiv.2405.16081

VanderWeele, T. J. (2017). On the promotion of human flourishing. *Proceedings of the National Academy of Sciences*, *114*(31), 8148–8156. https://doi.org/10.1073/pnas.1702996114

Wilder, B., Horvitz, E., & Kamar, E. (2020, May). Learning to Complement Humans [arXiv:2005.00582 [cs]]. https://doi.org/10.48550/arXiv.2005.00582

Yeung, N., & Summerfield, C. (2012). Metacognition in human decision-making: Confidence and error monitoring. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *367*(1594), 1310–1321. https://doi.org/10.1098/rstb.2011.0416

Zhang, H. (2024). Searching for the Non-Consequential: Dialectical Activities in HCI and the Limits of Computers. *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 1–13. https://doi.org/10.1145/3613904.3641945

Supervision Theatre: Human-on-the-Loop as the Spear Carrier in the Agentic Play

Joana Vieira
OutSystems
joana.vieira@outsystems.com

Carlos Silva
OutSystems
carlos.silva@outsystems.com

Menno Cramer
OutSystems
menno.cramer@outsystems.com

Abstract

The vocabulary of Human-Computer Interaction was established in an era when humans executed and computers assisted. Norman's gulfs, GOMS, Cognitive Work Analysis - these frameworks share an assumption: *the human perceives and acts, the system responds, and good design closes the gap between intention and outcome*. Forty years on, AI and Agentic systems have started challenging that relationship. The human is no longer the sole executor but the supervisor, or the approver, or, more often than designers care to admit, the bystander. Yet we continue to reach for the same conceptual toolkit, dressing supervision-era problems in execution-era language. This flaw in categorization goes beyond the conceptual: it produces systems that cannot name what their human operators are actually doing, and therefore cannot design for it.

The transformation we're witnessing in software development tools is not an incremental evolution but a category shift in what it means to be the human on the loop. We argue that the most common human-agent configuration now being deployed, the developer passively monitoring a capable autonomous system, labelled Human-on-the-Loop (HOTL), is not a guarantee of safe performance. In fact, it is the most cognitively dishonest position of all. It promises supervision while structurally rendering it quite difficult. We call this pattern the *supervision theatre*. In this performance, the human acts more like the spear carrier - the non-speaking background actor whose presence implies a scene they cannot affect.

Our argument rests on Rasmussen's Decision-Ladder and his distinction between skill, rule, and knowledge-based behaviour. HOTL assumes the human can ascend on demand (perceiving, interpreting, evaluating, reformulating intention), in time to intervene meaningfully. But when the system is fast, opaque, and confidence-inducing, the human is held at the skill-based rung, reacting to salient surface signals rather than reasoning about them. Agentic coding tools optimised for frictionless flow create exactly these conditions, and then call it oversight.

Kahneman's account of System 1 and System 2 thinking compounds the problem. System 2 - the slow, deliberate, effortful reasoning that meaningful intervention requires - does not simply activate because the situation demands it. It activates when friction signals that something is wrong. Agentic interfaces optimised for fluency are, in this precise sense, System 2 suppressants: the confident output stream, the seamless commit-and-deploy removes the very friction that would trigger deliberate evaluation. The human watches the agent act, nothing feels wrong, and System 2 never engages.

Aviation and automotive automation are instructive here. Those fields demonstrate what genuine supervisory infrastructure requires: moving yokes, mode awareness displays, and recurrent simulation training. What made them a benchmark for human-automation partnership was not a commitment to HOTL throughout. It was the investment in making supervisory positions cognitively honest, and the discipline to insert Human-in-the-Loop (HITL) gates at moments of genuine consequence, such as the pre-flight checklist or the mandatory handover protocol.

Effective Human-Agent Interaction (HAI) requires knowing where the human actually is in the loop at any given moment, and designing for that position, not the one we assume or the one for which is easier to design. We propose four dimensions (accuracy required, latency tolerated, risk exposure, and cognitive feasibility) for locating the human in the HAI space, and derive corresponding interface

design commitments. Patching a vulnerability identified in a production codebase is not the same kind of task as drafting a commit message, and current interfaces should surface the appropriate differences. The framework is not a taxonomy for its own sake: it is a diagnostic instrument. Each dimension answers one of the questions that current tools and design discourse cannot answer, and we show that inability using the concrete failure patterns already endemic to today's agentic coding tools. Naming the position is the precondition for designing for it.

Software designers are choosing right now which patterns become the default for an industry still forming new Human-Agent interaction habits. These choices should be grounded in the cognitive-engineering tradition that tackled these problems in industries that faced somewhat similar transformations. We argue for the conditions it must meet: design that names the loop position the human inhabits, that treats every transition as a critical surface for design, and that measures transparency by what the human can do with what is shown, not by how much is shown.

1. Introduction

Human Computer Interaction emerged to close a gap. Norman's gulfs of Evaluation and Execution (1986), GOMS (Moran, Newell & Card, 1983), Cognitive Work Analysis (Rasmussen, 1986) are frameworks for a world in which the human acts, the system responds, and good design minimises the distance between human intention and system outcome. HCI did not stop at the reactive computer. Intelligent user interfaces and mixed-initiative interaction (Horvitz, 1999) have modelled systems that take initiative. But even there, the human held execution authority turn by turn: the system proposed, the human disposed, and the gulf being closed was still the one between a human's intention and the system's response to it. Agentic AI does not inhabit this world. When a system autonomously plans, delegates, and executes multi-step tasks, the human's role shifts from actor to supervisor. The gulf that matters now is between what the agent interprets of the human intent, what the agent executes, and what the human understands the agent to be executing. This is not an incremental challenge, but a shift on the interaction paradigm.

A vocabulary has grown up for positioning humans relative to automated systems. Human-In-The-Loop (HITL), Human-On-The-Loop (HOTL), Human-Behind-The-Loop (HBTL) (Rzeszucinski, 2024) - acronyms that treat supervisory positions as a spectrum with safe options along it. HOTL in particular has acquired the status of a reasonable middle ground: autonomous enough for efficiency, supervised enough for accountability. The argument that this framing is incomplete and actively dangerous is not new (Parasuraman, Sheridan & Wickens, 2000), but it gains new traction in the context of Human-Agent Interaction.

What we are witnessing with AI agentic systems is the appearance of meaningful human oversight without its cognitive substance, a condition we call *supervision theatre*. By supervision theatre we mean an arrangement that stages all the visible signals of oversight: a human present, plans displayed, approvals requested; while denying that human the cognitive conditions needed to actually evaluate what the agent is doing, so the oversight is *performed* rather than *exercised*. Decades of work on supervisory control (Endsley & Kiris, 1995; Parasuraman, Sheridan & Wickens, 2000) have documented when human supervision works and when it fails. This paper argues that the problem is not HOTL as a concept, but HOTL as it is currently practiced in the specific context of Agentic Coding tools. When properly designed with intervention affordances, situation awareness, and task decomposition, supervisory positioning can work - including in agentic contexts where HOTL and HITL operate on the same task. However, these conditions must be designed in, they do not emerge from the architecture of oversight alone.

The current pace of agentic deployment creates pressure to establish accountability structures before the underlying cognitive design has been done. The problem of designing for these tools is, at its root, a vocabulary failure. Our existing frameworks were built to name what happens when humans execute. They give us precise language for the gulf of evaluation, for the breakdown between intention and feedback, for the mapping between mental models and system state. What they cannot

name is the difference between a human who is genuinely supervising and a human who is merely present. Without that vocabulary, we cannot identify when supervision theatre is occurring, cannot specify what cognitive work is actually required at a given loop position, and cannot test whether an interface design delivers that work or merely performs it. The framework proposed in this paper is, in a first instance, a vocabulary for naming what is happening. The design commitments follow from that.

2. The Supervision Theatre

In a paper from 1978, Sheridan and Verplank outlined a spectrum of ten distinct modes of human-automata collaboration, running from "the human does the whole job up to the point of turning it over to the computer" to "the computer does the whole job and tells the human only if it decides it should". The term human-in-the-loop entered the HCI vocabulary through the cockpits of teleoperated vehicles and aircraft before making its way to control rooms (Sheridan, 1981), and ultimately the cubicle that holds a single human and a computer (Parasuraman, Sheridan & Wickens, 2000). A core anxiety runs through all of this initial work: when a human is nominally overseeing a system but not actively paying attention to it, or genuinely understanding it, something dangerous forms in the gap between presence and attention. Wickens (1995) named this the "*out-of-the-loop unfamiliarity*", a state produced by diminished situational awareness, complacency, and the erosion of skills.

Endsley's three-level model of situation awareness (1995) describes that passive monitoring degrades situational awareness through three compounding effects: operators become less vigilant, process information less deeply because they are not acting on it, and receive insufficient feedback to reconstruct what the automation has actually been doing. The result is an operator who can intervene in principle but cannot intervene effectively in practice.

By the 1990s, human-on-the-loop (HOTL) had emerged as the dominant architectural choice in advanced autonomous systems, conceived to reserve human attention for exception handling, with the system acting autonomously in real-time and the default loop closing without the human. Aviation adopted HOTL not naively but with a supporting infrastructure: strict certification, crew resource management training, regulated rest periods, and accident investigation bodies capable of feeding failure modes back into design. Crucially, the human supervisory role was designed around transition moments, from HOTL to human-in-the-loop (HITL) to manual control - not treated as a fixed position.

Rasmussen's work gives us two complementary tools. The decision ladder models how an operator moves from an observed signal to an executed action: detecting an alert, interpreting its meaning, evaluating it against goals, defining a task, and only then acting. Its central insight for our purposes is that a skilled operator does not climb every rung: familiar situations are handled by lateral "shortcuts" that leap from a cue straight to a stored response, bypassing deliberate evaluation entirely. Rasmussen's Skills-Rules-Knowledge taxonomy (1986) illuminates why those transitions matter. In principle, HITL operates at the Knowledge level: the human reasons about a specific case to form adequate goals and plans. HOTL operates at the Skill or Rule level: the supervisor has enough pattern recognition to catch anomalies without deliberate reasoning. Since skill- and rule-based shortcuts are fertile ground for use errors, any system that fails to account for these cognitive tendencies is prone to interaction failures (Silva et al., 2019). AI Agentic systems intensify this further: these systems frequently perform novel actions that exceed a human's ability to form clear goals at the Knowledge level, while their opacity and speed bypass the pattern-recognition on which Skill- and Rule-level monitoring depends.

The cognitive mechanism underlying failures that might happen in these moments has recently been named *cognitive surrender*. Shaw and Nave (2026) define it as the uncritical adoption of AI outputs with minimal scrutiny. The user stops deliberating and accepts the agent's answer as their own. In three experiments, participants followed faulty AI recommendations on roughly four out of five occasions, and their confidence increased regardless of whether the AI was correct or not. When outputs arrive confidently and without friction, the signal that would ordinarily route a response to

System 2 deliberation does not fire. Kahneman's account of dual-process cognition predicts exactly this: System 2 engages when something demands effort or signals error; remove those signals and it stays dormant (Kahneman, 2011).
This is the cognitive complement to supervision theatre. Where supervision theatre names the design pattern, an interface that claims oversight while preventing it; *cognitive surrender* names what happens inside the developer watching the stream of confident agent output. The Human-on-the-loop becomes the *spear carrier*. In theatre, the spear carrier is the non-speaking actor who stands in the background of a scene: their presence completes the stage picture and implies a world they take no part in shaping. The HOTL supervisor is increasingly cast in this role on stage, but unable to affect the action.

2.1. The Current State: where-in-the-loop in Agentic Coding Tools?

In the space of roughly three years, Agentic coding tools traversed the full HITL-to-HOTL spectrum, driven by productivity framing rather than any explicit theory of human supervisory control. The consequence is a set of tools where the human is technically In or On the Loop, but the conditions of the system (speed, volume, opacity, automation bias) render oversight effectively illusory. The hybrid approach of IDEs coupled with LLMs capable of generating code (such as GitHub Copilot, Cursor, Visual Studio) sits closer to the HITL end. Depending on settings, each code suggestion may require active human acceptance. Cognitive engagement is higher and more continuous than in fully agentic approaches. The failure mode here is mostly reflected in an automation bias in suggestion acceptance: the developer approves suggestions faster than they can genuinely evaluate them.
Agentic coding tools (such as Claude Code, OpenAI Codex, Google Jules, GitHub Copilot Workspace) represent the genuine HOTL shift. The developer specifies a goal: "migrate this codebase to Python 3," "add authentication with 2FA", "set up an Approval workflow", and the agent plans, executes, and iterates with minimal per-step human involvement. The human might still review logs, inspect diffs, and approve merges, but this is identical to how a pilot supervises an autopilot during cruise: authority is delegated, the human monitors, and the expectation is that intervention is reserved for exceptional events. Although software development is not a real-time task like flying an airplane (between a developer’s inattention and its consequences sits a test suite, a code review and a staging environment - a pilot has none of those) the problem we identify is not concerned with the timing of the consequences, but instead with its root cause. A human watching a fast, fluent, confidence inducing yet opaque process is fertile ground for disaster. Development-time errors are committed, replicated, and executed later, in production, often by someone else, and the absence of real-time pressure is not a mitigation strategy. The flying an airplane analogy is instructive precisely because aviation did not stop there, it built the cognitive infrastructure that made that supervisory position honest (Sebok et al., 2015). Agentic coding tools, so far, have not.
Compiling a catalog of performative design traits in today's agentic coding tools is the definition of aiming at a moving target. These tools change at an unprecedented speed, but nevertheless some patterns have remained confusing or performative:

- **The code snippets and reasoning that pass you by at lightning speed** - Cursor originally used it extensively, streaming code snippets to users at the agent's generation speed, though this has been moderated in newer releases. Claude.ai employs a similar approach for artifact generation, and nearly every other agentic AI tool uses it to provide a window into the agent's internal reasoning. While these movements signal activity and suggest transparency, the sheer volume and velocity of information often exceed human processing capabilities. There is a growing trend toward abandoning this practice. It remains unclear if this shift is driven by usability concerns or, as OpenAI co-founder Greg Brockman suggested (2026), a strategic decision to stop exposing raw chains of thought. According to Brockman, hiding these reasoning steps serves two purposes: (1) preventing *distillation*, as exposing internal reasoning allows competitors to more easily replicate the model's technology; and (2) preserving *faithfulness*, as the pressure to format thoughts for user

approval might lead the model to prioritize appearances over actual problem-solving accuracy.

- **The enigmatic Stop button** - Present in almost all agentic coding tools, the stop button does one thing for sure, it stops the agent work right in its track. The user is left with an eerie sense of not knowing what, or if something, was written. And if it was, was it reversed? Official documentation from the biggest players clearly explaining this behavior is surprisingly difficult to find. Fortunately you can ask the agents and according to Claude Opus 4.7, the stop button seems to stop the agent execution guaranteeing that it doesn't do it in the midst of running a tool but reverting any code changes will depend on when the stop happens - "*..."code being generated" is discarded; "code already handed to the Write/Edit tool" is likely already on disk by the time you can react*". That the most authoritative available account of the stop button's semantics comes from interrogating the agent itself, rather than from any documentation the vendor publishes, is, of course, part of the point. Cursor and Codex seem to apply a similar strategy and a similar resistance to put it out on official documentation.

- **The unscrutable permission requests** - The default configuration for IDEs coupled with AI assistants typically restrict consequential autonomous agent actions, requiring manual authorization for most write operations. But ultimately this constant stream of permission request prompts, coupled with a high success rate, tends to condition users to bypass scrutiny, a behaviour we have described as *cognitive surrender* (Shaw & Nave, 2026). Developers frequently resort to options like whitelisting in Cursor, selecting "always allow" or "bypass permissions" in Claude Code, or enabling permissions to apply to all projects in VS Code. This shift begins with transparent requests but eventually extends to more obscure ones that even technically proficient users may find difficult to decipher.

- **The unpredictable context filling** - Context filling is frequently represented by a circular progress indicator or token counter, but this representation systematically misleads. What fills the context window is opaque to the developer: is it the conversation history? Tool calls? File contents the agent retrieved? The counter tells you how full the context is without telling you what is in it. When the context approaches full capacity, agent behaviour degrades in ways that are difficult to predict and harder to diagnose. The agent might silently drop earlier instructions, lose track of architectural decisions made at task outset, or repeat work it has already done. The degradation is rarely announced. Some tools offer a "compact" or "summarise" action to reclaim context space, but the summary is itself produced by the model, substituting an original record for the lossy compression, while the developer does not know what was lost. This is a transparency problem at the centre of the supervisory relationship. A human supervisor who cannot know what their agent remembers cannot meaningfully evaluate what their agent is about to do.

- **Long plans, big small changes** - Agentic tools have developed a pattern of presenting elaborate, multi-step plans before execution with numbered lists of intended actions that signal thoroughness and invite implicit approval. The plan creates an expectation, but the execution can break it, not through spectacular failure but through an accumulation of small, locally reasonable deviations: a dependency quietly upgraded, an abstraction layer introduced that was not specified, a configuration file modified because it was in the way of the task's main goal. Each individual decision might be defensible but the aggregate is a codebase that has drifted from what the developer understood themselves to be building. This is precisely the *out-of-the-loop* degradation described before: the human approved a plan, not a process, and the agent executed a complex, hard to follow process. The design problem is that plan presentation has become its own form of supervision theatre. A ritual of the numbered lists substituting for the harder work of making moment-to-moment agent decisions genuinely legible.

This is not an exhaustive list and it is one focused on the core of the development tasks. One could easily add to it the difficulty to predict token usage (and pricing) and other struggles that are more related to the customer journey. These are patterns we kept running into as practitioners building and shipping software on a development platform. Also important to say is that these features might behave differently depending on the tool but, at the time of the writing of this article, the described problematic behaviors were endemic to most of the more popular tools.

2.2. The Moral Crumple Zone: Coming Soon to a Supervision Theatre Near You

The concept Elish (2019) introduced as the "*moral crumple zone*" describes what happens when the conditions of illusory supervision play at scale. The term names the scenario in which a human operator, positioned in the loop by design, absorbs blame for a failure they could not realistically have prevented given the oversight conditions they were actually working under. The crumple zone is a predictable consequence of designing accountability into a system without designing the cognitive infrastructure that would make accountability meaningful.

In its original context, Elish drew on cases from robotics and autonomous vehicles, namely the human safety driver in the Uber fatality of 2018, supervising a system operating faster and with more sensor data than any human could process in real time. This pattern is now migrating into software development tools and the stakes, while lower in consequence, are probably higher in volume. Grudin (2009) documented how AI and HCI spent decades talking past each other, developing separate vocabularies for what is ultimately the same problem: *how does a human work effectively with a system that knows more than they do, faster than they can process, across a task space they cannot fully observe?* The cognitive ergonomics tradition answered this question carefully for aviation, nuclear operations, and process control. The agentic coding tools are answering it again, from scratch, and seemingly without much reference to the previous answers. They are arriving, predictably, at the same failure modes.

To avoid going through the same errors, we propose that the first step should be towards effectively defining the Human-Automation partnership in the context of the new AI Agentic Systems.

3. Defining HAI against its neighbours

Human–Computer Interaction assumes the human executes and the computer assists. The Human–AI Interaction literature treats the AI as a single interactive surface to be made legible and correctable (Amershi et al., 2019; Shneiderman, 2020). Neither addresses what AI agentic systems put on the table: a job-to-be-done can be performed by the human, by the agent, or by both, and the question of whether a human should be in the loop has become detached from whether their presence does any work.

We define Human-Agent Interaction (HAI) as a design stance concerned with deciding, for each job-to-be-done, whether a human belongs in the loop at all; if so, where, why, and under what conditions their presence is cognitively honest rather than performative. Therefore, HAI's contribution is not a taxonomy of loop positions, it is a principle of refusing to claim oversight that the design cannot deliver.

The frameworks we draw on each answer a real question: Sheridan and Verplank (1978) and Parasuraman, Sheridan and Wickens (2000) tell us how much authority has been delegated; Endsley (1995) tells us what the human must perceive to act; Bainbridge (1983) documented what happens when the residual role is not performable. What we found, working with agentic tools daily, is that these models quietly assume the human's loop position is stable, assigned by design and held, with any transitions few, scripted, and trained for, as aviation's handover protocols illustrate. In agentic development, neither assumption holds. The position shifts constantly and without script, within a single task and sometimes within a single minute: plan approval is one position, watching execution is another, being asked mid-stream to permit a file write is a third, and no protocol governs any of these transitions. Once the position is that fluid, feasibility can no longer be a background finding, it has to

be assessed continuously, per job-to-be-done, which is why we make cognitive feasibility (C) an explicit dimension rather than an inherited assumption.

3.1. The fallacy at the centre of the market

"Human in the loop" has become the industry's reassurance phrase. It is invoked to license agent deployment in high-stakes domains, to satisfy regulators, and to absorb accountability, often without any examination of what the human in the loop is actually doing. When a reviewer faces 10,000 loan applications and a confident agent recommendation on each, the human is not in the loop. They are next to the loop, pressing approve. The system inherits a signature of human oversight it did not earn, and the organisation inherits a defence, "a human reviewed it", it cannot honestly mount. The EU AI Act's Article 14 requires that high-risk systems be designed so they "*can be effectively overseen by natural persons*", and explicitly warns deployers about automation bias (European Parliament & Council, 2024). The aspiration is correct. The implementation, in most deployments, is not, and this is worse than no human at all. A system without a human in the loop is at least honest about its autonomy; its failures attach to its designers. Green and Kak (2021) call this "*the false comfort of human oversight*": a legal and rhetorical device that protects institutions while doing little for the people the oversight was meant to protect. Crootof, Kaminski and Price (2023) make the related legal argument in their identification of the MABA-MABA trap, *Men Are Better At / Machines Are Better At,* originally named by Dekker and Woods (2002), in which policymakers respond to algorithmic incapacities by inserting a human into the decision without ensuring they have the authority, information, or time to exercise meaningful judgement.

The cognitive evidence is older and equally unambiguous. Bainbridge (1983) named the dynamic forty years ago: the ironies of automation. The more you automate, the more the residual human role degrades into monitoring, and the worse humans get at the monitoring. For instance, the switch to semi-automated vehicles intended human drivers to spend less time in basic tasks such as steering, and spend more attention monitoring the traffic environment. Nevertheless, automation increased the likelihood these drivers performed non-driving related tasks, thus reducing situation awareness and response times to alerts (Jansen et al., 2019).

The five patterns documented in Section 2.1 are the evidence of this claim. The streaming code/reasoning pattern cannot answer whether the cognitive work of verification is feasible under the conditions the interface creates. The permission request pattern cannot answer what specific cognitive work the human is supposed to perform when approving. The plan presentation pattern cannot answer whether the human's approval of the plan constitutes meaningful oversight of the execution. The context filling pattern cannot answer what the agent currently knows, which means neither can the human. The stop button cannot answer what loop position the human transitions into when they use it. These are not implementation failures, but rather patterns derived from a failure to ask the right questions before the design was built. The framework below provides those questions.

3.2. The decision space: accuracy, latency, risk, and cognitive feasibility

Three dimensions characterise most jobs-to-be-done. **Accuracy required (A)** which is the tolerance for incorrect output; **Latency tolerated (L)** which is the time available for cognition between action and consequence; and **Risk exposure (R)** which is the blast radius of an incorrect action. These three set the demand for human involvement. A fourth dimension sets the supply: **Cognitive feasibility (C)**: whether a human, under the conditions the interface actually creates, can perform the work the loop position would require. C is what most "human in the loop" claims silently assume and routinely violate. Reviewing 10,000 applications is not cognitively feasible. Reviewing a streaming code diff faster than it is generated is not either. Catching a subtle logic error in a confident agent output, when nine out of ten previous outputs were correct, is not cognitively feasible, and Parasuraman and Manzey's (2010) attentional model predicts exactly this failure. Green and Chen (2019) showed it empirically: reviewers presented with algorithmic recommendations underperformed the algorithm and introduced new disparities rather than catching old ones.

This decision space is helpful to decompose tasks and understanding the H*TL implication of each job-to-be-done. Let's walk through the example of "Unifying auth login flows", as it is an extremely complex task which can be decomposed into at least five jobs-to-be-done, each with a different A/L/R/C profile, resulting in a different honest loop position.

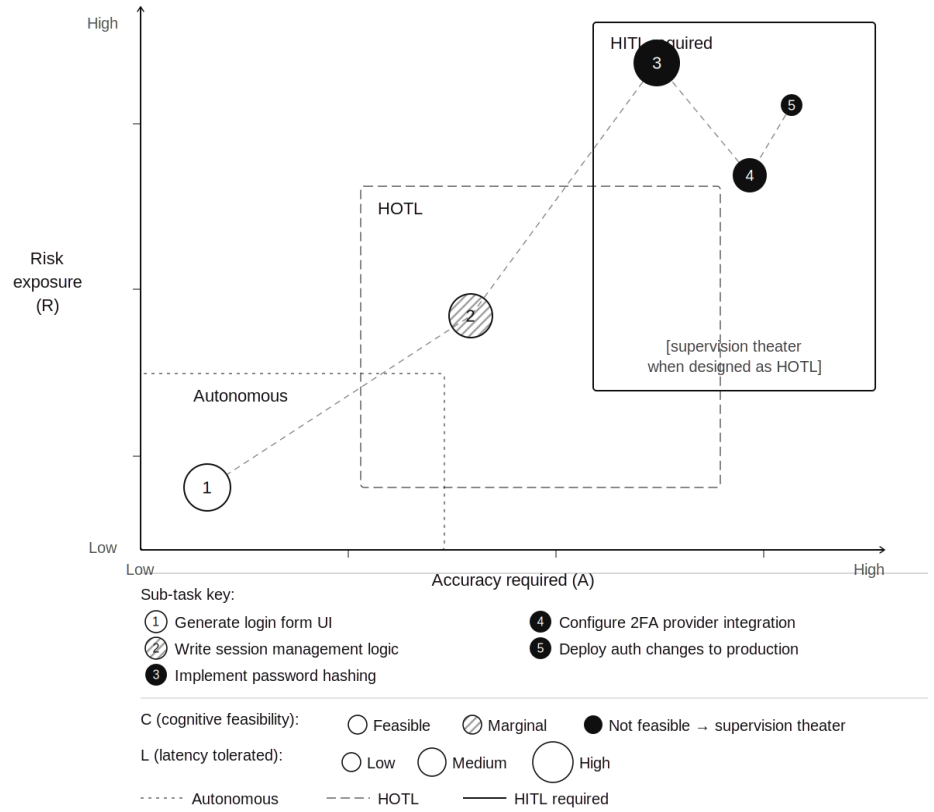


Figure 1 - The HAI decision space applied to the decomposition of a single task: "Unify auth login flows." Each numbered point represents one job-to-be-done, plotted by accuracy required (A), risk exposure (R), and latency tolerated (L, bubble size). Bubble fill encodes cognitive feasibility (C)

The figure makes visible something that neither the standard HITL/HOTL vocabulary nor the five-pattern analysis above can show directly: that jobs-to-be-done 4 and 5 in the Unifying auth login flows example (configure 2FA provider integration; deploy auth changes to production) look like reasonable HOTL candidates on the first three dimensions alone. Add cognitive feasibility, and they become visible as supervision theatre. The framework earns its place by making that category distinction possible. Without C as an explicit dimension, a designer looking at those tasks would see high-stakes work requiring human review and would design a review step. With C, they see that the review step, under the conditions current tools create, is not genuine oversight and they are forced to either redesign the conditions or own the autonomy.

3.3. The decision model

The framework to define a loop position asks four questions, in order:

1. **Does the job require human judgement at all?** Many do not. Low-A, low-R jobs are honest candidates for full agent autonomy, and inserting a human into them is a tax on the human. If every action requires approval, no action is meaningfully approved.

2. **If a human is required, what specific cognitive work must they perform?** Not "review", that is a verb that hides the work. "Detect" what kind of error? "Reconcile" what against

what? “Decide” between which alternatives? If the work cannot be named, the loop position cannot be designed.

3. **Is that work cognitively feasible under the conditions the interface creates?** Volume, time pressure, fluency of agent output, base rate of agent error, and the salience of the cues the human would need, these determine feasibility (Parasuraman & Manzey, 2010). If the answer is no, the loop position is theatre, and the honest response is to redesign the job (reduce volume, increase latency, add second-order checks) or to remove the human and own the autonomy.

4. **What would make the human's presence irreplaceable?** Forcing functions on attention, calibrated friction at moments of consequence, structured disagreement protocols, sampling regimes that make the work tractable, dashboards that surface the absent signal rather than the present one.

The output of the model is a defensible answer to the question: *should a human be in this loop, and if so, what does the design owe them to make their presence real?*

4. Design Implications: Four Questions, Four Commitments.

The decision model proposed above asks four questions, and each question produces a design commitment. Each commitment is also a direct response to a failure pattern documented in Section 2.1. The framework is not proposing new requirements, instead it is naming what the existing failures demonstrate was missing.

1. **Name the loop position.** A commit message and an authentication refactor are not the same kind of job, and the interface should say so before the agent acts. The practical move is a task-level declaration: this action runs autonomously; this one requires plan approval; this one will pause before any irreversible step. Magentic-UI (Mozannar et al., 2025) does this through its co-planning interface, which distinguishes tasks that warrant a direct response from those requiring an editable plan and explicit acceptance. The developer knows, before execution begins, whether they are expected to evaluate or simply to observe, and that expectation shapes how they pay attention.
   Claude Code's recently introduced /goal command works from the opposite end of the spectrum but honours the same commitment: the developer defines a completion condition and the agent works autonomously across multiple turns until it is met (Anthropic, 2026, release notes). What makes this honest is not the autonomy but the declaration, the interface states up front that no one will be watching turn by turn, and substitutes an explicit, checkable goal for the pretence of continuous supervision. Naming the position as unsupervised is still naming the position.

2. **Specify the cognitive work.** Telling a developer to "review" an agent's output names a role without specifying the work. Review of what, against what standard, to reach what decision? In agentic contexts it’s important to show not just what changed but what the agent considered and rejected. A diff shows output. A decision diff shows reasoning: "*I needed to handle the case where a user is not logged in. I could redirect to the login page or return an error. I chose the redirect because every other route in this codebase does the same.*" That gives the developer a decision to evaluate rather than a result to accept or refuse. Magentic-UI's answer verification mechanism moved in this direction, providing a structured action history at task completion. The authors learned that users found long action stories hard to monitor and asked for visual aids to quickly assess the agent’s work.

3. **Test for feasibility.** The relevant question about any transparency feature is not whether the information is shown but whether a developer, under realistic conditions, can act differently because of it. AI Agentic systems should tell the developer not just that the agent "*... is AI and can make mistakes*" (as per Claude's disclaimer), but how reliably it handles this class of task, so they know how closely to attend. The design test is simple: take any proposed transparency feature and ask what decision it enables. A stream of code passing at generation speed fails that test: the information is present, but the conditions make it unprocessable. A task-level summary such as "I changed how the application handles failed logins across four files; here is the before and after behaviour" passes it, because a developer can read in a fairly short amount of time and decide whether to look closer. During the validation of automation systems in vehicles time-and-motion tests are highly advisable, even if done in a simulated context, as these look to replicate the real world time and motion constraints that the human occupant faces when deciding how to act upon the system (ISO 34502:2022).

4. **Build forcing functions.** Forcing functions are a class of physical or logical constraints that make it impossible to perform an action incorrectly or out of sequence. They force the right behaviour by making the wrong behaviour unavailable (Norman, 1988). In AI agentic systems, the more pressing problem is conveying the consequences of agent actions before they become irreversible. The design commitment is to gate on risk, not on agent uncertainty. Magentic-UI's ActionGuard architecture operationalises this precisely: every agent action carries a pre-defined irreversibility heuristic (*always*, *maybe*, or *never* irreversible). The gate is triggered by the consequence category, not by agent uncertainty or agent confidence. Pushing to production, modifying authentication logic, or deleting data should require human approval not because the agent is hesitant but because the cost of an error is high. Permission scoping (read-only by default) is itself a forcing function that prevents most users from making consequential changes.

5. Conclusion: The Importance of Being Earnest

The cognitive engineering tradition answered the question of when human supervision works and what it requires. The agentic AI industry is answering the same question from scratch, without much reference to those answers, and arriving at known failure modes. We began this paper by proposing to explore what design for human-agent supervision should look like to be effective and cognitively honest. Our exploration concludes that current agentic coding tools, for most tasks, position the human as a signature, a presence in the loop that produces no different outcome from their absence. This stems from a lack of adequate language for problems of supervision in this agentic era. The framework to diagnose the correct position of the human in the loop, complemented with corresponding design commitments, serve as a first instance of that needed vocabulary. This does not fully solve the design problem, but makes the challenge visible. The design decisions being made now will be the defaults of the next decade. They should be grounded in what we actually know about human cognition under supervisory conditions. If the human cannot meaningfully supervise, own the autonomy. Some jobs-to-be-done should not have a human in the loop at all. If the cognitive feasibility analysis shows that genuine supervision is not achievable under the conditions the interface creates, the honest move is to remove the human, redesign the conditions, or not deploy. The vocabulary this paper offers is a precondition for making that choice more deliberately, rather than by default. On the other hand, if a human can supervise, let's design for it: name the loop position, specify the cognitive work, test for feasibility, and gate on risk. Accountability, for now, still rests where the law places it: with the human in the loop (European Parliament & Council, 2024). But the cognitive ground beneath that placement is shifting faster than the legal frame around it. The landscape is not yet settled, and we do not claim to settle it. We do insist that where accountability is assigned, the cognitive conditions that make it meaningful must be designed in, not assumed.

One assumption runs beneath the whole argument: that there is a human on the loop at all. The mid-term trajectory may well be agents reviewing agents, with the developer moving from on the loop to nowhere near it. We expect these commitments to hold even then, because underneath they are commitments about legibility and communication at the point of consequence, not about human attention specifically - but that is an expectation we have not yet tested, and we flag it as the boundary of what this paper claims.

In Wilde's *The Importance of Being Earnest*, the comedy depends on the gap between what characters perform and what they actually are. That gap is sustainable as farce. In the design of systems that carry real consequences, it is not.

Acknowledgements

We thank the PPIG reviewers, whose observations, in particular on the societal persistence of the distinction between writing and running software, sharpened the conclusion of this paper. We are equally grateful to the participants of the 37th PPIG workshop in London, whose questions, comments, and generous discussion improved this work, and whose warm welcome made presenting it a pleasure.

6. References

Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., Suh, J., Iqbal, S., Bennett, P. N., Inkpen, K., Teevan, J., Kikin-Gil, R., & Horvitz, E. (2019). Guidelines for Human–AI Interaction. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–13. https://doi.org/10.1145/3290605.3300233

Anthropic. (2026, May). *Claude Code release notes, v2.1.139* [Software changelog]. GitHub. https://github.com/anthropics/claude-code/blob/main/CHANGELOG.md

Bainbridge, L. (1983). Ironies of automation. *Automatica*, 19(6), 775–779. https://doi.org/10.1016/0005-1098(83)90046-8

Brockman, G. (Guest), & Parrish, S. (Host). (2026, April 22). *Greg Brockman: AI Is About to Go Parabolic! Here's What's Next*. In *The Knowledge Project Podcast*. YouTube. https://www.youtube.com/watch?v=6JoUcQ1qmAc

Crootof, R., Kaminski, M. E., & Price, W. N., II. (2023). Humans in the loop. *Vanderbilt Law Review*, 76(2), 429–510.

Dekker, S., & Woods, D. D. (2002). MABA-MABA or Abracadabra? Progress on human–automation co-ordination. Cognition, Technology and Work, 4, 240–244. https://doi.org/10.1007/s101110200022

Elish, M. C. (2019). Moral crumple zones: Cautionary tales in human-robot interaction. *Engaging Science, Technology, and Society.* https://doi.org/10.17351/ests2019.260

Endsley, M. R. (1995). Toward a Theory of Situation Awareness in Dynamic Systems. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, *37*(1), 32-64. https://doi.org/10.1518/001872095779049543

Endsley, M. R., & Kiris, E. O. (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), 381-394. https://doi.org/10.1518/001872095779064555

European Parliament & Council. (2024). *Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act)*. Official Journal of the European Union, L series, 12 July 2024.

Green, B., & Chen, Y. (2019). Disparate interactions: An algorithm-in-the-loop analysis of fairness in risk assessments. *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT* '19)*, 90–99.

Green, B., & Kak, A. (2021, June 15). The false comfort of human oversight as an antidote to A.I. harm. *Slate*. https://slate.com/technology/2021/06/human-oversight-artificial-intelligence-laws.html

Grudin, J. (2009). AI and HCI: Two fields divided by a common focus. *AI magazine*, *30*(4), 48-57. https://doi.org/10.1609/aimag.v30i4.2271

Horvitz, E. (1999). Principles of mixed-initiative user interfaces. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99), 159–166. https://doi.org/10.1145/302979.303030

International Organization for Standardization. (2022). *Road vehicles - Test scenarios for automated driving systems - Scenario based safety evaluation framework* (ISO Standard No. 34502:2022). https://www.iso.org/standard/78951.html

Jansen, R. J., van der Kint, S. T., & Hermens, F. (2019). Does agreement mean accuracy? Evaluating glance annotation in naturalistic driving data. Behavior Research Methods, 51(6), 2520–2535 https://doi.org/10.3758/s13428-020-01446-9

Kahneman, D. (2011). *Thinking, fast and slow*. Farrar, Straus and Giroux.

Moran, T. P., Newell, A., & Card, S. (1983). GOMS (Goals, Operators, Methods, and Selection rules). *The Psychology of Human Computer Interaction*. Lawrence Erlbaum Associates.

Mozannar, H., Bansal, G., Tan, C., Fourney, A., Dibia, V. C., Chen, J., Gerrits, J., Payne, T., Maldaner, M. K., Grunde-McLaughlin, M., Zhu, E., Bassman, G., Alber, J., Chang, P., Loynd, R., Niedtner, F., Kamar, E., Murad, M., Hosn, R., & Amershi, S. (2025). Magentic-UI: Towards Human-in-the-loop Agentic Systems. *arXiv preprint arXiv:2507.22358*. https://doi.org/10.48550/arXiv.2507.22358

Norman, D. A. (1986). Cognitive engineering. In *User centered system design* (pp. 31-62). CRC Press.

Norman, D. A. (1988). *The psychology of everyday things*. Basic books.

Parasuraman, R., & Manzey, D. H. (2010). Complacency and bias in human use of automation: An attentional integration. *Human Factors*, 52(3), 381–410. https://doi.org/10.1177/0018720810376055

Parasuraman, R., Sheridan, T. B., & Wickens, C. D. (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), 286-297. https://doi.org/10.1109/3468.844354

Rasmussen, J. (1986). A framework for cognitive task analysis in systems design. In *Intelligent decision support in process environments* (pp. 175-196). Berlin, Heidelberg: Springer Berlin Heidelberg.

Rzeszucinski, P. (2024, February 29). *AI, humans and loops*. Medium. https://medium.com/@pawel.rzeszucinski_55101/ai-humans-and-loops-04ee67ac820b

Sebok, A., Wickens, C., Laux, L., & Jones, M. (2015). Supporting human-automation interaction in the rail industry by applying lessons from aviation. In *Proceedings of the Human Factors and*

Ergonomics Society Annual Meeting (Vol. 59, No. 1, pp. 1661-1665). Sage CA: Los Angeles, CA: SAGE Publications. https://doi.org/10.1177/1541931215591359

Shaw, S. D., & Nave, G. (2026). *Thinking, fast, slow, and artificial: How AI is reshaping human reasoning and the rise of cognitive surrender*. OSF Preprints. https://doi.org/10.31234/osf.io/yk25n_v1

Sheridan, T. B. (1981). Understanding human error and aiding human diagnostic behaviour in nuclear power plants. In *Human detection and diagnosis of system failures* (pp. 19-35). Boston, MA: Springer US.

Sheridan, T., & Verplank, W. L. (1978). *Human and computer control of undersea teleoperators*. A Man-Machine Systems Laboratory Technical Report.

Shneiderman, B. (2020). Human-centered artificial intelligence: Reliable, safe & trustworthy. *International Journal of Human–Computer Interaction*, 36(6), 495–504. https://doi.org/10.1080/10447318.2020.1741118

Silva, C., Masci, P., Zhang, Y., Jones, P., & Campos, J. C. (2019). A use error taxonomy for improving human-machine interface design in medical devices. *ACM SIGBED Review*, *16*(2), 24-30. https://doi.org/10.1145/3357495.3357498

Wickens, C. D. (1995) Designing for Situation Awareness and Trust in Automation. *IFAC Proceedings Volumes*, Volume 28, Issue 23. https://doi.org/10.1016/S1474-6670(17)46646-8

Just tell the robot what to do, in VR

Anahide Silahli
The Maersk M. Moller Institute
University of Southern Denmark
Odense, Denmark
ansil@mmmi.sdu.dk

Thiago Rocha Silva
The Maersk M. Moller Institute
University of Southern Denmark
Odense, Denmark
thiago@mmmi.sdu.dk

Robotic systems are increasingly deployed across industrial and service domains; however, their adoption remains limited by the scarcity of skilled programmers. Training domain experts to configure robotic systems is resource-intensive, motivating the need for more accessible programming approaches. End-User Development (EUD) aims to enable non-experts to create and modify software (Lieberman, Paternò, Klann, & Wulf, 2006), with End-User Programming (EUP) focusing on simplified programming paradigms. In robotics, this leads to End-User Robot Programming (EURP), which seeks to democratize robot interaction (Ajaykumar, Steele, & Huang, 2021). Existing approaches, including visual programming and programming by demonstration, reduce complexity but still impose structured interaction models (Vaiani & Paternò, 2024). Recent advances in Large Language Models (LLMs) offer a complementary paradigm by supporting free-form natural language interaction (Karli, Chen, Antony, & Huang, 2024).

In our previous work, we introduced a web-based system that combines an LLM with a Domain-Specific Language (DSL) to support intuitive yet verifiable end-user programming of a robot arm. DSLs provide structured, domain-tailored representations (Fowler, 2010; Mernik, Heering, & Sloane, 2005), while LLMs enable unconstrained user input. Our approach integrates both by translating natural language instructions into a reviewable intermediate representation based on Behavior-Driven Development (BDD) (Smart & Molak, 2023) scenarios. Users interact through a conversational interface, formulating their intentions via text or speech. The LLM then generates one or more scenarios expressed in a BDD-inspired DSL using a *Given–When–Then* structure that can be visualized either as text or as a block-based representation. This intermediate layer makes robot behavior explicit and human-readable, allowing users to inspect and edit generated programs before execution. In contrast to fully automated LLM-based systems, this design emphasizes transparency and user control.

The demonstration extends this work by coupling this EUP framework with an immersive virtual reality (VR) environment in which a mobile manipulator executes logistics tasks in a warehouse. VR provides a realistic yet safe setting for observing robot behavior, improving spatial understanding compared to traditional 2D simulations. It also opens opportunities for future multimodal interaction, such as gesture-based or spatial commands, complementing natural language input. Our contribution illustrates how combining LLMs with a structured DSL enables a hybrid interaction paradigm that balances usability and reliability. The BDD-based intermediate layer serves as a bridge between free-form user intent and executable robot behavior, supporting validation, transparency, and trust. By integrating conversational programming with immersive visualization, the system contributes to ongoing efforts in EURP to broaden access to robotic technologies for non-experts.

1. References

Ajaykumar, G., Steele, M., & Huang, C.-M. (2021). A survey on end-user robot programming. *ACM Computing Surveys (CSUR)*, *54*(8), 1–36.

Fowler, M. (2010). *Domain-specific languages*. Boston, MA: Addison-Wesley Professional.

Karli, U. B., Chen, J.-T., Antony, V. N., & Huang, C.-M. (2024). Alchemist: LLM-aided end-user development of robot applications. In *Proceedings of the 2024 acm/ieee international conference*

on human-robot interaction (pp. 361–370).

Lieberman, H., Paternò, F., Klann, M., & Wulf, V. (2006). End-user development: An emerging paradigm. In H. Lieberman, F. Paternò, & V. Wulf (Eds.), *End user development* (Vol. 9, pp. 1–8). Dordrecht: Springer.

Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR)*, *37*(4), 316–344.

Smart, J. F., & Molak, J. (2023). *Bdd in action: Behavior-driven development for the whole software lifecycle*. Simon and Schuster.

Vaiani, G., & Paternò, F. (2024). End-user development for human-robot interaction: Results and trends in an emerging field. *Proceedings of the ACM on Human-Computer Interaction*, *8*(EICS), 1–40.

NodeToJoy: Understanding Probabilistic State through Interactive Visualisation

Charlotte Cox
Computer Laboratory
University of Cambridge
cc2207@cam.ac.uk

Alan F. Blackwell
Computer Laboratory
University of Cambridge
afb21@cam.ac.uk

Abstract

As probabilistic machine learning systems are deployed increasingly widely, computational literacy will require better understanding of the fundamental distinctions between conventional deterministic models of computation, and probabilistic behaviours. This may be especially important in cases that seem superficially similar, such as text string generation via sequence prediction in LLMs. We introduce NodeToJoy[1], a visualisation that juxtaposes a Finite State Machine visualisation with a Markov Chain visualisation, both adapted to the exploration of large and complex datasets by interactive hierarchical semantic zooming (including a novel extension of Harel's Statecharts to Markov Chains). Our paper reports classroom studies demonstrating that lessons including interaction with NodeToJoy significantly advance student understanding of these fundamental principles.

1. Introduction

The way computers appear to 'understand' what users mean remains a mystery to many young students. Traditionally, a computer's apparent 'understanding' of language could be explained through lexical analysis and parsing - deterministic, rule-governed systems, where Finite State Machines (FSMs) play a significant role. Large language models (LLMs) might appear superficially similar effect to non-technical users, in the sense that they interpret a sequence of input text. Yet, the underlying probabilistic model is formally distinct from the rule-governed manipulation of state that has historically been central to the mathematics of computation (Vidal, Thollard, de la Higuera, Casacuberta, & Carrasco, 2005), and would be better described in mathematical terms as a Markov Chain (MC). Emphasising the difference between rule-based and probabilistic-inference machines can shape a more informed engagement with such tools, and potentially empower students to think critically about interactions with LLMs.

Despite the computational distinction, both FSM and MC models are state-based formalisms. Both can be represented diagrammatically in node-link form, where states correspond to notes, and links represent either transition likelihood, or allowable transitions. FSMs are generally introduced in this way, where the metaphor of state as a visual location in (a "state space") aids spatial intuition. MCs are often taught through algebraic notation or matrix representation of transition probabilities, but graphical rendering significantly improves user response times (Keller, Eckert, & Clarkson, 2006), and visualisation has been shown to enhance learning (Kim & Jin, 2022; Presmeg, 2006; Gates, 2018; Elliott, Hudson, & O'Reilly, 2000; Wang, Han, & Chou, 2018).

We start from this diagrammatic motivation to explore an urgent educational need. Although UK students studying Computer Science at high-school (A Level, corresponding to 11th and 12th grades in the USA) are introduced to Finite State Machines (AQA, n.d.), no existing course touches on probabilistic models of computation. Following recent developments elsewhere, reforms are now planned for the UK national curriculum with a focus on probabilistic literacy and real-world applications (Royal Statistical Society, 2024). As a teaching strategy, the Royal Statistical Society already recommends greater use of visualisation software for Data Science and AI (Royal Statistical Society, 2024),(Mathematics in Education and Industry (MEI), n.d.).

[1] A reviewer has asked us to explain our choice of name. For decades, students of theoretical computer science have looked upon Finite State Machines and Markov Chains as cold, unfeeling arrays of circles and arrows. Just as Beethoven shattered classical boundaries by infusing his final symphony with vocal passion, NodeToJoy transforms static state diagrams into a living chorus.

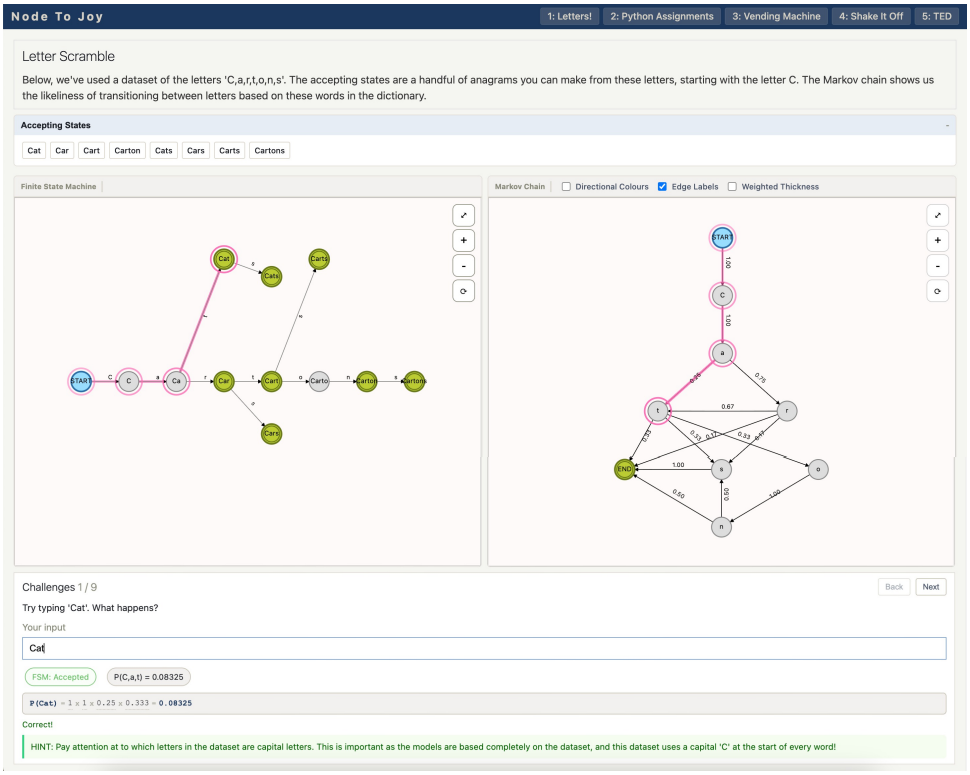


Figure 1 – The first page of NODETOJOY*, featuring a simple FSM, left, and MC, right.*

A foundational question for AI literacy in this turn toward probabilistic system behaviour is for students to gain a critical understanding of the difference between computational concepts of *validity* and of *likelihood*. Our system NODETOJOY seen in figure 1, allows students to explore and contrast these concepts through interactive parallel visualisations of Finite State Machines and Markov Chains.

1.1. Previous FSM and MC visualisations

Literature on educational FSM and MC visualisation addresses the two models in isolation: no existing platform explores their juxtaposition for novice learners. The most popular educational tool for visualising FSMs is JFLAP, a visual simulation platform (Rodger, 2006). The expressive power of JFLAP is valuable for undergraduate students, but less appropriate to novices in their first-time encounter with automata. Other studies have explored methods around gamification and narrative to make FSMs more approachable with interactivity and visualisation. An identified lack of K-12 resources led to Castro's game-building tool using FSM blocks (Castro, 2022), whereas introducing FSMs through storytelling improved approachability and memory retention for graduate students (Iudean & Vescan, 2025). Dengel combined both strategies, suggesting a Virtual Reality scenario of acting within a FSM, emphasising the immersive 'unplugged' benefits to engagement (Dengel, 2018).

Markov chain education is often centred on transition matrices and probabilistic calculation, rather than visual tools. Pfannkuch and Budgett addressed this gap through an interactive tool linking transition matrices with their equivalent diagram in real time, finding that visualisation aided students' conceptual understanding (Pfannkuch & Budgett, 2016). In a similar domain of Markov decision processes, Mutua found that graphical visualisations employing narrative were more effective than textbook learning for undergraduates in Kenya (Mutua & Blackwell, 2023). This project aims to work at the intersection of these interactive and playful strategies.

1.2. Contributions

We have created an interactive educational platform to juxtapose two foundational but contrasting models of computation: finite state machines and Markov chains. The goal is to help students who may be unfamiliar with these models to develop foundational understanding of both. Importantly, this en-

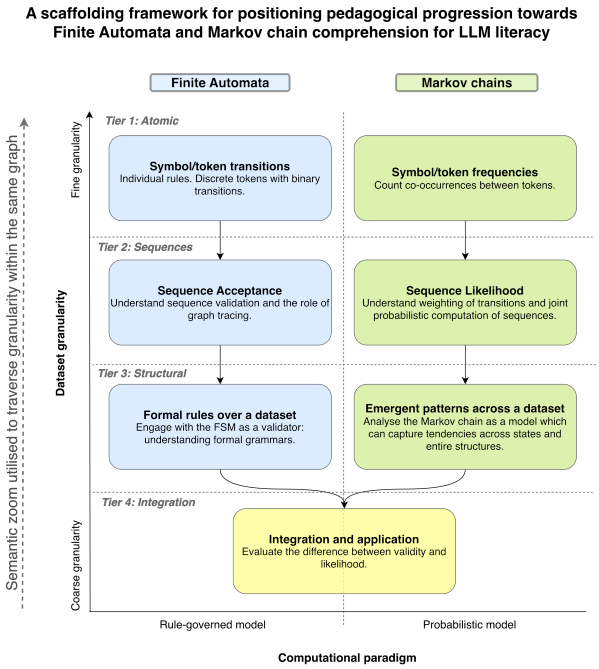


Figure 2 – Framework for describing the range of datasets that may be used in NODETOJOY*, for reference in justifying design choices. Dataset scale across these positions can range from toy illustrative to pedagogically reinforcing to statistical, large corpora.*

ables dialogue around the difference between validity and likelihood, questions that are critical to the stochastic behaviour of LLMs. NODETOJOY makes a number of key contributions towards extending computational model visualisations in education. It creates robust node-link graph visualisations for both FSMs and MCs across a wide range of scale and complexity, with integrated interactive techniques such as semantic zooming, node manipulation, and sequence tracing to promote active engagement with the models.

Most notably, this project draws on David Harel's statecharts to extend both the finite state machine and Markov chain visualisations through hierarchical structure (Harel, 1987). Whilst statecharts have become an important component of the UML framework, research on *Markov statecharts* - an extension identified, but not pursued, by Harel in the 1987 paper "On Statecharts" - has been sporadic across disciplines and underexplored. The use of hierarchy is invaluable for pedagogical scaffolding, abstraction, and greater expressiveness (Melero, Hernández-Leo, & Blat, 2012). It facilitates modelling real-world applications through information hiding, reducing educators' reliance on contrived datasets for visual simplicity. We describe a novel platform and visualisation framework, which we believe is the first to combine hierarchical FSMs and Markov chains, and also the first to utilise semantic zoom to traverse levels of depth in statecharts. The project's effectiveness was evaluated through two in-school user trials with the target audience of secondary school students in year 12 of the English school system (also called Lower Sixth, equivalent to USA 11th Grade or High School Junior). These provide evidence that NODETOJOY is highly effective in increasing students' understanding of finite state machines, Markov chains, sequence processing, and the probabilistic nature of LLMs.

2. A framework for computational state literacy

Our central claim is that the conceptual distance between rule-governed and probabilistic models of computation can be bridged through traversable interaction design. To provide a principled basis for the design decisions made throughout implementation, this section provides a framework to characterise the conceptual space this platform navigates. The main axes of concern are granularity, scale, and computational paradigm.

Granularity describes the level of abstraction at which a student engages with the model. Following Bloom's taxonomy (Forehand, 2007), learners' progression of comprehension skills map onto three tiers, illustrated by figure 2. **Atomic (Tier 1)** granularity focuses on individual symbol transitions, before composing these to parse **sequences (Tier 2)**. As students understand the models' functionality, states can be abstracted into **structures (Tier 3)**, reflecting abstraction. This dissertation implemented support for all tiers: clear node layout to ease graph tracing in atomic granularity, path-highlighted sequence computation for sequential granularity, and statecharts to support structural granularity. Semantic zoom is the mechanism by which students can navigate **between tiers of granularity within a singular graph visualisation**, supporting the integration of knowledge for Tier 4.

Independent of granularity, corpus scale was also considered throughout development. As students begin learning, toy datasets are often used by educators for illustration, before progressing to pedagogically-scoped, scale-bounded, reinforcement datasets, eventually to large, statistical corpora of real-world data. Clear layout for all dataset scales was considered throughout development: layout-wrapping and viewport culling ensure smooth rendering performance for large corpora. To verify that datasets of the expected JSON format could be integrated in the project without bespoke changes, additional speech text corpora containing 500+ nodes were parsed, tested, and successfully laid out clearly without dataset-specific changes.

We selected teaching datasets at different positions within this framework, providing a scaffolded progression from fine-granularity detail to coarse-granularity exploration(Melero et al., 2012). The lesson plan presentation of NODETOJOY consists of four themed pages in which an FSM and MC for each of these datasets are presented in side-by-side panels. A fifth themed page presents just the Markov chain for a very large corpus, with fifty TED Talks parsed into fine-granularity bigrams that allow students to engage in higher-order, exploratory learning.

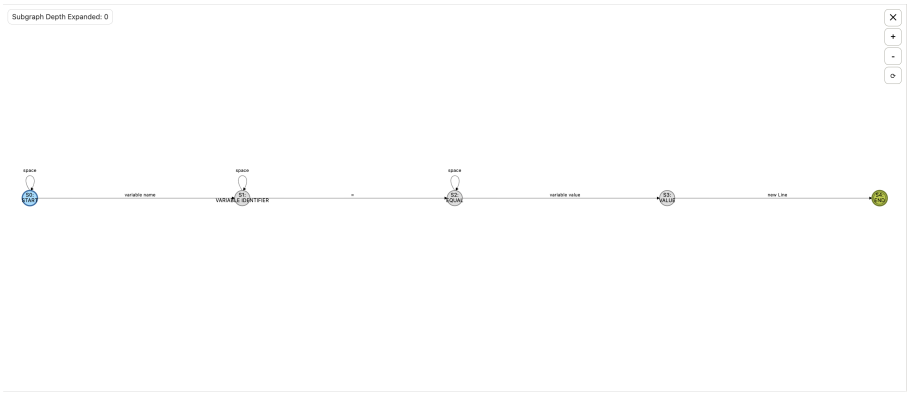
3. Visual interaction strategies

Our design employs a number of graph visualisation and interaction strategies to ease the cognitive load of novice users exploring potentially very large graphs (Sweller, 2016). We make extensive use of semantic zooming, adding detail as the user zooms in, to progressively reveal levels of statechart hierarchy (Cockburn, Karlson, & Bederson, 2009). A focus+context strategy is employed to ensure that revealed detail can be understood within a graph's wider context (Furnas & Bederson, 1995). In the Markov chain viewer, this is enacted through viewport-sensitive edge-culling; only rendering the edges between visible nodes when a user zooms in, which preserves surrounding context without visual noise. Baudisch and Rosenholtz's 'halos' are also used to visualise off-screen nodes: rectangles are nested around subgraphs to preserve the context of expanded objects (Baudisch & Rosenholtz, 2003).

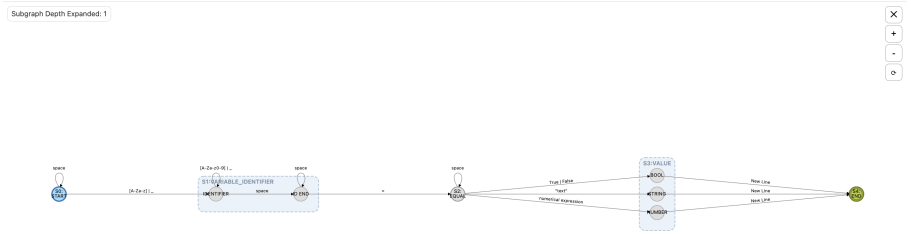
Combining hierarchical graph nesting with semantic zoom offers a novel interactive traversal mechanism between fine and coarse grained datasets. When a user zooms past a semantic zoom threshold (a depth of granularity in the dataset) a parent's node and edges disappear and are replaced by its child subgraph in the correct position. On collapse, the reverse must occur without corrupting node positions that the user may have manually adjusted. When a parent node is expanded, its incident edges are left dangling with no semantic meaning. To maintain the sequential flow of edges in a FSM, these transitions must be rerouted to preserve the visual continuity between the parent's subgraph and the surrounding graph.

3.1. Markov chain hierarchical subgraphs

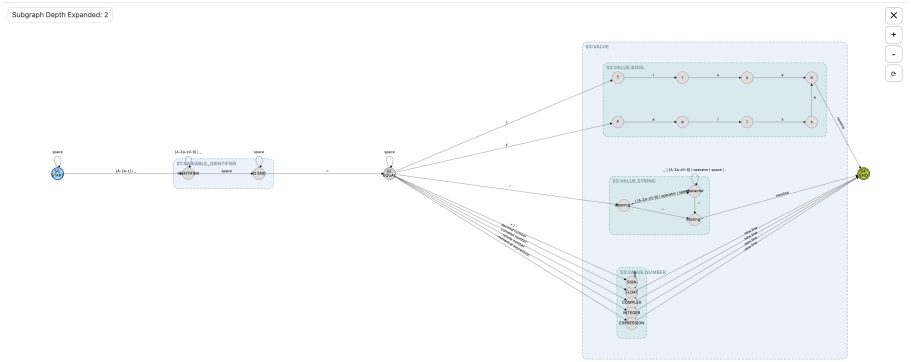
A coarsely-grained MC dataset presupposes probabilistic transitions between structural states represented as nodes. Within each structural state may reside a finer-grained subgraph, meaning that a coarsely-grained state can contain its own probability distribution over a finer-grained state space. Hierarchical MCs thus reflect the goal of granularity traversal through semantic zooming while preserving structural probability. Each subgraph is rendered within its halo, with coarse-level, inter-structure transitions drawn as arrows against each bounding rectangle using parametric ray-intersection. This preserves the user's understanding that each coarse state's subgraph is locally isolated but globally situated within a probabilistic state space. Inter-structure transition arrows are rendered as dashed Bezier curves to



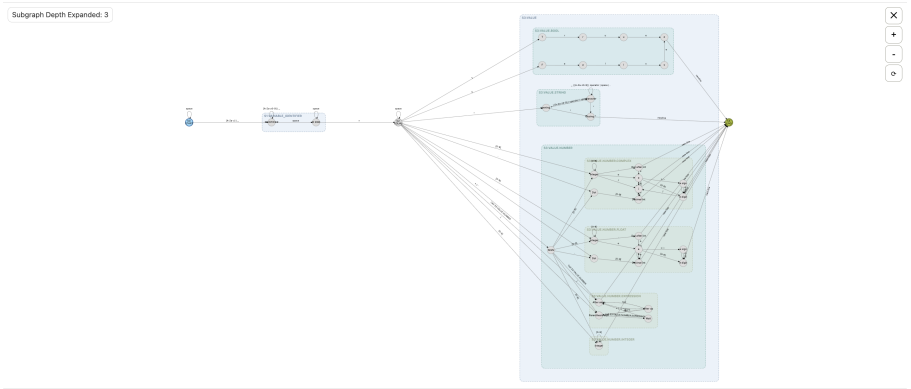
(a) Python hierarchical FSM before subgraph expansion



(b) Hierarchical FSM expanded to depth 1



(c) Hierarchical FSM expanded to depth 2



(d) Hierarchical FSM expanded to depth 3: letter-level parser

Figure 3 – Expansion progression of a hierarchical FSM across 3 nested depths.

Dataset	Granularity tier	Scale	Models Implemented	Pedagogical Reasoning
Subset of anagrams of "Cartons"	Fine-grained: character-level atoms	Illustrative toy	Flat FSM (Partial DFA) + flat MC	A simple letter level parser to familiarise students with both models. Connects FSM structure to compiler tokenisation.
Python assignments	FSM: Coarse-grained abstract structures → fine-grained character-level atoms MC: fine-grained character-level atoms	Pedagogical reinforcement	Hierarchical FSM (NFA with 3 levels of subgraph nesting) + flat MC	Hierarchical FSM across 4 depth levels walking students from complete abstraction to regex-like letter parsing. MC dataset of 1000 real world assignments.
30p Vending Machine	Structural whole: state represents cumulative currency	Illustrative toy	2x Flat FSM (DFA) + Flat MC	Demonstrates state explosion by adding a new valid coin to dataset 2. Illustrates FSM/MC uses beyond language parsing to construct an informed mental model.
Shake It Off lyrics, by Taylor Swift	FSM: Structural whole (suffix automaton) MC: Coarse-grained song structure → fine-grained word-level tokens	Exploratory analysis of statistically-sized dataset	Flat FSM (Partial DFA) + Hierarchical MC (1 level of hierarchy)	Demonstrates unsuitability of FSM for some real-world problems, alongside hierarchical MC traversal of an engaging context.
TED Talks (×50)	Structural whole: word-level tokens	Exploratory analysis of statistically sized dataset	50x Flat MCs	Student-directed exploration at scale (500+ nodes). Supports higher-order thinking about emergent patterns in natural language.

Table 1 – Datasets used in NODETOJOY*, positioned within the framework for comparative computational literacy.*

differentiate them from solid intra-state edges, with obstacle avoidance to prevent intersections with subgraphs. Edge thickness is linked to the probability of transition, providing a visual dual-coding of probability seen in figure 4.

Given that a subgraph may nest arbitrarily many times, there must be a graphical notion of sub-subgraphs to visually support user's understanding of hierarchy. This is addressed by encapsulating each subgraph in a rectangular "halo" container (Baudisch & Rosenholtz, 2003), of which sub-subgraphs reside in their parent subgraph's halos - visible in figures 3c and 3d.

3.2. Secondary notations

Interactive engagement is core to our pedagogical design aim; active learning through manipulation is encouraged, as guided by constructivism (Ben-Ari, 1998). Accordingly, interaction must be low-stakes, reversible, and consistent with the graph's underlying structure. For example, the default graph layout may not match a student's mental model (Ben-Ari, 1998), so we support dragging for users to organise the graph to reflect their own understanding. Semantic zooming is core to hierarchy traversal, but deep zoom levels risk contextual disorientation (Touya, Gruget, & Muehlenhaus, 2023). Geometric (rather than semantic) zoom buttons let users pan out to view the current subgraph within the graph's global context.

3.3. Focus+context filtering

Large Markov chain graphs present a density problem specific to natural language datasets, which are key to this project's LLM literacy aim. Following Zipf's law (Piantadosi, 2014), some words will occur with disproportionately high frequency in corpora, producing densely connected nodes whose edges

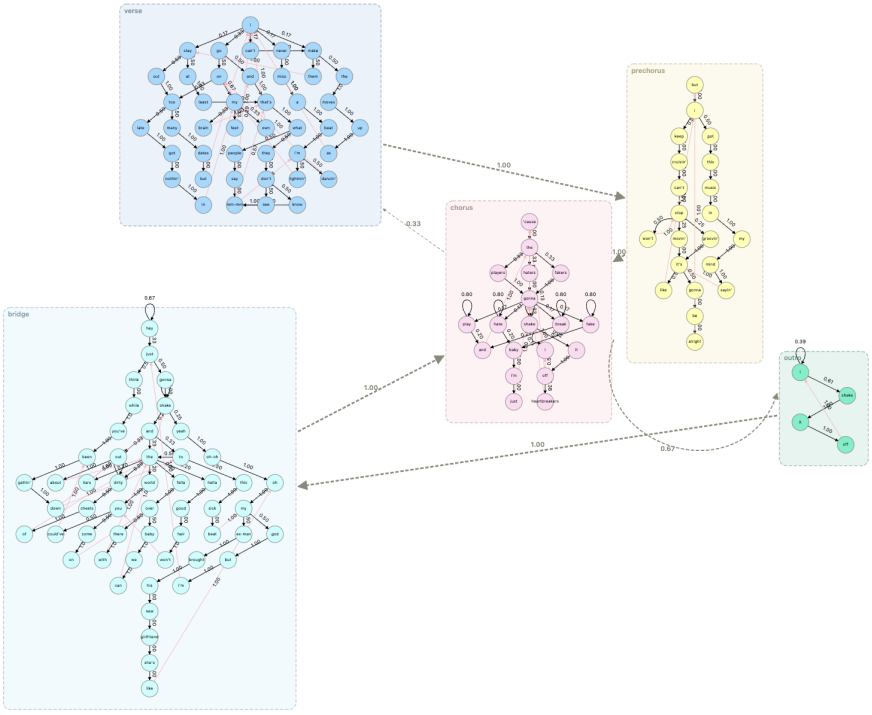


Figure 4 – Hierarchical Markov statechart with subgraphs expanded

become visually occluded. A focus+context approach (Furnas & Bederson, 1995) is taken from 3 filtering angles in the Markov viewer to address this: viewport culling to suppress edges when not zoomed in, per-node focus fades all non-incident edges and nodes to opacity 0.15, and dataset-item focus to highlight the subgraph for a specific piece of input data as in figure 5.

3.4. Sequence inspection

For guided lesson engagement, NODETOJOY also provides a panel of interactive challenges. This allows students to test hypotheses about the graph: inputting sequences and receiving immediate feedback on probability/validity computation. As a student inputs a sequence, the platform highlights the path taken by their sequence on flat graphs. This is important for drawing cross-form connections between tracing the graph and linking to their real-world purposes by seeing live validity/probability computation.

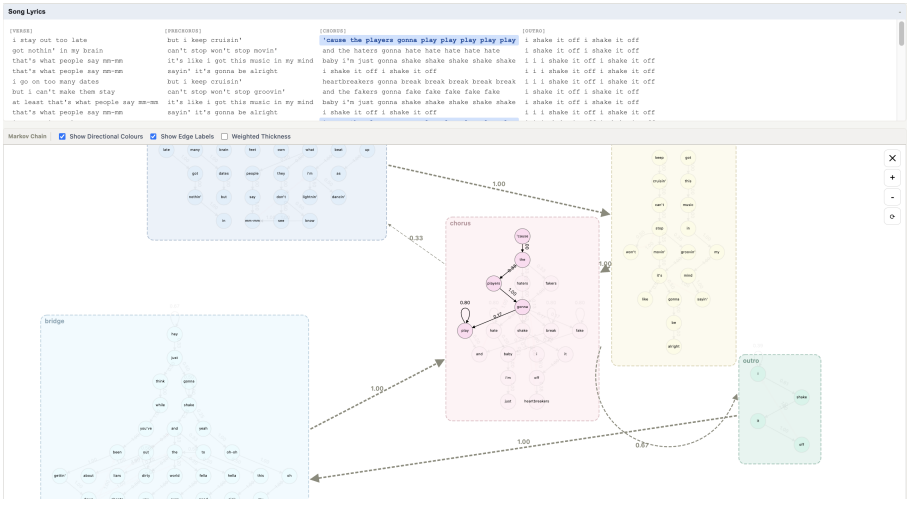


Figure 5 – Dataset item selected and highlighted in Markov chain subgraph

4. Classroom evaluation

We have evaluated NODETOJOY in two in-school trials. This study was approved by the Ethics Committee of the University of Cambridge Department of Computer Science and Technology. The lessons were delivered by the first author who has appropriate DBS-certification. Students were assigned an animal name as a pseudo-anonymous identifier to connects Likert-scale ratings, open-ended question answers and logged platform interactions.

4.1. Method

Each of the two lessons followed the same 1 hour structure:

1. Students completed an open response question to gauge their pre-lesson understanding: "The use of Large Language Models in platforms like ChatGPT have made computers increasingly able to write convincingly human-like text. How is the way a computer reads code different to the way it may read an essay or poetry?"

2. The researcher/teacher led a short presentation of the intuition behind FSMs/MCs.

3. Students spent the majority of the lesson (over 40 minutes) engaging with NODETOJOY, with interaction events logged anonymously.

4. The open question from the start of the lesson was repeated, for post-lesson evaluation of gained understanding.

5. The lesson closed with a Likert questionnaire. Students assessed their level of understanding before and after using the platform across key learning objectives.

Responses to the pre- and post-lesson question, were classified into four categories: (1) incorrect or no answer, (2) some evidence of computational thinking, (3) some understanding of FSMs or Markov chains, (4) comparative discussion of FSMs against Markov chains. Within-subjects comparison used a Wilcoxon signed-rank test. Students also self-assessed their confidence of each concept on a scale of 1 (no understanding) to 7 (confident understanding). To reduce bias when measuring program effectiveness, Nimon, Zigarmi & Allen's work informed the questionnaire structure (Nimon, Zigarmi, & Allen, 2011). Interaction logs recorded 18 event types, covering graph manipulation, dataset navigation, sequence input, and question progression.

4.2. Results summary

In both lessons, results found a consistent, significant gain in student understanding. A pooled repeated-measures multivariate test was computed with time (pre-lesson versus post-lesson) as the within-subject factor. Across all 24 students and 6 concepts, the test yielded $T^2 = 119.50$, $F(6,18) = 15.59$, $p = 0.000003$, providing strong evidence of effect on learning. 22 students increased their Likert score assessment across every concept and both cohorts converged to a post-lesson median confidence of 6/7 in every concept (except AI explanation at All Saints' with a median of 5.5). This consistency across both groups is particularly encouraging, suggesting that the platform successfully scaffolds these complex topics to support learning from a range of prior knowledge baselines.

4.3. Year 12 non-CS class

The first lesson took place at Hills Road Sixth Form College, Cambridge, as part of an extra-curricular Data Science class with eighteen year 12 students (one left early). Most students in the class studied A Level mathematics; two studied A Level computer science, so had learned FSMs as part of the AQA curriculum (AQA, n.d.). A small group expressed an extra-curricular interest in Markov chains from YouTube videos, although were unfamiliar with the model's formalisms. Most students accessed NODETOJOY on a laptop or iPad, with some opting to use desktop computers and one using their mobile phone. The platform worked well across all devices, with the graphs using dynamic layout resizing and pinch-to-zoom functionality for devices without a trackpad or mouse.

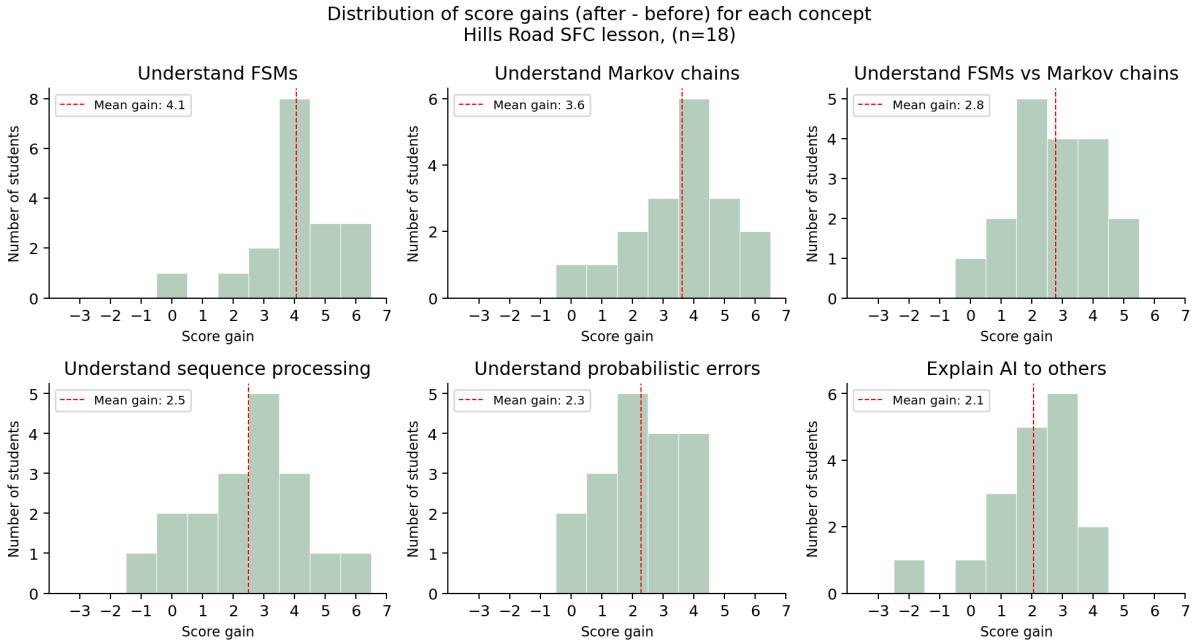


Figure 6 – Distribution of score gains per Likert scale question at Hills Road

NODETOJOY proved accessible against the wide range of knowledge backgrounds. Students new to the concepts often engaged by dragging and investigating the graphs, alongside answering structured exploration questions. More confident students often progressed to the later datasets; they conversed with peers about patterns they noticed across different datasets, as intended with the exploratory, statistically-sized corpora. Many students responded positively about their interest in the use of the models, with their teacher noting the high-level of engagement amongst students.

Students showed highly significant gains in comprehension across all six categories, with an average gain of 2.88. A repeated-measures multivariate test for the 18 students across 6 concepts at Hills Road showed a significant overall effect, with $T^2 = 156.28$, $F(6, 12) = 18.39$, $p = 0.000021$.

The largest mean gains were in FSM and MC understanding, consistent with most students being unfamiliar with these concepts before the lesson. Students' pre-lesson understanding of more applied topics (probabilistic errors and AI) was high (medians of 3.5 and 3.0), reflecting general awareness of LLM behaviour. Responses converged to the top of the scale on every criterion, and all six comparisons were highly significant ($p<0.001$), providing strong evidence that a single lesson with NODETOJOY led to a meaningful increase in self-reported understanding across all target concepts. Individualised score changes can be seen in figure 6.

Figure 7 highlights the shift in open response categorisation across the lesson. The progression is clear, from a majority of students showing no/limited evidence of computational thinking, to sixteen students discussing FSMs and MCs explicitly. Before the lesson, many students applied general computing knowledge to the question without any formal terminology. One student "Chipmunk" gave long and relatively sophisticated answers, demonstrating prior knowledge. Post-lesson, their understanding was explicitly linked to the datasets explored, discussing the use of FSMs as Python syntax validators. This suggests that NODETOJOY can help solidify and encourage application of knowledge for particularly able students.

Students with no prior knowledge also demonstrated improved understanding. "Alligator" scored only 1 pre-lesson, but gave an elegant category post-lesson response discussing FSMs as syntax validators versus AI using probabilities to "generate a prediction of the statistically most likely outcome". A further interesting response from "Parrot" included an unprompted, insightful comparison of FSM and MC

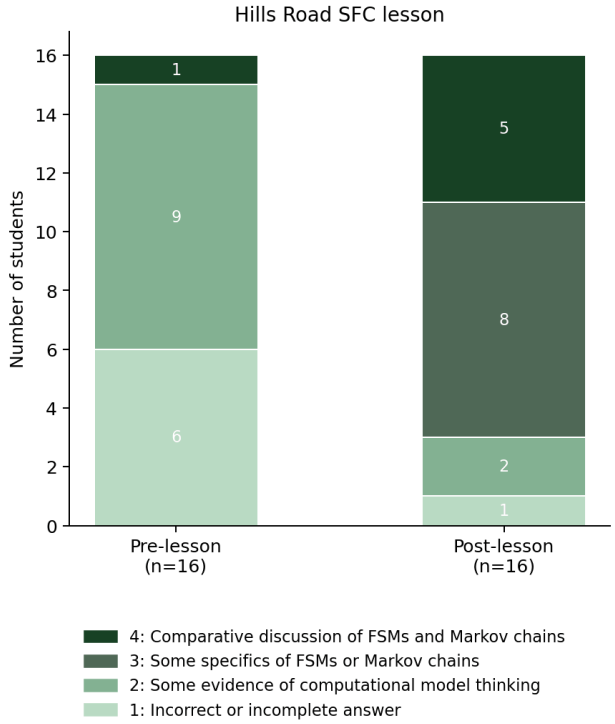


Figure 7 – Comparison of pre-lesson and post-lesson open-ended question answers at Hills Road

similarities, suggesting that the side-by-side design was effective in supporting higher-order thinking skills (Forehand, 2007). More broadly, answers across the cohort shifted towards contrasting structure and probability, reflecting our core aim to emphasise the difference between rule-based and probabilistic inference models.

4.4. Blended year 12/13 CS class

The second lesson took place at All Saints' Catholic Voluntary Academy Sixth Form, Mansfield, with a group of six A Level computer science students, spanning year 12 and 13 (two withdrew before completion). Most had not taken GCSE computer science, so were finding A-Level content challenging. The school faces socioeconomic challenges, reflected in classroom behaviour issues and lower levels of prior computer science knowledge. None of the students were familiar with finite state machines. The lesson was cut slightly short due to technical delays, reducing the time students had to explore later datasets. Throughout the lesson, students were engaged and enjoyed working with a partner to progress through questions. The score breakdown by student can be seen in figure 8, where the data is more spread than at the first school. One student's reducing Likert scores were inconsistent with open questions, suggesting possible non-compliance, misunderstanding, or possibly inflated pre-lesson confidence.

Many students were able to discuss the uses of FSMs or MCs in relation to language and code parsing. One student, "Polarbear", increased from a pre-lesson score of 1 to 3. Their response evolved from discussing "structure" and logical differences between essays and code, to an explanation of probabilistic next word prediction in LLMs, and the risk of LLMs sounding articulate without factually understanding information. "Otter" was able to make a similar jump from discussing machine code to directly identifying the use-cases of FSMs and MCs in relation to the question. "Barracuda" jumped from a classification of 1 to 2, a small gain despite spending 94% of their time actively, answering questions. This could be due to spending over 75% of their time on the first dataset, suggesting that the shortened lesson length prevented deep exploration of the platform.

All pre-lesson responses fell into categories 1 or 2, with post-lesson responses falling between categories 2 or 3. The shift was smaller than at Hills Road, although most students did increase their categorisa-

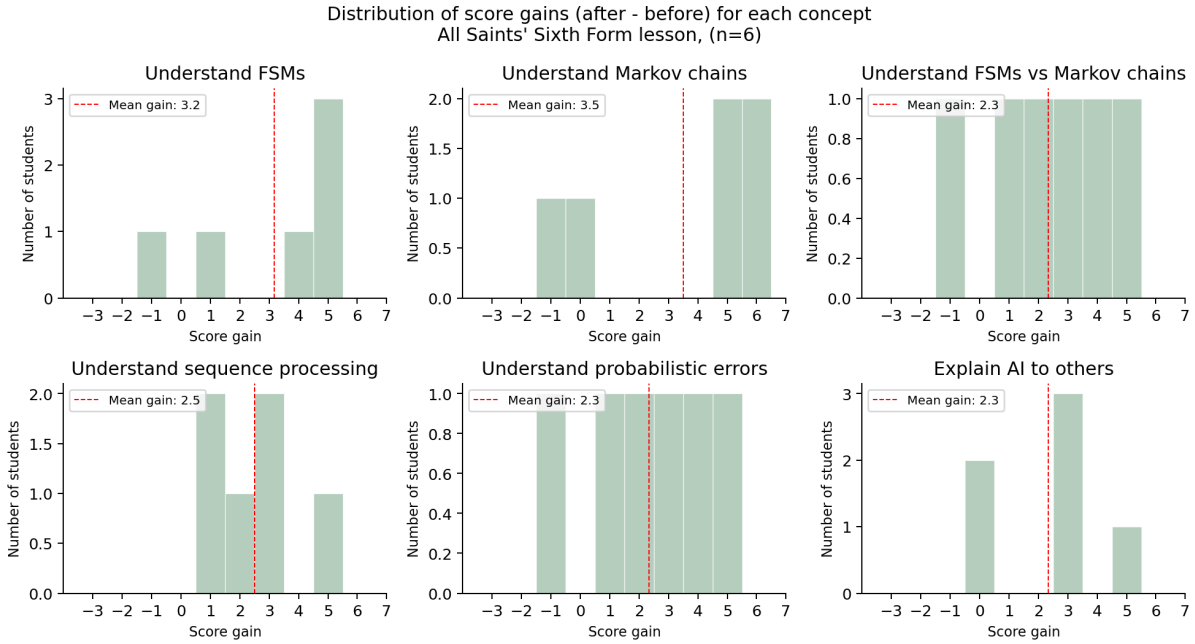


Figure 8 – Distribution of score gains per Likert scale question at All Saints'

tion score by at least 1, perhaps reflecting the reduced time available for self-directed exploration of statistically-sized datasets or a lower baseline of prior knowledge.

4.5. Comparative findings

The pattern of average score gains across concepts is consistent with Bloom's taxonomy, such that lower-order knowledge gains (understanding what FSMs and MCs are) saw the largest increase, whereas higher-order gains (transferring this knowledge to sequence processing more generally, and explaining the link to generative AI language models) saw more modest increases (Forehand, 2007). This is to be expected within a single 1 hour lesson, and suggests that extended use of the platform, or a follow-up lesson, could consolidate these higher-order skills.

The quantitative and qualitative results together support our design and learning goal of developing improved intuitive understanding of likelihood and validity. Across both cohorts, the pooled repeated-measures test found a highly significant overall effect on learning, and 22 students increased their long-form response categorisation scores. Even for students in post-test category 2, there was a shift towards references to sequence processing, rule-governed validation and probabilistic prediction. This supports our aim of critically informed LLM usage.

5. Conclusion

NodeToJoy is, to the best of our knowledge, the first educational platform contrasting rule-based and probabilistic models of computation using interactive visualisation. This comes at a time national curriculum plans call for greater emphasis on probabilistic literacy (Mathematics in Education and Industry (MEI), n.d.), (Royal Statistical Society, 2024). We have created visualisation pipelines to support hierarchically nested subgraphs. Our robust layout algorithms and pipeline can render up to 11,000 nodes. Most significantly, semantic zoom is applied as a novel mechanism for traversing levels of dataset granularity: navigating continuously from coarse, structurally-encapsulating nodes to fine-grained, character-level subgraphs. This allows non-trivial datasets to be abstracted and used approachably in educational contexts. We have introduced a hierarchical Markov chain viewer, exploring a gap that Harel identified in 1987 but did not pursue (Harel, 1987). In-school trials with our target audience of 16/17 year olds demonstrated significant learning gains, among cohorts with a wide range of socioeconomic and prior-knowledge backgrounds.

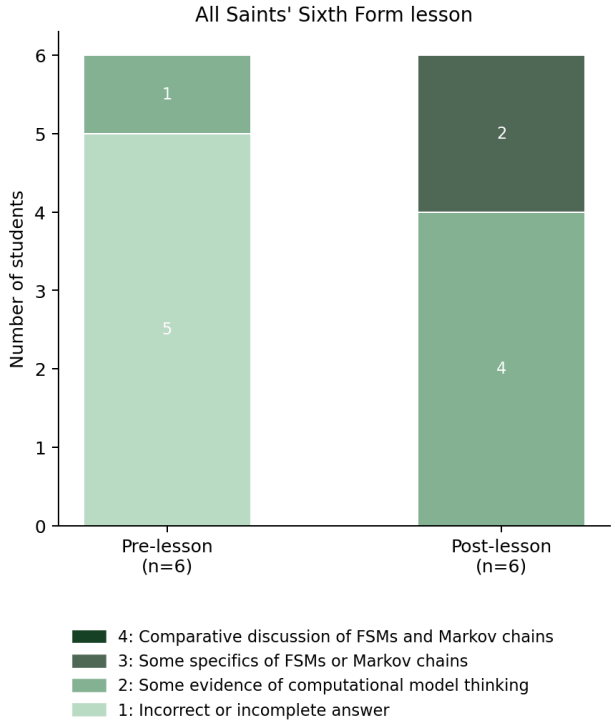


Figure 9 – Comparison of pre-lesson and post-lesson open-ended question answers at All Saints'

6. References

AQA. (n.d.). *A-level Computer Science 7517 Specification.* Retrieved 2026-04-15, from `https://www.aqa.org.uk/subjects/computer-science/a-level/computer-science-7517/specification/specification-at-a-glance`

Baudisch, P., & Rosenholtz, R. (2003). Halo: a technique for visualizing off-screen objects. In *Proceedings of the sigchi conference on human factors in computing systems* (p. 481–488). doi: 10.1145/642611.642695

Ben-Ari, M. (1998). Constructivism in computer science education. In *Proceedings of the twenty-ninth sigcse technical symposium on computer science education* (p. 257–261). doi: 10.1145/273133.274308

Castro, F. E. V. (2022). Exploring the use of finite-state machines and game creation to teach computational thinking in middle schools. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 2* (p. 618). doi: 10.1145/3502717.3532137

Cockburn, A., Karlson, A., & Bederson, B. B. (2009, January). A review of overview+detail, zooming, and focus+context interfaces. *ACM Computing Surveys*, *41*(1), 1-31. doi: 10.1145/1456650.1456652

Dengel, A. (2018). Seeking the treasures of theoretical computer science education: Towards educational virtual reality for the visualization of finite state machines. In *2018 ieee international conference on teaching, assessment, and learning for engineering (tale)* (p. 1107-1112). doi: 10.1109/TALE.2018.8615288

Elliott, S., Hudson, B., & O'Reilly, D. (2000, 4). Visualisation and the influence of technology in 'A' level mathematics: A classroom investigation. *Research in Mathematics Education*, *2*(1), 151-168. doi: 10.1080/14794800008520074

Forehand, M. (2007). Bloom's taxonomy. In *Emerging perspectives on learning, teaching, and technology.* Retrieved from `https://resources.finalsite.net/images/v1741805303/d41org/icn48k4luekuivu9f4vm/bloomstaxonomy.pdf`

Furnas, G. W., & Bederson, B. B. (1995). Space-scale diagrams: understanding multiscale interfaces. In

Proceedings of the sigchi conference on human factors in computing systems (p. 234–241). doi: 10.1145/223904.223934

Gates, P. (2018). The importance of diagrams, graphics and other visual representations in STEM teaching. In R. Jorgensen & K. Larkin (Eds.), *STEM Education in the Junior Secondary* (pp. 169–196). Singapore: Springer.

Harel, D. (1987, 6). Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, *8*(3), 231-274. doi: 10.1016/0167-6423(87)90035-9

Iudean, B., & Vescan, A. (2025). Narrative-driven learning: Teaching finite state machines through storytelling. In *Proceedings of the 17th international conference on computer supported education - volume 2: Csedu* (p. 627-638). doi: 10.5220/0013471600003932

Keller, R., Eckert, C. M., & Clarkson, P. J. (2006, March). Matrices or node-link diagrams: which visual representation is better for visualising connectivity models? *Information Visualization*, *5*(1), 62–76. doi: 10.1057/palgrave.ivs.9500116

Kim, M., & Jin, Q. (2022). Studies on visualisation in science classrooms: a systematic literature review. *International Journal of Science Education*, *44*(17), 2613-2631. doi: 10.1080/09500693.2022.2140020

Mathematics in Education and Industry (MEI). (n.d.). *Developing a Level 3 Qualification in data science and AI.* Retrieved 2026-04-15, from `https://mei.org.uk/insight/developing-a-level-3-qualification-in-data-science-and-ai/`

Melero, J., Hernández-Leo, D., & Blat, J. (2012). A review of constructivist learning methods with supporting tooling in ICT higher education: Defining different types of scaffolding. *Journal of Universal Computer Science*, *18*(16), 2334–2360.

Mutua, B. M., & Blackwell, A. F. (2023). Interactive narrative visualization for learning Markov decision process. In *Proceedings of the 34th annual workshop of the psychology of programming interest group* (p. 96-106).

Nimon, K., Zigarmi, D., & Allen, J. (2011). Measures of program effectiveness based on retrospective pretest data: Are all created equal? *American Journal of Evaluation*, *32*(1), 8-28. doi: 10.1177/1098214010378354

Pfannkuch, M., & Budgett, S. (2016). Markov processes: Exploring the use of dynamic visualizations to enhance student understanding. *Journal of Statistics Education*, *24*(2), 63-73. doi: 10.1080/10691898.2016.1207404

Piantadosi, S. T. (2014, 10). Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, *21*(5), 1112. doi: 10.3758/S13423-014-0585-6

Presmeg, N. (2006). Research on visualization in learning and teaching mathematics. In *Handbook of Research on the Psychology of Mathematics Education* (p. 205-235). Leiden, The Netherlands: Brill.

Rodger, S. (2006, 6). Learning automata and formal languages interactively with JFLAP. *ACM SIGCSE Bulletin*, *38*(3), 360-360. doi: 10.1145/1140123.1140270

Royal Statistical Society. (2024). *Key recommendations for the statistics curriculum in the UK.* Retrieved 2026-04-15, from `https://rss.org.uk/RSS/media/File-library/Policy/2024/Recommendations-for-statistics-curriculum-in-the-UK-full-paper-final.pdf?ext=.pdf`

Sweller, J. (2016). Cognitive load theory and computer science education. In *Proceedings of the 47th acm technical symposium on computing science education* (p. 1). doi: 10.1145/2839509.2844549

Touya, G., Gruget, M., & Muehlenhaus, I. (2023, 2). Where am I now? modelling disorientation in pan-scalar maps. *ISPRS International Journal of Geo-Information*, *12*(2). doi: 10.3390/IJGI12020062

Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., & Carrasco, R. C. (2005, July). Probabilistic finite-state machines-Part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(7), 1013–1025. doi: 10.1109/TPAMI.2005.147

Wang, C. H., Han, D., & Chou, W. K. (2018). Overview of information visualisation in science

education. *International Journal of Computational Science and Engineering*, *17*(1), 118-125. doi: 10.1504/IJCSE.2018.094423

Conceptualizing Agentic Programming with Scratch: Enhancing Computational Thinking on a Higher Level of Abstraction

Orit Hazzan
Education in Science and Technology
Technion, Israel Institution of Technology
Haifa, Israel
oritha@technion.ac.il

Noa Ragonis
Computer Science Studies
Beit Berl College
Beit Berl, Israel
noarag@beitberl.ac.il

Abstract

Agentic programming is becoming increasingly central to contemporary software and AI development. Rather than viewing programs as scripted procedures, agentic programming frames them as semi-autonomous systems that perceive their environment, pursue goals, make decisions, and adapt over time.

This paper proposes a pedagogical lens that uses Scratch, a block-based visual programming environment, to make agentic programming ideas more concrete and teachable. We argue that Scratch offers an intuitive space in which learners can conceptually model agentic behavior without relying on advanced AI infrastructure. Sprites function as agents, variables hold internal states, sensing blocks provide perception, broadcasts enable communication, and control structures create sense–think–act cycles that resemble agentic reasoning. Thus, Scratch becomes a shared language through which computer science and software engineering educators and practitioners can discuss and explore agentic systems.

We illustrate how familiar Scratch activities can be reframed to highlight agency, autonomy, decision-making, and interaction, thereby strengthening computational thinking on a higher level of abstraction. This reframing enables students to view programs as purposeful digital collaborators rather than passive executors of instructions, building mental models that connect traditional coding practices with emerging agentic approaches. In a future work, we plan to develop AgentScratch, which will include a Cognition Palette whose basic building blocks simulate key processes of agentic software development (Hazzan and Ragonis, 2026).

1. Introduction

Traditional programming is based on explicitly writing algorithms that execute predefined instruction sequences. In this paradigm, the programmer is responsible for specifying the logic, rules, and decision processes, and the computer simply follows these.

Over the last few decades, a shift has begun with the growth of machine learning approaches. A further conceptual step is emerging with the current development of experiences with Generative AI (GenAI) environments, particularly with agentic programming. Here, AI systems do not merely assist with writing code but operate as semi-autonomous agents that can plan, reason, select tools, generate and revise code, test their own outputs, and iteratively work toward goals. In this sense, agentic systems behave more like collaborators who analyze a problem, consider possible solutions, and take action rather than simply executing instructions.

The shift from viewing programs as linear, predetermined procedures to seeing them as autonomous, goal-directed entities introduces a level of conceptual abstraction and presents challenges for learners (Jin et al., 2024; Gonzalez et al., 2025). Learners must reason about perception, internal state, planning, coordination, and adaptation – capabilities that signal the emergence of a new, up-to-date form of computational thinking (CT): not only sequencing, loops, and conditionals, as well as problem decomposition, abstraction, and generalization, but also thinking in terms of goals, interactions, dependencies, feedback, and emergence, which represent higher levels of abstraction.

We suggest that when agentic ideas are grounded in familiar, playful contexts, such as guiding a sprite through a maze, coordinating sprites that communicate, or designing a simulation that reacts to player decisions, as done in Scratch programming, these invisible ideas become visible. Students can experience what it means for a program to “think,” “decide,” “reflect” or “respond,” rather than merely

executing instructions. Through such experiences, learners develop the contemporary CT required to conceptualize the foundational elements of agentic programming.

Although full agentic programming cannot be implemented within Scratch, we propose using Scratch as a conceptual bridge (Ejersbo et al., 2014). By engaging learners in the development of Scratch projects, educators can deliberately foreground ideas, such as autonomy, goals, perception, communication and adaptation. In this way, learners begin to encounter the core concepts of agentic programming in an accessible, manipulable environment that scaffolds their understanding of abstract notions. In a follow up work, we propose a Cognition Palette whose basic building blocks simulate key processes of agentic software development, providing the conceptual foundation for what we envision as AgentScratch, a block-based development environment for agentic programming (Hazzan and Ragonis, 2026).

2. Agentic Programming: Concepts and Challenges

Agentic programming represents a paradigm shift in contemporary software development, in which programs are constructed around autonomous or semi-autonomous agents rather than solely around predefined algorithmic procedures. In this approach, LLM-driven agents do more than generate code: they interpret high-level goals, decompose tasks, plan multi-step actions, interact with external tools (such as search engines or debuggers), incorporate feedback, and iteratively refine their outputs (Yao et al., 2022; Schick et a., 2023; Wang et al., 2025; Chowa et al., 2026). Consequently, programming becomes an orchestrated cycle of reasoning and action, in which decision-making and adaptation are integral to the computational process. Recent ACM and HCI literature characterize this shift as a move away from direct instruction toward collaborative human–AI problem-solving, where control is distributed between the programmer and the agent (Hoda, 2025; Schömbs et al., 2025; Wadinambiarachchi et al., 2025). Rather than writing complete algorithms, developers increasingly focus on specifying intentions, constraints, and evaluation criteria, while agents accept responsibility for generating, adapting, and evaluating code artifacts throughout the development process (Jin et al., 2024, Shah et al., 2025).

From a computer science *education* perspective, agentic programming introduces substantial challenges because it fundamentally reshapes what it means to "write a program." When learners interact with agent-like entities, programming activity shifts from constructing explicit algorithms to formulating goals, interpreting feedback, and supervising autonomous behavior (Jin et al., 2024, Gonzalez et al., 2025). Empirical studies indicate that students may produce functional solutions with the assistance of agents while holding only a partial understanding of the underlying logic and control structures (Shah et al., 2025; Wermelinger, 2023; Kazemitabaar et al., 2023). This creates instructional challenges not primarily around misuse but around how learners conceptualize core agentic ideas, such as autonomy, decision-making, responsibility, and control. Therefore, educators are required to rethink learning objectives and assessment practices to distinguish between what the human learner understands and performs independently and what is delegated to the agent. More broadly, teaching with agentic systems demands new pedagogical approaches that support productive human–agent collaboration, including helping students learn how to specify intentions, evaluate agent behavior, verify outcomes, and reflect on the division of labor between themselves and the agent (Jin et al., 2024, Vaithilingam et al., 2022). These challenges highlight the need for educational models that make the underlying concepts of agentic programming explicit and cognitively accessible.

Agentic programming also introduces multiple layers of abstraction simultaneously, including goal specification, autonomous decision-making, hidden reasoning processes and tool-mediated execution. Unlike traditional programming, where causality can often be traced directly through explicit code paths, the actual execution of programming in agentic systems is distributed across interactions between agent reasoning, prompts, external tools, and iterative feedback loops (Yao et al., 2022; Schick et al., 2023). As a result, learners may struggle to understand why a system behaves as it does because outcomes often emerge from complex, partially opaque processes rather than from transparent, line-by-line logic. ACM and CHI studies report that this opacity of causality presents a significant cognitive challenge, particularly for novice programmers, as essential reasoning and decision-making processes occur behind the scenes, reducing traceability and explainability (Jin et al., 2024, Vaithilingam et al.,

2022). Broader reviews of LLM-based agents similarly emphasize that educational applications must address issues of transparency, overreliance, and effective integration into learning environments, precisely because agents' intermediate reasoning and evidentiary bases are not always visible to learners (Wang et al., 2025; Chowa et al., 2026). Together, these challenges complicate learners' ability to form accurate mental models of agentic behavior, underscoring the need for pedagogical approaches that externalize agent reasoning and make core agentic concepts more accessible to learners.

From the CT perspective, agentic programming foregrounds an expanded and up-to-date set of practices that extend well beyond traditionally agreed-upon skills. In agentic contexts, learners are required to engage more deeply in goal specification, decompose into subgoals, plan and monitor multi-step processes, evaluate generated solutions, iteratively refine outcomes, and coordinate actions, often in collaboration with autonomous agents (Jin et al., 2024, Schömbs et al., 2025). These competencies reflect the growing prominence of human–agent collaboration in real-world programming practices (Hoda, 2025; Wadinambiarachchi et al., 2025). Together, these developments signal the need to reconceptualize CT in ways that make agency, autonomy, and intentional behavior explicit and teachable in the classroom. In this paper, we discuss CT dimensions of agentic programming that can be introduced and scaffolded through accessible programming environments, such as Scratch.

3. Scratch as a Learning Environment for Simulating Contemporary Agentic Programming

Scratch is a block-based visual programming environment developed by the MIT Media Lab with the explicit goal of enabling learners to create interactive stories, games, and simulations while lowering the syntactic barriers to programming (Resnick et al., 2009; Maloney et al, 2010). Its design emphasizes snap-together blocks, immediate visual feedback, and an event-driven execution model, allowing learners to focus on computational ideas, behaviors, and interactions rather than syntax. Central to Scratch's philosophy is the notion of programming for all, where creative expression, experimentation, and sharing are core components of learning (Resnick et al., 2009).

A substantial body of research has demonstrated that Scratch supports the development of foundational CT concepts and practices, including decomposition, abstraction, and debugging, through the design and refinement of meaningful computational artifacts (Brennan & Resnick, 2012). Systematic reviews of Scratch-based learning research further indicate consistent positive effects on learners' CT development across K–9 contexts, with particular strengths in supporting iterative problem-solving and conceptual understanding (Hazzan et al., 2020; Zhang & Nouri, 2019; Talan, 2020; Ragonis et al., 2022A; Ragonis et al., 2022B; Stewart & Baek, 2023). While Scratch is most commonly associated with children aged 8–16 years, empirical studies show that its pedagogical value extends beyond young learners. In teacher education, Scratch has been shown to positively influence attitudes toward programming, self-efficacy, and conceptual views of computing (Yükseltürk, 2017).

In recent years, Scratch has increasingly been explored as an entry point for introducing ideas from AI, machine learning, and data-oriented reasoning (Kazemitabaar et al., 2023). Studies have described extensions that connect Scratch to external services, enabling learners to experiment with data streams, pattern recognition, and algorithmic decision-making (Wald et al., 2019).

Importantly, Scratch's core constructs – sprites with internal states, sensing blocks, message passing, and iterative control structures – naturally lend themselves to modelling aspects of agentic behavior. Although Scratch does not support full agentic programming in the sense of LLM-driven autonomous agents, its design enables learners to conceptualize sprites as agents – entities that perceive, decide, act, and interact with their environment. *This design positions Scratch as a promising pedagogical environment for introducing the essential ideas of agency, autonomy, and feedback that underlie agentic programming in a concrete and developmentally appropriate manner*.

There are several environments that share the pedagogical spirit of Scratch, including Snap! (which extends Scratch with more advanced programming constructs), MIT App Inventor (which emphasizes event-driven, real-world application development), Cognimates (which integrates Scratch-like blocks with capabilities such as speech recognition, image recognition, and robotics), and Tynker (which focuses on game-based learning). The approach presented in this paper is conceptually aligned with

these environments in its use of block-based programming as a medium for developing CT. However, Scratch is selected as the focal environment due to its widespread global adoption and its well-established role in supporting the learning of programming foundations across all age groups, including higher education.

4. Mapping Agentic Programming into Scratch

Following our pedagogical suggestion, in this section we describe the conceptual bridge between agentic programming and Scratch by mapping core agentic programming ideas onto concrete Scratch constructs. As discussed in the previous sections, while Scratch cannot support full LLM-driven agentic systems, it can be used as an environment for conceptualizing agentic behavior. Therefore, the mapping presented here is pedagogical rather than technical. The idea is to ground agentic programming concepts in familiar Scratch programming practices, enabling learners to reason about agency, autonomy, and interaction through hands-on design. As stated above, in parallel, we plan to develop AgentScratch to supports LLM-driven agentic behavior.

The proposed mapping is first presented by identifying core agentic concepts and their corresponding Scratch constructs and subsequently, by analyzing the accomplishment of agentic conceptions by their Scratch analogs.

4.1. Core Agentic Concepts and Their Scratch Analogues

Table 1 summarizes the conceptual mapping underling how Scratch can naturally support agentic ideas through its design primitives. This reflects how agentic programming commonly characterizes systems as composed of agents who pursue goals, perceive their environment, make decisions, act, communicate, and adapt over time.

Agentic Programming Concept	Scratch Representation
Goals	The target of the entire simulation
Agent	Sprite with its own scripts and local variables
Environment	Stage, stage scripts, other sprites, global variables
Perception	Sensing blocks (e.g., touching?, distance, key pressed?)
Internal state / memory	Sprite-specific variables and lists, clones
Decision making / planning	Control structures (if, if-else, repeat, forever), time
Action	Motion, looks, sound, pen
Communication	Scratch actions such as broadcast / when I receive / when touch
Prioritizing competing goals	Rule ordering, conditional precedence, variable-based priorities
Reflection / monitoring	Lists or variables used as logs or counters, inputs from the user gottten by the operation ‘ask’ and the variable ‘answer’
Tool use	Custom blocks (procedures/functions)

Table 1 – Conceptual mapping between agentic programming concepts and Scratch representation

This mapping makes it explicit that Scratch embodies many of the structural elements required to reason about agentic systems, even in the absence of AI capabilities. Hence, explicit instruction that addresses these analogies can promote a meaningful understanding of agentic programming and scaffold the abstract concepts that agentic programming introduces.

4.2. Accomplishing Agentic Conceptions and Their Scratch Analogues

This section describes the three main elements of agentic programming and their corresponding implementation in Scratch.

a) *Modeling Sense–Think–Act Cycles in Scratch:* A central idea in agentic programming is the continuous sense–think–act cycle. In Scratch, this cycle can be explicitly modeled using forever loops that repeatedly:
   - sense the environment (using sensing blocks);
   - evaluate internal state and conditions;
   - select and execute actions.

By designing sprites around such sense-think-act loops, learners shift from writing linear scripts to designing autonomous behavior. ***Such loops do not precompute the sequence of behaviors***; rather, behaviors emerge ***dynamically from*** interactions among the system state or the environment, perception, and control logic. This reflects a fundamental property of agentic systems and supports learners' understanding of autonomy and adaptation.

b) *Single-Agent and Multi-Agent Designs:* Scratch supports both single-agent and multi-agent perspectives. In single-agent designs, a sprite autonomously navigates an environment (e.g., a maze or simulation). In multi-agent designs, multiple sprites operate concurrently, each with its own goals and decision logic, while coordinating through message passing, or global variables values. The multi-agent Scratch scripts provide a powerful analogy to agentic programming, since learners can reason about:

- parallel execution;
- coordination and negotiation;
- shared versus local state;
- emergent system's behavior.

These properties of Scratch foreground interaction and dependency – key concepts in contemporary agentic systems, as they underpin the autonomy of such systems.

c) *Reflection, Explainability, and Metacognition:* As agentic-oriented designs emphasize reflection on decision making, Scratch learners can be encouraged to log decisions, record reasons for actions, or visualize internal state changes. Such practices externalize otherwise invisible processes and support explainability, metacognition, and debugging. Here are several examples of such externalization activities:

- display state changes visually (e.g., speech bubble, costume change, color effect),
- externalizes decision making,
- create a list of variables and present their changing values,
- observe project behavior, change one rule, re-run, and compare decision logs.

Pedagogically, this aligns with the need to make agent reasoning visible, which was identified earlier as a major abstraction barrier in learning agentic programming (Rombaut et al., 2024).

5. Illustarting the Mapping with a Concrete Scratch Simulation

In this section, we describe a simulation that models the motion vectors of two billiard balls before, during, and after a collision, until both balls come to a complete stop, while accounting for friction and table boundaries. This demonstration is inspired by a project developed by a student during his training to become a physics teacher, as part of a course on CT. The project simulated a billiards game as a computational process of collisions between bodies in physics, taking into account the objects involved, the forces acting on them, and the angle of impact.

Figure 1 presents a screenshot of the simulation. Figure 2 shows part of the Scratch blocks that set up the screen and simulate the forever loop that governs the simulation. Table 2 outlines the representation of agentic programming concepts in the simulation. As can be seen, even a relatively simple simulation can be described using agentic terminology.

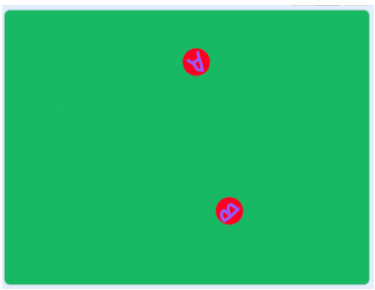


Figure 1 - A simulation of two billiard balls during a collision

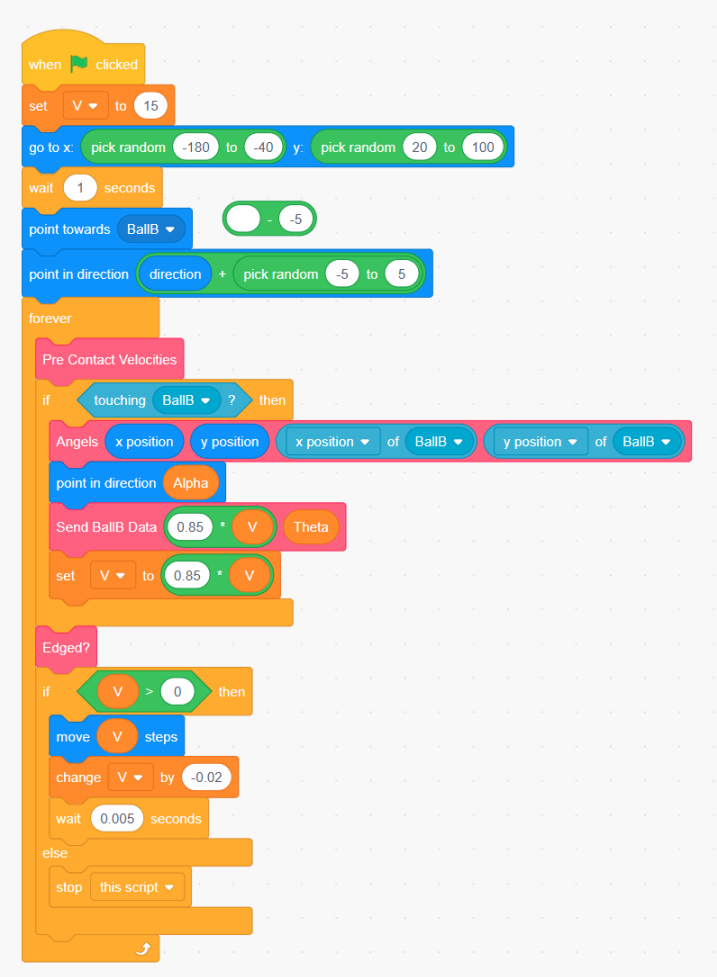


Figure 2 - Part of the Scratch blocks that creates the simulation of the two balls[1]

6. Computational Thinking Shifts and Pedagogical Implications

To examine the shift in CT skills required for agentic programming, we refer to the analytical components of agentic programming in Scratch, as presented in Section 4. Table 3 summarizes these analytical components and lists the fundamental CT and the agentic CT skills relevant to their implementation.

By foundational CT skills we refer to more traditional algorithm-centered CT skills that are now frequently established in programming curricula. They are typically emphasized in introductory programming and early CT frameworks, where the focus is placed on single-program, rule-based, and locally traceable logic.

By agentic CT skills, we refer to CT skills involved in designing, reasoning about, and coordinating autonomous agents and their interactions with environments and other agents. These agentic CT skills emphasize intentionality, interaction, monitoring, coordination, and reflection. They encompass CT skills that are often underrepresented or implicit in traditional curricula but become salient and necessary in agentic contexts.

We note that the distinction presented in Table 3 does not represent a hierarchy of difficulty, but rather a shift in CT orientation prompted by agentic programming.

[1] Created by Ali Zur (with his permission).

Agentic Programming Concept	Representation in the Simulation of the Collision
Goals	The simulation models the motion vectors of two billiard balls, before, during, and after collision while accounting for friction and table boundaries until both balls come to a complete stop.
Agent	The simulation include two objects: ball A is in motion, with a nonzero velocity, whereas ball B is stationary prior to the collision.
Environment	The environments includes: A felt-covered billiard table; ball A with initial velocity and direction prior to the collision and ball B is stationary beforehand; a variety of variables
Perception	Ball A accounts for friction with the felt-covered table, detects and handles collisions with ball B, and manages interactions with the table boundaries. The code uses *touching?* and variables values.
Internal state / memory	Ball A has 11 variables, for example: gV2, gTheta, and Right Attack. Ball B has four variables, for example: gV2 and gTheta.
Decision making / planning	Ball A includes a *loop* that runs while its velocity remains *greater than zero* and controls the simulation of the two ball's motion. This is a *forever* loop useing *if* and *if-else* structures.
Action	The two balls move on the table according to the scripts.
Communication	Ball B responds to the collision message from A by receiving its updated angle and velocity. The comunication use *broadcast.*
Prioritizing competing goals	Figure 1 presents a part of the rule ordering. The simulation implements several basic behaviors but does not account for others, such as the spin of the ball, which affects both the friction with the felt and the motion vectors after the collision.
Reflection / monitoring	This simulation does not include inputs; inputs could be added, for example, for friction.
Tool use	The simulation includes a variety of custome blocks such: *Edged?*, *Send Ball B Data(V,Theta)*

Table 2 – Representation of agentic programming concepts in the simulation of the collision.

Agentic Programming Conceptions	Computational Thinking Skills	
	Foundational Computational Thinking Skills	**Agentic Computational Thinking Skills**
Core Agentic Concepts	Skills needed to identify agents, environments, goals, and states, and to abstract these concpets into programmable representations	
Modeling Sense–Think–Act Cycles	Skills needed to design continuous control scripts that integrate perception, decision logic, and action, highlighting adaptive and dynamic computation	
	• Algorithmic thinking beyond linear sequencing • Conditional reasoning and decision-making	• Iterative process design • Monitoring system state and feedback • Temporal reasoning (reasoning over time and repeated cycles)
Single-Agent and Multi-Agent Designs	Skills for developing system-level thinking, requiring learners to reason about distributed control, inter-agent dependencies, and collective outcomes	
	• Decomposition into interacting components • Coordination and communication	• Parallelism and concurrency • Reasoning about interactions and emergent behavior • Responsibility allocation and delegation
Reflection, Explainability, and Metacognition	Skills for making computational processes explicit, supporting learners' ability to explain, justify, and revise agent behavior.	
	• Evaluation and debugging • Iterative refinement and improvement	• Tracing and explaining computational behavior • Reflective reasoning about design choices • Justification of decisions • Awareness of limitations and uncertainty

Table 3 – Computational thinking skills: From traditional to agentic programming

The analysis presented in Table 3 does not imply that Scratch itself is an agent-native environment or that its existing computational vocabulary fully captures the concepts required for agentic programming. More generally, we propose that a programming environment designed for a different programming paradigm may not fully represent the concepts and computational processes underlying agentic programming. Therefore, a potential compatibility gap may exist between Scratch's current

programming abstractions and the conceptual requirements of agentic programming. This concern motivates our proposal for AgentScratch, an agent-oriented, block-based programming environment inspired by Scratch. Rather than simply repurposing Scratch for agentic programming, AgentScratch is intended to retain the accessibility and hands-on nature of block-based programming while introducing constructs that explicitly represent agentic concepts and processes. Specifically, AgentScratch introduces learners to LLM-based agents while constraining their design and behavior through a predefined set of visual, conceptual blocks. In this way, learners can explore agentic concepts without being overwhelmed by the technical complexity of directly programming LLM-based agents (Hazzan and Ragonis, 2026).

7. Conclusion and Summary

This study presents a pedagogical mapping that identifies structural and behavioral analogies between agentic programming and Scratch, enabling learners to experience agentic thinking within a familiar block-based programming environment. Rather than attempting to replicate agentic programming architectures, the proposed approach focuses on concretizing abstract agentic concepts, such as autonomy, goals, decision-making, interaction, and adaptation, through accessible Scratch design practices.

As mentioned above, though Scratch provides a powerful medium for conceptually modeling agentic concepts, it cannot be considered an agentic programming environment in the technical sense. Agentic programming involves systems in which agents can autonomously interpret goals, plan multi-step actions, dynamically select and use external tools, and iteratively refine their behavior based on feedback, often supported by large-scale models and access to external resources. In contrast, Scratch operates within a closed, deterministic, and predefined execution model: all behaviors must be explicitly programmed in advance, decision-making is limited to rule-based logic, and there is no intrinsic support for autonomous goal interpretation, dynamic tool invocation, or learning from data (unless such capabilities are explicitly defined and programmed). Hence, while Scratch enables the representation of core agentic programming concepts as presented in this paper, it is limited to scripted simulations of agency rather than authentic autonomous processes.

Yet, from a pedagogical perspective, we view Scratch as a pedagogical foundation whose familiar block-based interaction can be extended with constructs explicitly designed to represent agentic processes. Our vision of AgentScratch intend to address the compatibility gap, identified in Section 6, while retaining the accessibility and hands-on nature of block-based programming, it introduces building blocks that make agent-specific concepts (such as goals, states, perception, planning, interaction, and autonomous action) explicit and manipulable. In this way, the proposed environment does not merely repurpose a tool designed for another programming paradigm; it adapts the pedagogical affordances of Scratch to the conceptual requirements of agentic programming, providing learners with a direct environment in which to construct, observe, and reason about agentic computational processes.

The rapid emergence of GenAI is fundamentally reshaping the programming landscape, including the growing prominence of agentic systems. Although the future trajectory of these technologies remains uncertain, it is already clear that the nature of CT required in this context is changing significantly. Programming increasingly demands reasoning about intentional behavior, coordination, evaluation, and responsibility, in addition to traditional algorithmic constructs (Wing in Hoffmann, 2026).

The analysis in this paper highlights that responding to these challenges does not require abandoning well-established educational tools or prematurely introducing opaque technologies. Instead, it demonstrates how a dynamic and widely used environment, such as Scratch, can function as a powerful pedagogical mediator. By grounding potentially intimidating and abstract ideas in concrete, comprehensible programming experiences, the proposed approach supports the development of up-to-date CT skills while preserving learners' sense of agency, understanding, and control over their learning.

However, it is important to clarify the limitations of this mapping. Scratch cannot natively host AI agents, access large-scale models, or perform autonomous tool selections. Therefore, its role is not to implement agentic programming but to serve as a conceptual preparation environment and, where relevant, as an execution or visualization layer for externally generated agentic designs. This distinction

reinforces the central claim of this study: Scratch provides a *didactic medium* through which the essential ideas of agentic programming can be introduced, explored, and understood. Thus, the work presented here represents a pedagogical step toward supporting learners and educators in navigating the evolving realities of programming in the age of GenAI. In Ragonis et al. (2025) one such pedagogy is presented. Extending these insights, we envision AgentScratch, a block-based programming environment designed to support learners in developing agentic thinking and programming skills by making key agentic concepts and computational processes explicit and manipulable (Hazzan and Ragonis, 2026).

References

Brennan, K., & Resnick, M. (2012). Using artifact-based interviews to study the development of computational thinking in interactive media design. *Proceedings of the American Educational Research Association (AERA).*

Chowa, S. S., Alvi, R., Rahman, S. S., Rahman, M. A., Raiaan, M. A. K., Islam, M. R., ... & Azam, S. (2026). From language to action: a review of large language models as autonomous agents and tool users. *Artificial Intelligence Review*.

Ejersbo, L. R., Leron, U., & Arcavi, A. (2014). Bridging intuitive and analytical thinking: Four looks at the 2-glass puzzle. *For the Learning of Mathematics*, *34*(3), 2-7.

Gonzalez, E., Chan, J., & Weintrop, D. (2025). Quack! Configuring large language models to serve as rubber duck coding assistants. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE TS 2025)*. Association for Computing Machinery. https://doi.org/10.1145/3641555.3705235

Hazzan, O., Ragonis, N., & Lapidot, T. (2020). Computational thinking. In *Guide to Teaching Computer Science: An Activity-Based Approach* (pp. 57-74). Cham: Springer International Publishing.

Hazzan, O. & Ragonis, N. (2026). The Cognition Palette: Teaching Programming in the Age of AI Agents Using Block-Based Environments, *BLOG@CACM, Communications of the ACM*. https://cacm.acm.org/category/artificial-intelligence-machine-learning/

Hoda, R. (2025). Toward Agentic Software Engineering Beyond Code: Framing Vision, Values, and Vocabulary. *arXiv preprint arXiv:2510.19692*.

Hoffmann, L. (2026). The rise of computational thinking in the age of AI, Opinion - Communications of the ACM, January 2026.

Jin, H., Lee, S., Shin, H., & Kim, J. (2024, May). Teach ai how to code: Using large language models as teachable agents for programming education. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (pp. 1-28).

Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., & Grossman, T. (2023, April). Studying the effect of AI code generators on supporting novice learners in introductory programming. In *Proceedings of the 2023 CHI conference on human factors in computing systems* (pp. 1-23).

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on Computing Education*, *10*(4), Article 16.

Ragonis, N., Bukai, A., & Hazzan, O. (2022A). Selecting examples for CS courses: the case of a computational thinking MOOC. *ACM Inroads*, *13*(3), 22-28.

Ragonis, N., & Hazzan, O. (2022B, December). A MOOC on Computational Thinking for All: Pedagogical Principles, Challenges, and Their Application. In *2022 International Conference on Computational Science and Computational Intelligence (CSCI)* (pp. 1943-1949). IEEE.

Ragonis, N., Rosenberg-Kima, R. B., & Hazzan, O. (2025). A computational thinking course for all preservice K-12 teachers: implementing the four pedagogies for developing computational

thinking (4P4CT) framework. *Educational technology research and development*, *73*(1), 301-329.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60–67.

Rombaut, B., Masoumzadeh, S., Vasilevski, K., Lin, D., & Hassan, A. E. (2024). *Watson: A Cognitive Observability Framework for the Reasoning of Foundation Model-Powered Agents.* https://arxiv.org/abs/2411.03455

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., ... & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, *36*, 68539-68551.

Schömbs, S., Zhang, Y., Goncalves, J., & Johal, W. (2025). From conversation to orchestration: HCI challenges and opportunities in interactive multi-agentic systems. In *Proceedings of the 13th International Conference on Human-Agent Interaction (HAI '25)* (pp. 1–11). Association for Computing Machinery.

Shah, A., Chernova, A., Tomson, E., Porter, L., Griswold, W. G., & Soosai Raj, A. G. (2025, February). Students' Use of GitHub Copilot for Working with Large Code Bases. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (pp. 1050-1056).

Stewart, W. H., & Baek, Y. (2023). Analyzing computational thinking studies in Scratch programming: A review of elementary education literature. *International Journal of Computer Science Education in Schools, 6*(1), 35–58.

Talan, T. (2020). Investigation of the studies on the use of Scratch software in education: A systematic review. *Journal of Education and Future, 18*, 109–134.

Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022, April). 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* (pp. 1-7).

Wald, I., Zuckerman, O., et al. (2019). Teaching children machine learning through gesture recognition in Scratch. *Proceedings of the Interaction Design and Children Conference.*

Wang, H., Gong, J., Zhang, H., Xu, J., & Wang, Z. (2025). *AI agentic programming: A survey of techniques, challenges, and opportunities*. University of Leeds. *arXiv preprint arXiv:2508.11126*

Wermelinger, M. (2023, March). Using github copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (pp. 172-178).

Wadinambiarachchi, S., Waycott, J., Rogers, Y., & Wadley, G. (2025). Imagining design workflows in agentic AI futures. In *Proceedings of the 37th Australian Conference on Human-Computer Interaction (OZCHI '25)* (pp. 1–17). Association for Computing Machinery.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., & Cao, Y. (2022, October). React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations.*

Yükseltürk, E. (2017). An investigation of the effects of programming with Scratch on the views of preservice information technology teachers. *British Journal of Educational Technology*, *48*(2), 537–550.

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K–9. *Computers & Education*, 141, 103607.

What if Offloading Is Not the Problem?

Ilya Zakharov
Human-AI Experience team
JetBrains Research
ilia.zaharov@jetbrains.com

Abstract

Debates about generative AI in software engineering often center on cognitive offloading and the possibility that developers will lose skills. This position paper argues that this focus is important but too narrow. Evidence of durable cognitive decline remains mixed, and offloading can involve task substitution, skill displacement, skill decay, or failures of skill formation. A more useful unit of analysis is the changing human–AI work system, including its metacognitive, social, motivational, and organizational conditions. Drawing on research on extended cognition, learning, teamwork, motivation, automation, and developer productivity, the paper examines how GenAI may reshape judgment, novice learning, shared understanding, communication, ownership, accountability, professional identity, status, and the distribution of productivity gains. It argues that the central question is not whether developers should avoid offloading, but which capabilities and forms of agency human–AI systems should preserve and cultivate, for whom, and under whose goals. Evaluations of AI-assisted work should therefore extend beyond task-level speed to learning, team knowledge, meaningful work, well-being, and the allocation of any time and effort that AI releases.

1. The Fear of Offloading

One of the dominant fears surrounding AI-assisted programming is that, with work delegated to AI, developers may gradually lose their professional skills. Such a process of deskilling is often seen as an outcome of cognitive offloading, the act of reducing the mental processing requirements of a task by using external tools instead (Risko & Gilbert, 2016). Cognitive offloading is both common and consequential. Search engines have changed what people remember, and today people are more likely to retain where information can be found than the information itself (Sparrow, Liu, & Wegner, 2011). Access to online explanations inflates estimates of one's own knowledge, creating the illusion of understanding (Fisher, Goddu, & Keil, 2015). In a survey of 319 knowledge workers, greater confidence in generative AI was associated with less self-reported critical thinking, while confidence in one's own ability was associated with more; participants described critical thinking as shifting toward verification, integration, and stewardship of AI output (H.-P. Lee et al., 2025).

The risks of cognitive offloading are especially vivid in software engineering. The danger here is not merely that a particular skill becomes rusty; it is that the pipeline by which people become capable of judgment is interrupted just as judgment becomes the skill the organization claims to need most. Today, there is a growing fear among industry practitioners that an experienced developer who reviews an expanding stream of machine-produced changes may gradually become a rubber stamp for work they no longer understand well enough to assess (a risk sometimes called "comprehension debt", Aragwal et al., 2026).

In this position paper, however, I want to claim that while cognitive offloading is a real risk and is likely already happening, the more important shift may be motivational: not only what developers can still do, but what they will still want to do (and why). When people discuss cognitive offloading, they often imply that the individual brain (and cognition) is the correct unit of analysis and that preserving the previous distribution of skills is the proper objective. I want to claim that, first, the relevant unit of analysis is often not the unaided developer but a changing human–AI team environment, and, second, that the central risk is not simply capability loss but a reconfiguration of motivation, ownership, accountability, and the social routes through which expertise is formed. I also want to raise the question of where the

resources released due to cognitive offloading might (and should) go. The potential benefits depend on the organizational allocation of the resources AI releases. Time saved can become learning, care, exploration, and better judgment, or merely a higher throughput target. The central question is therefore not "How do we prevent developers from offloading?" but rather: **"Which cognitive, motivational, social, and organizational capacities should a human–AI work system cultivate, for whom, and under whose goals?"**

2. Offloading Is an Outcome, Not a Diagnosis

The term *offloading* can include several distinct phenomena (ref). It can mean **task substitution:** a person no longer performs an activity, such as a software engineer recalling an API or writing a routine transformation. It can also mean **skill displacement:** practice in one skill decreases while another skill (prompting, evaluating, integrating, or coordinating) grows. Cognitive offloading can also lead to different outcomes. One is **skill decay:** a capability previously held by the person becomes less available through disuse. Another is **failure of skill formation:** a novice never develops the representations required for independent performance or evaluation. Other outcomes can include **miscalibration** (the person mistakes the joint system's competence for their own or cannot identify when assistance is unreliable) or **loss of agency** (the person retains nominal responsibility but no longer meaningfully selects goals, methods, or acceptance criteria).

Generative AI is already being used to offload large parts of knowledge work. One of the strongest studies to date surveyed 319 knowledge workers across 936 concrete AI-use examples: greater confidence in GenAI was associated with less reported critical thinking, while the work shifted toward checking, integrating, and supervising AI output (H.-P. Lee et al., 2025). A cluster-randomized classroom study (Lindo & Cutad, 2024) found that structured GenAI use increased reported offloading and that greater offloading correlated with worse two-week retention (yet the AI condition itself had no reliable direct effect on delayed retention). In another study, across three groups (GenAI-aided, search-engine-aided, and unaided), GenAI users reported lower ownership and had more trouble quoting their own essays (Kosmyna et al., 2025). Moreover, participants showed weaker and less distributed brain activation (measured as EEG connectivity) in the GenAI group than in search-engine or unaided groups. However, it should be noted that the main part of the experiment involving EEG recording included only 18 people, with the measures being task-specific proxies rather than a direct test of lasting ability, so the results should be treated as preliminary.

There is also a contradictory set of results. Two 2025 video-learning experiments (with 101 and 215 participants; Şimşek, Anders, Göth, Specht, & Huff, 2025) found no retention or transfer difference between GPT summaries and ordinary transcripts. A small randomized medical-student study likewise found higher immediate quiz performance with ChatGPT but no retention advantage one week later (Kalam et al., 2025). Some research has also shown benefits for the creativity of outputs generated with GenAI. The most directly positive results come from the study by B. C. Lee and Chung (2024). Across five experiments on everyday and innovation problems, people using GPT-3.5 generated ideas rated more creative than people working unaided or with conventional web search. The gain held across both low- and high-constraint problems, but was stronger for **incremental** novelty than for radical innovation; the authors attribute it partly to the model's ability to connect distant concepts into coherent proposals. However, results for creativity are as mixed as results for cognition. An experiment with 60 participants (Wadinambiarachchi, Kelly, Pareek, Zhou, & Velloso, 2024) using an AI image generator during visual ideation found a decrease in creative output: greater fixation on an initial example, with fewer, less varied, and less original ideas. A larger 2025 randomized priming study (387 students, Medeiros, Cropley, Marrone, & Reiter-Palmon, 2025) likewise found no broad divergent-thinking advantage from ChatGPT examples over human examples; low-creativity ChatGPT prompts actually reduced performance.

Overall, it seems too early to establish from the available data that ordinary AI use causes durable detrimental effects on human skills and learning. However, given the historical precedents, I would say that we can state that cognitive offloading is likely to be happening. Before the invention of the printing

press, European education treated memory as a deliberately trained intellectual skill. Medieval scholars used spatial mnemonics, repeated recitation, and highly structured manuscripts to internalize material for preaching, argument, and composition. Mary Carruthers describes memory as one of the central instruments of medieval intellectual life, rather than merely a passive store of information (Carruthers, 2008). Although the available evidence is circumstantial, research supports the idea that as inexpensive, standardized books became dependable external stores, elaborate mnemonic training gradually declined as it lost much of its practical necessity (Rachal, 1988). A much more recent example of cognitive offloading is known as the "Google effect": a shift from remembering *what* to remembering *where it can be found* (Sparrow et al., 2011). In one of the first studies on this topic, participants read and typed trivia statements. Those who expected the computer to erase the statements later recalled approximately 29–31% of them, whereas those who believed the statements had been saved recalled only 19–22%. Explicitly telling participants to remember the statements did not eliminate the effect. In another experiment, participants subsequently recalled the location where information had been stored better than the information itself: mean recall was 0.49 for location versus 0.23 for content (although the authors noted that location recall received a stronger cue).

Cognitive offloading itself is not new. Although some skills have already been lost, new skills have replaced them. The fear of cognitive offloading often comes hand-in-hand with a larger and weakly supported claim: that AI use is leading to the overall decline of human intelligence. However, evidence of declines in task-specific knowledge, effort, or even calibration does not imply that general intelligence is declining. Human intelligence is not a warehouse of isolated procedures. It includes learning relational structure, connecting representations, selecting goals, and adapting action to unfamiliar conditions. According to recent advances in biological theories of intelligence, it is based on a set of interconnected functional nodes that probably combine various skills and abilities into a complex network (Santarnecchi, Galli, Polizzotto, Rossi, & Rossi, 2014; Barbey, 2018; Zakharov, Tabueva, Adamovich, Kovas, & Malykh, 2020). While tools can atrophy particular routines, they can simultaneously enable much more complex activity. The normative question is not whether the pre-AI repertoire of skills remains intact in every person, but rather whether the emerging repertoire of new skills supports competent, autonomous, resilient, and meaningful action.

One of the hardest questions related to cognitive offloading due to GenAI may be how novices acquire new skills. Experts can often delegate an activity because prior experience lets them recognize implausible output and retrieve a deeper model when needed. Novices, however, risk receiving the product without the formative process. A within-subject study of graduate CS students working in a legacy code base found that Copilot improved task productivity but did not improve code-base comprehension (Qiao, Shihab, Haque, & Hundhausen, 2026). The authors also observed that faster performance was associated with poorer reverse-engineering comprehension. These results raise questions about student education. What can be learned through assisted performance? Which forms of struggle are diagnostically or developmentally productive, and which are merely friction? A design that is appropriate for an expert may be actively harmful to a beginner. However, it seems that such systems can be designed with AI in mind. A large randomized field experiment with nearly 1,000 high-school students (Bastani et al., 2025) demonstrated that access to a conventional GPT-4 interface improved performance on practice mathematics problems by 48%, but when AI was removed, these students scored 17% worse than students who had never used it. Crucially, they did not perceive that they had learned less. However, a tutor-style AI system that withheld complete answers and supplied progressive hints largely eliminated the negative effect (Bastani et al., 2025). This shows that, with a carefully designed approach to training novices (one that is yet to be firmly established) expertise can be developed.

There is at least one more important AI-related danger, which lies in the phenomenon of the illusion of explanatory depth. Overall, people believe they understand mechanisms (such as toilets, zippers, or helicopter flight) until they are asked to explain them step by step (Rozenblit & Keil, 2002). The illusion of understanding is also related to the perception of the availability of information. After searching online for explanations, people rated their own ability to explain unrelated topics more highly. They

appeared to partially mistake access to external knowledge for knowledge located in their own minds (Fisher et al., 2015). An important and real risk here is that easy access to AI tools can allow people to produce correct-looking work while overestimating how much they themselves understand.

So, we can claim that GenAI can accelerate cognitive offloading while trading task-specific skills for new coordination abilities (taking into account the risk of leaving novices without foundational knowledge and the risk of creating a dangerous illusion of personal understanding). With cautious optimism, we might also expect GenAI to contribute to an evolution of human intelligence rather than simply to its decline.

3. Extended Cognition and Metacognition

The image behind much offloading anxiety is a developer whose authentic cognition takes place inside the skull and whose tools subtract from that internal activity. This is historically and theoretically narrow. Vygotsky (1978) described higher mental functions as mediated by cultural tools and signs. Hutchins (1995) showed how cognition can be distributed across people, artifacts, and coordinated procedures. Clark and Chalmers (1998) argued that when external resources are reliably integrated into thought and action, they can be parts of a cognitive system rather than merely aids to an otherwise self-contained mind.

Programming has always made this point concrete. A developer thinks with notation, documentation, issue trackers, version history, debuggers, tests, code review, and colleagues (Ehn, 1990). These resources do not simply replace internal operations. They alter which internal operations are useful. Version control changes the value of remembering every prior state; a debugger changes how hypotheses can be tested; a high-level language makes some machine details less salient while making larger abstractions tractable. The resulting competence belongs partly to a coordinated arrangement.

Clark (2025) applies this perspective to generative AI. Human brains are unusually capable of adapting their own contributions to the opportunities created by external resources. The result of such adaptation can be a new, interwoven whole in which the person, interface, model, artifacts, and social setting continually change one another's roles. This view does not imply that every convenient tool is literally part of the mind, or that dependence is harmless, but it does change the evaluative target. Instead of measuring only what the unaided individual retains, we might consider the complex system that includes both intelligent tools and human decisions. For SE-related questions, evaluation might include **calibration** (whether people know when and why to trust an output), **contestability** (whether they can challenge, redirect, or override the AI output), **resilience** (whether work can continue when the model is wrong or unavailable), or even inter-individual phenomena, such as a team's **distribution of skills** (whether competence and control are spread across a team or concentrated in a vendor, model, or small group of experts).

Introducing new tools into human activity has been shown to alter human behavior in multiple areas. After superhuman AI entered Go, human players produced moves that were both of higher quality and more novel; the novelty was not reducible to copying the machine's distinctive moves (Shin, Kim, van Opheusden, & Griffiths, 2023). In a field experiment at Procter & Gamble, individuals working with AI performed comparably to conventional two-person teams and produced solutions that crossed established technical and commercial boundaries (Dell'Acqua et al., 2025). Generative AI can also increase the judged creativity of individual stories, especially for less creative writers, although it simultaneously makes the collection of stories more similar (Doshi & Hauser, 2024). In a different domain, personalized conversations with language models reduced belief in emerging conspiracy theories more than control interactions or static corrective text, with some effects persisting for one to two months (Costello, Rabb, Stagnaro, Pennycook, & Rand, 2026). The latter is a recent preprint outside software engineering, but it illustrates the theoretical possibility that AI can reinforce revisions to users' current knowledge.

There are, of course, objections to the idea that AI can meaningfully extend our cognition. Pritchard (2026) argues that while traditional technologies normally transform domain-specific skills, agentic AI

can take over activities through which executive intellectual abilities and character are exercised, leading to a decrease in skill formation without any corresponding change in human executive functions. This is an important hypothesis, but it should not be accepted as a definitional truth. The boundary between "domain" and "executive" cognition is neither fixed nor technologically determined. Planning, monitoring, and revision are learned in interaction with tools, and while AI can be configured to suppress these activities, it can also expose alternatives, solicit commitments before acting, or make uncertainty actionable. A good example of a system that enhances our judgment comes from Sarkar (2024), who proposes recasting AI from assistant to provocateur: a system that questions assumptions and offers counterarguments rather than obediently completing the task. In an experimental instantiation of this idea, brief AI-generated "provocations" that critiqued and proposed alternatives to AI suggestions during a shortlisting task induced critical and metacognitive thinking relative to plain AI assistance (Drosos, Sarkar, Xu, & Toronto, 2025).

However, the risk that cognitive offloading places too much burden on executive functions is real. As code becomes cheaper to produce, it can proliferate faster than it can be understood, shifting scarce effort toward choosing worthwhile problems, connecting technical decisions to user and business consequences, integrating changes into an evolving system, verifying behaviour, and preserving a shared account of why the system is as it is. This is often described as a move "from coding to judgment," but judgment is not a residual capacity that appears automatically when implementation is automated. It depends on learned discrimination, a working model of the system, calibrated uncertainty, comparison among alternatives, and responsibility for the outcome. If organizations expand output as generation becomes cheaper, developers gain no spare capacity for this work; they simply receive more artifacts to inspect. More of the burden therefore moves from object-level production to metacognitive control: monitoring AI output, calibrating confidence in both the model and one's own understanding, and deciding when scrutiny is sufficient (H.-P. Lee et al., 2025; Ackerman & Thompson, 2017). This may be frustrating rather than liberating. Continuous self-monitoring consumes limited cognitive resources (Safati, Seli, & Smilek, 2026), sustained monitoring of automated systems is effortful and stressful (Warm, Parasuraman, & Matthews, 2008), and mental effort is reliably associated with negative affect across tasks (David, Vassena, & Bijleveld, 2024). The result may be less cognitive relief than a new, less visible form of strain.

So, the risks of cognitive offloading, deskilling, novice learning, and new burdens on metacognition are real. I would argue, however, that the current preoccupation with GenAI's cognitive risks may divert attention from other, arguably more consequential ones.

4. GenAI Tools and Software Teams

One area of change that is overlooked in discussions of GenAI's effects on software engineering is how it can affect software teams. Margaret-Anne Storey (2026) has recently proposed additions to the well-known notion of technical debt. The term technical debt is used to describe the additional cost of maintaining a system that may arise when one chooses an expedient solution to accelerate its development. Technical debt is a property of the technical artifact, the system that is being built. The idea of cognitive debt extends future costs and deficits to a team's shared understanding of the system. Cognitive debt might accumulate alongside intent debt (the loss of goals, constraints, and rationale) that led to the creation of the technical artifact in the first place. It is important to notice that both cognitive and intent debt are, to a considerable extent, interpersonal. A codebase can compile and pass generated tests while no team member can explain its invariants or why a tradeoff was accepted, leading to potentially catastrophic problems when some part of the system breaks. Thus, retrieving information about architectural solutions is becoming one of the bottlenecks in contemporary software engineering. Rising communication barriers are already among the managerial concerns associated with technological change. Microsoft's natural-experiment study of 61,182 US employees (Yang et al., 2022) found that the firm-wide move to remote work produced more static and siloed collaboration networks, fewer bridges across structural groups, and a shift toward more asynchronous communication, potentially leading to

a decline in innovation within the company. At the same time, a recent systematic review of 39 peer-reviewed studies on LLM assistants and developer productivity found that communication and team activity remain largely underexplored, with little longitudinal or team-level evidence (Mohamed, Assi, & Guizani, 2026).

But the move from building with a colleague to building with GenAI can change more than information retrieval. Asking another developer a "small" question can transmit tacit context, reveal who knows what, establish permission to ask again, and make uncertainty visible to the team. Asking AI may be faster, less embarrassing, and available at any hour, and it can democratize access to explanation. But if many such exchanges disappear, the organization may lose the weak ties through which trust, mentorship, and situation awareness develop. In a study of 33 student teams, groups with an AI teammate showed lower human-to-human responsiveness and lower reported belonging and status than all-human groups; greater AI dominance was associated with participants feeling less valued (Nixon et al., 2026). The study warns that while a GenAI tool can increase the number of useful utterances or the speed of a solution, it can also decrease the social conditions under which people learn and remain willing to contribute. For software engineering, cognitive debt may therefore be partly a network property. If AI answers questions privately, the answer may never enter the team's shared memory. The immediate exchange is efficient, while the organization becomes less aware of itself.

5. Offloading and Motivation

Another weakness in the offloading debate is that it often treats developers as information processors whose main relationship to work is cognitive demand. Cat Hicks (2026) calls this the "brains-in-jars" view: developers are separated from the social, emotional, and organizational conditions in which software is actually made. Once the problem is framed this way, it is easy to overlook why people become developers and why they remain in the profession.

People work in software engineering for many reasons: the pleasure of making something, technical challenge, problem-solving, usefulness, autonomy, recognition, and participation in a community. Beecham, Baddoo, Hall, Robinson, and Sharp (2008) found recurring roles for problem-solving, meaningful or beneficial work, and interest in the work itself, while stressing that motivation varies across contexts. A more recent survey of 521 developers similarly found that enjoyment, ownership, learning, challenge, importance, community, and recognition all mattered, but that no single motive was sufficient (Amit & Feitelson, 2026). Self-determination theory provides a broader explanation: autonomy, competence, and relatedness support sustained, high-quality motivation and well-being at work (Ryan & Deci, 2000; Gagné et al., 2022). Developer-specific research also links agency, motivation, a learning culture, and belonging with self-reported productivity (C. M. Hicks, Lee, & Ramsey, 2024).

Church and Marasoiu (2025), in a deliberately provocative opinion paper, argue that software organizations should treat having a good time at work as a legitimate aim rather than translate every humane concern into the language of productivity. Enjoyment cannot settle questions of safety or responsibility. But a future of programming that is efficient while draining the work of curiosity, inquiry, and inspiration would still be a design failure.

Recent studies of developer preferences support this account. In a mixed-methods study of 860 developers, Choudhuri, Badea, et al. (2026) found strong use of and demand for AI in coding and testing, as well as demand for reducing toil in documentation and operations. Developers drew firmer boundaries around work tied to identity and relationships, such as mentoring. Their responsible-AI priorities also varied by task: reliability and security mattered most for system-facing work; transparency, alignment, and steerability mattered where control was central; and fairness and inclusion mattered for human-facing tasks. These were not general attitudes toward "AI," but judgments about particular forms of support in particular tasks. A second study of 448 professional developers sharpened this point. Task identity (the extent to which developers identified with and intrinsically enjoyed a task) predicted less willingness to give AI final decision-making authority. Accountability reduced willingness to let AI produce work on the developer's behalf, while high task demands made delegation of decisions more

acceptable (Choudhuri, Bird, Badea, Gerosa, & Sarma, 2026). Resistance to AI is therefore not simply resistance to change. It often reflects what developers find meaningful, what they feel responsible for, and which parts of the work they regard as their own.

AI changes developers' authorship and agency. Contribution, understanding, control, and responsibility were never perfectly aligned in software engineering, but GenAI can pull them further apart. A developer may define the goal without writing the implementation, understand the intended behavior without understanding the mechanism, approve a change without controlling how it was produced, and remain accountable when it fails. Elish's (2019) concept of the *moral crumple zone* describes situations in which a human absorbs blame for an automated system despite having little meaningful control. Recent interviews with experienced developers using software agents show that meaningful oversight involves more than final approval. It includes setting constraints, planning with the agent, monitoring its execution, and reviewing the result. Developers also reported difficulty assessing agent-generated code and sometimes relied on imperfect heuristics to decide whether it was acceptable (Dhanorkar, Passi, & Vorvoreanu, 2026). Human-in-the-loop design is therefore not automatically human-centered. If the human's practical role is only to approve output under time pressure, the loop may preserve liability without preserving agency. Together, these changes may reorganize not only how software is produced, but also which kinds of work carry identity, authority, and status.

6. Social Hierarchy and Offloading

Software engineering often presents itself as a meritocracy: skill can be demonstrated through working artifacts, difficult problems, and visible contributions. AI adoption can also affect the social structure of software engineering. On the one hand, AI can widen participation by lowering some barriers. In three randomized field experiments involving 4,867 developers, access to an AI coding assistant increased the number of completed tasks on average, with larger gains among less experienced developers (Cui et al., 2026). The finding concerns task completion rather than the quality of every resulting artifact, but it supports the possibility that AI can help some developers overcome practical barriers to production. At the same time, AI may produce a Matthew effect: people with stronger prior knowledge, better access, and greater organizational power may be better able to direct, verify, and integrate its output. A recent study of open-source projects found that GitHub Copilot increased participation and code contributions overall, but peripheral contributors gained less and faced larger increases in coordination work than core contributors (Song, Agarwal, & Wen, 2026, preprint). A similar pattern has been observed outside software engineering: AI-assisted translation increased content production across Wikipedia language communities, but the largest gains went to communities that already had the strongest resources (Zhu & Walker, 2026). AI can therefore broaden participation while also allowing established groups to capture disproportionate benefits.

For many developers, the ability to demonstrate skill through working artifacts is also a source of professional pride. If routine code no longer signals mastery, status may shift toward architecture, taste, problem selection, system judgment, domain knowledge, or the ability to coordinate people and agents across a project. Some of these changes may be valuable. They may also create new barriers whose criteria are less visible and harder for newcomers to acquire.

7. Where Do the Saved Resources Go?

Perhaps the most neglected question in debates about offloading is what happens to the productivity dividend. If AI makes a task 30% faster, the released time could support reflection, more ambitious problems, contact with users, learning, mentorship, or recovery. It could instead be absorbed by higher output targets, a larger review queue, or a smaller team. Which outcome follows depends less on the tool than on organizational choices about workload, staffing, incentives, and performance measures.

Before deciding how saved time should be used, however, we must establish that time was saved at all. In a preregistered study of 1,237 participants completing short cognitive tasks, Yu et al. (2026) found no overall difference between unaided and AI-assisted completion times. AI reduced reported

effort, and participants expected it to be faster; it produced a measurable speed gain only on the more difficult tasks. The authors call this mismatch a speedup illusion. The tasks were brief and should not be treated as a model of professional programming. Still, a small randomized study of experienced open-source developers found a similar gap: participants were 19% slower with AI while expecting and (later believing) that it had made them faster (Becker, Rush, Barnes, & Rein, 2025, preprint). These results should not be generalized, but the *speedup illusion* illustrates the possibility that AI can feel easier without being faster. Task-level speed is also too narrow a measure of value. A dashboard showing faster pull-request completion does not tell us whether quality, rework, system understanding, learning, collaboration, or well-being improved. As Goodhart's law suggests, once a measure becomes a target, it can reorganize the practice it was meant to observe (Goodhart, 1975; Strathern, 1997). The relevant outcome is not whether an AI feature saves minutes in isolation but whether the surrounding system converts those minutes into capabilities and conditions that people value. This question must be asked at several levels: individual workload and learning, team communication and mentorship, organizational staffing and incentives, and the profession's entry routes and educational opportunities.

AI may also move some developers closer to problem framing, users, and business decisions. This should not be framed as a sudden move away from "real" software engineering. Professional software work has never consisted only of writing code. In one observational study, coding accounted for 21% of monitored work time, while substantial time went to communication, information seeking, planning, and other collaborative activities (Meyer, Barton, Murphy, Zimmermann, & Fritz, 2017). Value-based software engineering likewise treats stakeholder perspectives and trade-offs between short- and long-term goals as central concerns of the discipline (Salleh, Mendes, Mendes, Lekamlage, & Petersen, 2023). If implementation becomes cheaper, requirements, architecture, integration, user understanding, and product judgment may become relatively more important (this is, however, just a plausible direction, not yet a demonstrated industry-wide outcome). Some developers may not welcome it, but organizations will need to teach and reward the required capabilities rather than assume that they appear automatically.

Reduced working time is one concrete way to distribute productivity gains. Fan, Schor, Kelly, and Gu (2025) studied a six-month four-day-week intervention with no reduction in pay, involving 2,896 employees in 141 organizations and 12 control companies. Burnout, job satisfaction, mental health, and physical health improved; reduced fatigue and sleep problems and better self-reported work ability partly explained these gains. The study was not about software development or AI, and it did not test the quality of technical decisions, but it nevertheless shows that reducing paid working time without reducing pay can support recovery and well-being, which might, in turn, benefit decision-making. Research on human–automation systems shows that sustained monitoring is effortful and stressful, while a recent experiment found that continuous self-monitoring impaired concurrent task performance (Warm et al., 2008; Safati et al., 2026). Better judgment is unlikely to come from maximizing throughput alone. Overall, better decisions are unlikely to come from increased speed, but rather from slowing down.

8. Conclusion

Cognitive offloading is a real concern, but it has become the default lens through which AI-assisted software work is discussed. That lens is too narrow. The more consequential changes may concern motivation, shared understanding, agency, accountability, status, and the use of any time that AI saves. A system can preserve an individual skill while damaging mentorship or ownership; it can also replace a routine while enabling more ambitious and meaningful work. The relevant question is therefore not simply what developers may forget, but what kinds of capabilities, relationships, and working conditions human–AI systems should sustain. A modest next step is for researchers and organizations to evaluate AI adoption with a few measures beyond output speed (for example, learning, team understanding, ownership, and well-being) and to reserve some of any released capacity for reflection, mentoring, and skill development. This would broaden the debate without dismissing the genuine cognitive risks.

9. References

Ackerman, R., & Thompson, V. A. (2017). Meta-reasoning: Monitoring and control of thinking and

reasoning. *Trends in Cognitive Sciences*, *21*(8), 607–617. doi: 10.1016/j.tics.2017.05.004

Amit, I., & Feitelson, D. G. (2026). A large scale survey of motivation in software development. *Journal of Systems and Software*, *231*, 112596. doi: 10.1016/j.jss.2025.112596

Barbey, A. K. (2018). Network neuroscience theory of human intelligence. *Trends in Cognitive Sciences*, *22*(1), 8–20. doi: 10.1016/j.tics.2017.10.001

Bastani, H., Bastani, O., Sungu, A., Ge, H., Kabakcı, Ö., & Mariman, R. (2025). Generative AI without guardrails can harm learning: Evidence from high school mathematics. *Proceedings of the National Academy of Sciences*, *122*(26), e2422633122. doi: 10.1073/pnas.2422633122

Becker, J., Rush, N., Barnes, E., & Rein, D. (2025). *Measuring the impact of early-2025 AI on experienced open-source developer productivity.* (Preprint) doi: 10.48550/arXiv.2507.09089

Beecham, S., Baddoo, N., Hall, T., Robinson, H., & Sharp, H. (2008). Motivation in software engineering: A systematic literature review. *Information and Software Technology*, *50*(9–10), 860–878. doi: 10.1016/j.infsof.2007.09.017

Carruthers, M. (2008). *The book of memory: A study of memory in medieval culture* (2nd ed.). Cambridge: Cambridge University Press. doi: 10.1017/CBO9781107051126

Choudhuri, R., Badea, C., Bird, C., Butler, J. L., DeLine, R., & Houck, B. (2026). AI where it matters: Where, why, and how developers want AI support in daily work. In *Proceedings of the 48th ieee/acm international conference on software engineering: Software engineering in practice (icse-seip 2026).* New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3786583.3786885

Choudhuri, R., Bird, C., Badea, C., Gerosa, M., & Sarma, A. (2026). *You shall not pass! where and why developers draw the line on AI autonomy.* (Preprint) doi: 10.48550/arXiv.2607.00533

Church, L., & Marasoiu, M. (2025). It's OK to want to have a good time. In *Companion proceedings of the 9th international conference on the art, science, and engineering of programming (programming 2025)* (Vol. 134, pp. 5:1–5:4). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi: 10.4230/OASIcs.Programming.2025.5

Clark, A. (2025). Extending minds with generative AI. *Nature Communications*, *16*, 4627. doi: 10.1038/s41467-025-59906-9

Clark, A., & Chalmers, D. J. (1998). The extended mind. *Analysis*, *58*(1), 7–19. doi: 10.1093/analys/58.1.7

Costello, T. H., Rabb, N., Stagnaro, M. N., Pennycook, G., & Rand, D. G. (2026). *Reducing belief in conspiracy theories as they unfold using large language models.* (Preprint) doi: 10.48550/arXiv.2608.06151

Cui, Z. K., Demirer, M., Jaffe, S., Musolff, L., Peng, S., & Salz, T. (2026). The effects of generative AI on high-skilled work: Evidence from three field experiments with software developers. *Management Science*. (Published online) doi: 10.1287/mnsc.2025.00535

David, L., Vassena, E., & Bijleveld, E. (2024). The unpleasantness of thinking: A meta-analytic review of the association between mental effort and negative affect. *Psychological Bulletin*, *150*(9), 1070–1093. doi: 10.1037/bul0000443

Dell'Acqua, F., Ayoubi, C., Lifshitz-Assaf, H., Sadun, R., Mollick, E., Mollick, L., … Lakhani, K. R. (2025). *The cybernetic teammate: A field experiment on generative AI reshaping teamwork and expertise* (Working Paper No. 33641). National Bureau of Economic Research. (A revised version was published in Organization Science in 2026) doi: 10.3386/w33641

Dhanorkar, S., Passi, S., & Vorvoreanu, M. (2026). *Human oversight of agentic systems in practice: Examining the oversight work, challenges, and heuristics of developers using software agents.* (Preprint; accepted at FAccT 2026) doi: 10.48550/arXiv.2606.05391

Doshi, A. R., & Hauser, O. P. (2024). Generative AI enhances individual creativity but reduces the collective diversity of novel content. *Science Advances*, *10*(28), eadn5290. doi: 10.1126/sciadv.adn5290

Drosos, I., Sarkar, A., Xu, X., & Toronto, N. (2025). *"it makes you think": Provocations help restore critical thinking to AI-assisted knowledge work.* (Preprint) doi: 10.48550/arXiv.2501.17247

Ehn, P. (1990). *Work-oriented design of computer artifacts*. Hillsdale, NJ: Lawrence Erlbaum Associates. (The original Scandinavian edition is often dated 1988; this record follows the 1990 English-language edition cited in the manuscript)

Elish, M. C. (2019). Moral crumple zones: Cautionary tales in human-robot interaction. *Engaging Science, Technology, and Society*, *5*, 40–60. doi: 10.17351/ests2019.260

Fan, W., Schor, J. B., Kelly, O., & Gu, G. (2025). Work time reduction via a four-day workweek finds improvements in workers' well-being. *Nature Human Behaviour*, *9*, 2153–2168. doi: 10.1038/s41562-025-02259-6

Fisher, M., Goddu, M. K., & Keil, F. C. (2015). Searching for explanations: How the internet inflates estimates of internal knowledge. *Journal of Experimental Psychology: General*, *144*(3), 674–687. doi: 10.1037/xge0000070

Gagné, M., Parker, S. K., Griffin, M. A., Dunlop, P. D., Knight, C., Klonek, F. E., & Parent-Rocheleau, X. (2022). Understanding and shaping the future of work with self-determination theory. *Nature Reviews Psychology*, *1*(7), 378–392. doi: 10.1038/s44159-022-00056-w

Goodhart, C. A. E. (1975). Problems of monetary management: The U.K. experience. In *Papers in monetary economics* (Vol. 1, pp. 1–20). Sydney: Reserve Bank of Australia. (Conference-paper version; a revised collection appeared in 1976)

Hicks, C. (2026). *The psychology of software teams* (1st ed.). CRC Press.

Hicks, C. M., Lee, C. S., & Ramsey, M. (2024). Developer thriving: Four sociocognitive factors that create resilient productivity on software teams. *IEEE Software*, *41*(4), 68–77. doi: 10.1109/MS.2024.3382957

Hutchins, E. (1995). *Cognition in the wild*. Cambridge, MA: MIT Press.

Kalam, K. A., Masoud, F. D., Muntaser, A., Ranga, R., Geng, X., & Goyal, M. (2025). ChatGPT as a learning tool for medical students: Results from a randomized controlled trial. *Cureus*. doi: 10.7759/cureus.85767

Kosmyna, N., Hauptmann, E., Yuan, Y. T., Situ, J., Liao, X.-H., Beresnitzky, A. V., … Maes, P. (2025). *Your brain on ChatGPT: Accumulation of cognitive debt when using an AI assistant for essay writing task.* (Preprint, version 2) doi: 10.48550/arXiv.2506.08872

Lee, B. C., & Chung, J. (2024). An empirical investigation of the impact of ChatGPT on creativity. *Nature Human Behaviour*, *8*, 1906–1914. doi: 10.1038/s41562-024-01953-1

Lee, H.-P., Sarkar, A., Tankelevitch, L., Drosos, I., Rintel, S., Banks, R., & Wilson, N. (2025). The impact of generative AI on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers. In *Proceedings of the 2025 chi conference on human factors in computing systems* (pp. 1–22). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3706598.3713778

Lindo, M. R., & Cutad, C. A. (2024). Impact of generative AI on self-regulated learning and cognitive offloading. *IMCC Journal of Science*, *4*(2), 18–22. doi: 10.65931/c7k2p5z9

Medeiros, K., Cropley, D. H., Marrone, R. L., & Reiter-Palmon, R. (2025). Human–AI co-creativity: Does ChatGPT make us more creative? *Journal of Creative Behavior*, *59*(2), e70022. doi: 10.1002/jocb.70022

Meyer, A. N., Barton, L. E., Murphy, G. C., Zimmermann, T., & Fritz, T. (2017). The work life of developers: Activities, switches and perceived productivity. *IEEE Transactions on Software Engineering*, *43*(12), 1178–1193. doi: 10.1109/TSE.2017.2656886

Mohamed, A., Assi, M., & Guizani, M. (2026). *The impact of LLM-assistants on software developer productivity: A systematic review and mapping study.* (Preprint, version 3; the latest version reports 39 included studies) doi: 10.48550/arXiv.2507.03156

Nixon, N., Choi, J., Martins De Bastos, P., Samadi, M. A., Mehner, L., Park, S., & JaQuay, S. (2026). *The social cost of an AI teammate: How an artificial teammate reshapes human–human communication in small-team decision-making.* (Preprint) doi: 10.48550/arXiv.2607.27179

Pritchard, D. (2026). Educating for intellectual character and the AI revolution in education. *Studies in Philosophy and Education*. (Published online 24 July 2026) doi: 10.1007/s11217-026-10094-z

Qiao, Y., Shihab, M. I. H., Haque, S., & Hundhausen, C. (2026). *Code comprehension with GitHub Copilot: Performance gains, comprehension trade-offs, and behavioral predictors in brownfield programming.* (Preprint, version 2, 11 May 2026) doi: 10.48550/arXiv.2511.02922

Rachal, J. R. (1988). Gutenberg, literacy, and the ancient arts of memory. *Adult Education Quarterly*, *38*(3), 125–135.

Risko, E. F., & Gilbert, S. J. (2016). Cognitive offloading. *Trends in Cognitive Sciences*, *20*(9), 676–688. doi: 10.1016/j.tics.2016.07.002

Rozenblit, L., & Keil, F. C. (2002). The misunderstood limits of folk science: An illusion of explanatory depth. *Cognitive Science*, *26*(5), 521–562. doi: 10.1207/s15516709cog2605_1

Ryan, R. M., & Deci, E. L. (2000). Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist*, *55*(1), 68–78. doi: 10.1037/0003-066X.55.1.68

Safati, A. B., Seli, P., & Smilek, D. (2026). The costs of monitoring: Does self-catching the wandering mind influence task performance and/or residual probe-caught mind-wandering? *Consciousness and Cognition*, *144*, 104112. doi: 10.1016/j.concog.2026.104112

Salleh, N., Mendes, E., Mendes, F., Lekamlage, C. D., & Petersen, K. (2023). Value-based software engineering: A systematic mapping study. *e-Informatica Software Engineering Journal*, *17*(1), 230106. doi: 10.37190/e-Inf230106

Santarnecchi, E., Galli, G., Polizzotto, N. R., Rossi, A., & Rossi, S. (2014). Efficiency of weak brain connections support general cognitive functioning. *Human Brain Mapping*, *35*(9), 4566–4582. doi: 10.1002/hbm.22495

Sarkar, A. (2024). AI should challenge, not obey. *Communications of the ACM*, *67*(10), 18–21. doi: 10.1145/3649404

Shin, M., Kim, J., van Opheusden, B., & Griffiths, T. L. (2023). Superhuman artificial intelligence can improve human decision-making by increasing novelty. *Proceedings of the National Academy of Sciences*, *120*(12), e2214840120. doi: 10.1073/pnas.2214840120

Şimşek, A. C., Anders, G., Göth, J., Specht, L., & Huff, M. (2025). Is ChatGPT a good study companion? the role of AI-generated summaries and reflective prompts in learning from educational videos. *Computers and Education: Artificial Intelligence*, *9*, 100512. doi: 10.1016/j.caeai.2025.100512

Song, F., Agarwal, A., & Wen, W. (2026). *The impact of generative AI on collaborative open-source software development: Evidence from GitHub Copilot.* (Preprint; record uses the 2026 version cited in the manuscript) doi: 10.48550/arXiv.2410.02091

Sparrow, B., Liu, J., & Wegner, D. M. (2011). Google effects on memory: Cognitive consequences of having information at our fingertips. *Science*, *333*(6043), 776–778. doi: 10.1126/science.1207745

Storey, M.-A. (2026). *From technical debt to cognitive and intent debt: Rethinking software health in the age of AI.* (Preprint) doi: 10.48550/arXiv.2603.22106

Strathern, M. (1997). Improving ratings: Audit in the british university system. *European Review*, *5*(3), 305–321. doi: 10.1002/(SICI)1234-981X(199707)5:3<305::AID-EURO184>3.0.CO;2-4

Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes* (M. Cole, V. John-Steiner, S. Scribner, & E. Souberman, Eds.). Cambridge, MA: Harvard University Press.

Wadinambiarachchi, S., Kelly, R. M., Pareek, S., Zhou, Q., & Velloso, E. (2024). The effects of generative AI on design fixation and divergent thinking. In *Proceedings of the 2024 chi conference on human factors in computing systems* (pp. 1–18). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3613904.3642919

Warm, J. S., Parasuraman, R., & Matthews, G. (2008). Vigilance requires hard mental work and is stressful. *Human Factors*, *50*(3), 433–441. doi: 10.1518/001872008X312152

Yang, L., Holtz, D., Jaffe, S., Suri, S., Sinha, S., Weston, J., … Teevan, J. (2022). The effects of remote work on collaboration among information workers. *Nature Human Behaviour*, *6*(1), 43–54. (Published online in 2021; assigned to the 2022 journal issue) doi: 10.1038/s41562-021

-01196-4

Yu, S., Cheng, M., Jabbar, A., Sucholutsky, I., Collins, K. M., Jurafsky, D., & Hawkins, R. D. (2026). Cognitive offloading and the speedup illusion in human–AI interaction. In *Proceedings of the 48th annual meeting of the cognitive science society.* doi: 10.48550/arXiv.2605.23177

Zakharov, I., Tabueva, A., Adamovich, T., Kovas, Y., & Malykh, S. (2020). Alpha band resting-state EEG connectivity is associated with non-verbal intelligence. *Frontiers in Human Neuroscience*, *14*, 10. doi: 10.3389/fnhum.2020.00010

Zhu, K., & Walker, D. (2026). The AI democratization paradox: Evidence from decentralized knowledge communities. *Management Science*. (Published online) doi: 10.1287/mnsc.2024.04717

Beyond the Screen: Taste, Smell, and the Untapped Sensorium of Human-Computer Interaction

Marianna Obrist
University College London

Keynote abstract

For decades, the psychology of programming has largely taken for granted the sensory channels through which humans and computers communicate, including visual displays, auditory feedback, and haptic responses. But what if we have been working with only a fraction of the human sensorium? This talk ventures beyond the familiar into the largely uncharted territory of the chemical senses, (taste and smell) as viable, meaningful modalities for interaction design. It presents examples of research exploring interaction modalities beyond the screen, with a particular focus on the chemical senses. I will walk through studies and prototypes that bring smell into computing contexts, and share findings on why taste is an exciting, worthwhile interaction modality in Human-Computer Interaction (HCI). I will discuss what makes the chemical senses both compelling and scientifically demanding: their deep ties to memory and emotion, their hedonic immediacy, and the significant technical and methodological challenges that come with integrating them into interactive systems. This talk offers a research perspective on what the field has achieved, where the hard problems lie, and why the psychology of programming community is uniquely well placed to advance this work, as part of a broader cross-disciplinary effort spanning cognition, interaction design, and multisensory hardware development.

Speaker bio

Marianna Obrist is Professor of Multisensory Interfaces in the Department of Computer Science at University College London (UCL). She pioneers research on touch, taste, and smell in human-computer interaction (HCI), with applications spanning virtual reality, healthcare, and wellbeing. Her recent projects include the EPSRC/NIHR Smell Care initiative on digital smell training and testing, the EU FET Touchless.AI project on touchless social interaction, and the UKRI Textile Circularity Centre on immersive textile experiences. She is also co-founder and Chief Scientific Officer of Hynt Labs Limited, a spin-out developing digital scent technologies for personalised sleep solutions. Marianna has authored over 100 publications in leading journals and conferences and co-wrote Multisensory Experiences: Where the Senses Meet Technology (Oxford University Press). See her website for further information, examples, and xSense design cards: multisensoryexperiences.co.uk.

Talking as Theory Reconstruction: Eliciting Architectural Decisions from Developers via Interview and LLM-Based Synthesis

Bernhard Mayr
Storyblok GmbH
bernhard.mayr@storyblok.com

Alexander Feiglstorfer
Storyblok GmbH
af@storyblok.com

Dominik Angerer
Storyblok GmbH
da@storyblok.com

Abstract

In *Programming as Theory Building* (1985), Peter Naur argued that programming is the construction of a theory in the programmer's mind: a theory that cannot be captured in code or documentation, and that dies when its bearers leave the project. The spread of large language models (LLMs), and of agentic development tools that need explicit decision artefacts to operate, makes the claim worth revisiting. This work-in-progress paper reports on a two-stage method for externalising part of the implicit theory behind early-stage software projects. A human interviewer first holds a semi-structured, podcast-style conversation with the developers involved, using open questions to draw out narrative accounts and tacit reasoning. An LLM then distils the transcripts into Architecture Decision Records (ADRs) that record each decision with its rationale, the alternatives, and the trade-offs. To assess fidelity, we cross-check the generated ADRs against the codebase, looking for where stated intent and current implementation match, where they leave gaps, and where they contradict each other. We expect a mixed answer to the question of how much of the theory Naur describes survives reconstruction. Decisions and their trade-offs look partly recoverable; the deeper mental models behind them likely do not. The resulting ADRs may also be useful as context for agentic coding tools, which increasingly need machine-readable rationale to work safely on existing code. We present preliminary findings from two pilot interviews and discuss methodological challenges, among them post-hoc rationalisation, interviewer framing, and LLM hallucination during synthesis.

1. Introduction

Naur (1992) argued that a program is not primarily its source code, but the theory held in the minds of its programmers. The theory connects the program to its problem domain, explains why each part is the way it is, and lets developers respond constructively when requirements change. Code and documentation alone do not recover it. When its bearers leave, it dies.

Forty years later the claim has new practical urgency. Agentic coding tools, AI systems that read, modify, and extend codebases on their own, routinely have to reconstruct rationale that was never written down. In Naur's terms, they arrive like developers who missed all the original meetings: they can read the artefacts, but they do not share the theory.

Architecture Decision Records (ADRs), short structured documents that record a decision together with its context and consequences (Nygard, 2011), have meanwhile emerged as a lightweight way to make rationale explicit. If we can elicit rationale from developers and write it down as ADRs before the knowledge is lost, some part of the implicit theory becomes shared context that future engineers and agentic tools can both read.

This paper asks how much of the implicit theory behind an early-stage software system can be recovered through semi-structured developer interviews and LLM-assisted synthesis into ADRs. We do not expect to refute Naur. We want to locate, in practice, the boundary between what can be reconstructed and what cannot.

2. Background

In Naur's account, the theory a programmer builds has three components. The first relates the program to the real-world domain it models, what we will call the representational layer. The second explains why the program is structured as it is rather than otherwise: the rationale layer. The third gives developers an

anticipatory sense of what would and would not fit the system, a layer that tends to surface only when a new requirement appears (Naur, 1992). Domain documentation and naming conventions recover part of the first layer. The third resists articulation even by the people who hold it. ADRs sit squarely in the second layer, which sets this paper's scope: we aim to recover the rationale layer ADRs were designed to express, a narrower target than Naur's theory in full.

ADRs sit within a longer design rationale (DR) tradition of capturing the reasons behind a decision as an artefact in its own right. Early DR notations were formal argumentation structures: IBIS represented a design conversation as a graph of issues, positions, and arguments (Kunz and Rittel, 1970); QOC decomposed a decision into a question, a set of options, and the criteria used to weigh them (MacLean et al., 1991); DRL added explicit goals and claims for richer decision analysis (Lee, 1991). Capturing rationale in such formal graphs proved disruptive alongside design work, a difficulty documented early on as the capture problem (Conklin and Burgess-Yakemovic, 1991). Lightweight, prose-based templates such as ADRs won out over the formal notations for that reason, at the cost of the explicit alternative-versus-criteria structure QOC and DRL required. The interview-based elicitation in this paper is itself a known DR capture strategy, reconstruction: rationale recovered after the fact rather than recorded as decisions are made (Moran and Carroll, 1996). The contribution here is not the capture strategy but pairing it with LLM-based synthesis and a code-level validation pass.

The choice of the word *reconstruction* in the title is intentional. As we argue in Section 5, the interview does not simply retrieve a theory that pre-exists in the developer's mind. It co-produces an account in dialogue with the interviewer. This limit follows from what an interview, as a dialogic act, produces and cannot be interviewed around.

ADRs are usually written at the moment of the decision. The retrospective case is harder. We are trying to elicit reasoning for decisions made years earlier, much of which was never written down in the first place.

LLM-based coding agents now operate across entire codebases without close human supervision (Hou et al., 2024). They face the same problem Naur described. A codebase contains the outcomes of decisions but not the reasoning behind them. Without the reasoning, an agent cannot tell a deliberate constraint from an accidental one, and may quietly refactor away invariants that exist for reasons not visible in the code. ADRs, even retrospectively reconstructed ones, are written in a form that both developers and coding agents can read. They sit between tacit human knowledge and the explicit context that agentic tools need.

3. Method

The pipeline combines human interviewing with LLM-assisted synthesis, plus a separate validation step against the codebase. The split is deliberate: it keeps each part doing what it is best at, and lets validation catch any drift between what was said and what was actually built. The method has three stages.

Stage 1: Semi-structured interview. A human interviewer holds a semi-structured conversation with one or more developers who were involved in the early stages of the system under study. Questions are open and narrative ("what led to this decision?") rather than closed ("did you choose X because of Y?"), and the interviewer adapts as the conversation moves (Kvale and Brinkmann, 2009; Patton, 2002). The interviewer is briefed to avoid introducing architectural framings or suggesting alternatives during elicitation. Wherever possible the interview takes place in the participant's native or strongest language: meaning is constituted in language, and a second language introduces interpretive loss and discourages free expression (Temple and Young, 2004). Interviews are recorded and transcribed. We hypothesise that the interviewer cannot be a complete outsider. Picking out the architecturally significant decisions in a given codebase takes prior exposure to the system, as does knowing which follow-up questions will lead to rationale and not just a restatement of what the code does. In Naur's terms, the interviewer needs to already share part of the theory; a project-agnostic interview will not get as deep.

Stage 2: LLM-assisted synthesis. The transcript is given to a large language model with instructions to identify the architectural decisions discussed and to format each as an ADR with title, status, context, decision, and consequences. The model is told to flag uncertainty when it is unsure. The specific prompt used in our pilot is shown in Section 4.

Stage 3: Validation. Validation has three steps. First, the interviewer and long-standing team members review the generated ADRs together; the review sharpens shared understanding and improves ADR quality at the same time. Second, where the transcript does not give a date, an LLM reconstructs one from the project's git commit history, producing a timeline view of how the architecture evolved. Third, the final ADRs are cross-checked against the source code with LLM assistance, to see whether the recorded decisions still hold or whether implementation has since drifted from stated intent.

Separating elicitation from synthesis is intentional. Human interviewers can build rapport, sense when to follow up, and pick up on the unspoken cues a model cannot. The LLM does the tedious work of imposing structure on hours of transcript. Separating the two also keeps the model from steering the interview toward architectural patterns it has seen before.

4. Pilot Study

We ran two pilot interviews. The first lasted around 30 minutes and was conducted by the Storyblok MCP product owner with one of the initial software architects of the Storyblok MCP Server, an open-source integration layer that exposes Storyblok's content management capabilities to AI coding assistants. The transcript was given to Claude Sonnet 4.6 via Claude Cowork with the following prompt:

> I've just given you a transcript of an interview we conducted with one of the initial software architects of Storyblok's MCP Server where we asked him about some architectural decisions that have been made while developing the MCP Server. I would like you to extract these architectural decisions and document them as architecture decision records (in English and markdown). Be as precise as possible and flag uncertainties.

The synthesis produced eleven ADRs. Two are reproduced below as representative examples.

ADR-001: Multi-step workflow (Search → Describe → Execute). Mapping the entire Storyblok Management API to individual MCP tools causes context bloat: every tool description gets injected into the model's system prompt, burning tokens before any conversation begins. An earlier two-step approach (search → execute) had no confirmation step before execution. The decision was therefore to introduce a three-step workflow: *search* (keyword-based operation discovery, scored by relevance), *describe* (surfaces the full parameter schema for the matched operation before execution), and *execute* (performs the API call). The initial tool footprint stays minimal, with only seven tools exposed at connection time, and operations are discovered on demand. The extra round-trip introduced by the *describe* step is offset by avoiding incorrect executions caused by insufficient knowledge of the parameters.

ADR-003: Role-based endpoint exposure. Different Storyblok users, content editors, developers, administrators, should only see and be able to call the API operations relevant to their role; exposing the full surface to everyone raises the risk of accidental or unauthorized operations. The decision was to accept a *role* query parameter on the MCP server URL that restricts which operations the model can see, enforced consistently across *search*, *describe*, and *execute*, so an operation hidden from search cannot be described or executed either. Embedding the role in the URL itself, rather than in a header or a runtime setting, keeps the restriction transparent to the end user and resistant to accidental override. The trade-off is that adding a role or changing role membership requires updating the OpenAPI overlay and redeploying, rather than a runtime configuration change.

The second interview lasted around 55 minutes and was conducted with one of the co-founders and initial developers of Storyblok, a headless CMS used by large-scale commercial clients. Interviewer and participant spoke their shared native language (Austrian German). The synthesis produced eight ADRs covering the founding and early scaling of the platform. We refer to several of them in the discussion below. One is reproduced here.

ADR-002: Visual Editor V2, full rewrite instead of incremental upgrade. The original Visual Editor (V1) had been written by the two co-founders starting from a pre-stable release of Vue.js. Over several years of growth the surrounding JavaScript ecosystem moved on, and a dedicated frontend team was eventually hired to whom the codebase was, by that point, only fully legible to the two founders who had written it. Rather than upgrading V1 in place and handing it to the new team for maintenance, the decision was to rewrite it from scratch as V2. Two reasons were given together: the Vue.js ecosystem had moved far enough to justify a clean start, and a rewrite would force the new team to build genuine understanding of the architecture, something inheriting a maintained V1 codebase would not have produced on its own. The stated trade-off was the cost of a full rewrite against an incremental upgrade, with the founders' own gloss that it is often easier to rewrite a system than to comprehend an existing one from its artefacts alone.

We are also cross-checking the Storyblok ADRs against the current codebase to see how the method holds up on a long-running production system.

5. Challenges to Validity

Several psychological phenomena limit what an interview-based reconstruction pipeline can produce. We focus on four.

Confabulation and post-hoc rationalisation. Nisbett and Wilson (1977) showed that people have poor introspective access to their own cognitive processes and readily produce plausible causal accounts that do not match the actual causes. When a developer explains an architectural decision years later, the explanation is at least partly reconstructed from current beliefs rather than recalled from the original moment. The MCP Server interview produced a clear example: one decision was described retrospectively as a considered trade-off, but the same interviewee also noted that the relevant aspect of the architecture had not actually been thought about deliberately at the time.

The articulation problem. Polanyi (1966) observed that we know more than we can tell, a claim that lines up with Naur's account of theory. Both hold that expert knowledge is partly tacit and exceeds what any artefact can express. Architectural judgement is partly procedural and resists language. Forcing verbalisation can also distort tacit knowledge: Schooler et al. (1993) documented that verbal descriptions of non-verbal knowledge can degrade subsequent performance and produce accounts that no longer align with the underlying cognition. Whatever the pipeline produces is therefore biased toward the verbalisable subset of the theory.

The observer-construction problem. This is probably the deepest challenge for the paper's framing. The interview does not simply pull a theory out of the developer's mind. It co-produces an account in dialogue with the interviewer. One of the ADRs from the Storyblok interview, documenting a decision about a cache version parameter, only took its shape through the interviewer's clarifying questions about the mechanism. The account did not exist in that form before the interview took place. The same partial sharing of theory that we treated as a precondition in Stage 1 reappears here as a source of bias: the interviewer in both pilots already knew enough about the system to ask useful follow-up questions, but that prior knowledge also shapes what the account ends up looking like. This is why *reconstruction* is more honest than *extraction*: the artefacts the method produces are shaped by the conversation, not by verbatim transcripts of dormant mental models. Design rationale researchers reached a similar conclusion independently: Carroll and

Rosson (1991) argue for scenario-based accounts of design reasoning over formal argumentation graphs, on the grounds that rationale elicited after the fact is argued into shape in the telling rather than retrieved intact.

Illusory theory building. The first three challenges concern what the pipeline cannot recover. The fourth concerns what it may seem to have recovered when it actually has not. Clayton (2026) puts it bluntly: "The issue is that it's easy to convince yourself that you've built theory when you haven't." The same risk applies to retrospective reconstruction. A plausible-looking ADR can create the impression of recovered rationale when the developer's account was shallow to begin with. The illusion of explanatory depth (Rozenblit and Keil, 2002) hits hard in agent-assisted development: a developer who accepts coding agent output without close scrutiny can end up feeling they understand the code better than they do, and may then explain decisions with that inflated sense of certainty in the interview. The Dunning–Kruger effect (Kruger and Dunning, 1999) adds a second layer: limited awareness of one's own knowledge gaps inflates confidence in the resulting account. A third mechanism is cognitive offloading: handing synthesis to an LLM can itself produce an illusion of competence, since users systematically overestimate their understanding of AI-generated outputs (Jose et al., 2025). We therefore treat the output of the pipeline as a provisional artefact that needs expert review, not a faithful reconstruction.

These challenges do not invalidate the approach. They define what the method actually produces: partial and dialogically shaped approximations of the implicit theory, biased toward what can be put into words. Whether such approximations are useful, whether for engineers onboarding to a codebase, for agentic tools that need rationale, or for preserving institutional memory, is an empirical question. We turn to it next.

6. Discussion

The pilot interviews suggest that decisions and their trade-offs are *partly* reconstructible through this pipeline. Several ADRs generated from the MCP Server interview contain specific and cross-checkable claims: runtime observability data, alternative designs considered and rejected, downstream constraints that cascade from a single decision. These go beyond what reading the code alone surfaces.

Practitioners have made a similar point about agentic tooling: Ekrem (2025) argues that as reflexive AI use lets developers accept generated code with little cognitive engagement, ADRs become one of the few documentation forms that LLM-based coding agents can actually use to recover design intent.

Deeper aspects of the theory Naur describes, the anticipatory judgements that constrain the design space before any decision is framed and the sense of what would and would not "fit" the system as a whole, did not surface clearly in the pilots. We do not expect them to. Where the boundary sits between the reconstructible and the irretrievable is something we are still working out.

The pilots also surfaced a finding that matters in practice: knowledge fragmentation accelerates quickly as teams grow. The Storyblok interview revealed that several decisions made years earlier were already unknown to current team members, and that later, incompatible revisions had been made in internal tooling without any cross-reference to the original rationale. Even a few years of team turnover seem to produce significant gaps. The method is most useful while the original decision-makers are still around.

One observation from the Storyblok interview is a direct instance of Naur's central claim. The founders described how, once the first dedicated frontend team was hired, continuing to develop V1 of the visual editor turned out to be impractical. It was easier to let the new team build V2 on current technologies than to transfer the theory embedded in V1. This is exactly the scenario Naur predicts: the theory had been lost to the point that rebuilding was preferable to inheriting the existing artefact.

The conference theme of *Juxtapositions* fits the work. The paper sits between human tacit knowledge and developer-readable artefact, and between Naur's pessimism about documentation and the practical need for AI systems to understand what they are modifying. We are not trying to resolve the tension,

only to map it empirically.

7. Conclusion

We have presented a three-stage method for reconstructing architectural rationale through semi-structured interviews and LLM-assisted synthesis into ADRs, and reported preliminary findings from two pilot studies. The method seems able to produce partial but useful reconstructions of the implicit theory behind early design decisions. Four psychological constraints set principled limits on what it can reach: confabulation, the articulation problem, the observer-construction problem, and the risk of illusory theory building. Our ongoing work is cross-checking the generated ADRs against their respective codebases, and following up on the implications for knowledge preservation practice and for agentic development tools.

References

Carroll, J. M. and Rosson, M. B. (1991). Deliberated evolution: Stalking the view matcher in design space. *Human–Computer Interaction*, 6(3-4):281–318.

Clayton, I. (2026). Programming as theory building is true now more than ever. `https://slightknack.dev/daily/2026-02-21/`. Accessed May 2026.

Conklin, J. and Burgess-Yakemovic, K. C. (1991). A process-oriented approach to design rationale. *Human–Computer Interaction*, 6(3-4):357–391.

Ekrem, C. (2025). Programming as theory building: Why senior developers are more valuable than ever. `https://cekrem.github.io/posts/programming-as-theory-building-naur/`. Accessed May 2026.

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., and Wang, H. (2024). Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–79.

Jose, B., Joseph, D., Mohan, V., Alexander, E., Varghese, S. K., and Roy, A. (2025). Outsourcing cognition: The psychological costs of AI-era convenience. *Frontiers in Psychology*, 16:1645237.

Kruger, J. and Dunning, D. (1999). Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of Personality and Social Psychology*, 77(6):1121–1134.

Kunz, W. and Rittel, H. W. J. (1970). Issues as elements of information systems. Technical Report Working Paper No. 131, Institute of Urban and Regional Development, University of California, Berkeley.

Kvale, S. and Brinkmann, S. (2009). *InterViews: Learning the Craft of Qualitative Research Interviewing*. Sage Publications, Thousand Oaks, CA, 2nd edition.

Lee, J. (1991). Extending the Potts and Bruns model for recording design rationale. In *Proceedings of the 13th International Conference on Software Engineering*, pages 114–125.

MacLean, A., Young, R. M., Bellotti, V. M. E., and Moran, T. P. (1991). Questions, options, and criteria: Elements of design space analysis. *Human–Computer Interaction*, 6(3-4):201–250.

Moran, T. P. and Carroll, J. M., editors (1996). *Design Rationale: Concepts, Techniques, and Use*. Lawrence Erlbaum Associates, Mahwah, NJ.

Naur, P. (1992). Programming as theory building. In *Computing: A Human Activity*, pages 37–48. ACM Press / Addison-Wesley. Originally published in *Microprocessing and Microprogramming*, 15(5), 1985, pp. 253–261.

Nisbett, R. E. and Wilson, T. D. (1977). Telling more than we can know: Verbal reports on mental processes. *Psychological Review*, 84(3):231–259.

Nygard, M. T. (2011). Documenting architecture decisions. `https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions`. Accessed May 2026.

Patton, M. Q. (2002). *Qualitative Research and Evaluation Methods*. Sage Publications, Thousand Oaks, CA, 3rd edition.

Polanyi, M. (1966). *The Tacit Dimension*. Doubleday, New York.

Rozenblit, L. and Keil, F. (2002). The misunderstood limits of folk science: An illusion of explanatory depth. *Cognitive Science*, 26(5):521–562.

Schooler, J. W., Ohlsson, S., and Brooks, K. (1993). Thoughts beyond words: When language overshadows insight. *Journal of Experimental Psychology: General*, 122(2):166–183.

Temple, B. and Young, A. (2004). Qualitative research and translation dilemmas. *Qualitative Research*, 4(2):161–178.

Investigation into the Factors Which Influence Students to Study Computer Science between Second and Third-Level Education – A Working Study

Miriam Harte
School of Enterprise Computing
And Digital Transformation
TU Dublin
miriam.harte@tudublin.ie

Keith Quille
School of Enterprise Computing
And Digital Transformation
TU Dublin
keith.quille@tudublin.ie

Rajesh Jaiswal
School of Enterprise Computing
And Digital Transformation
TU Dublin
rajesh.jaiswal@tudublin.ie

Keith Nolan
School of Enterprise Computing
And Digital Transformation
TU Dublin
keith.nolan@tudublin.ie

Abstract

As the demand for digital skills among second-level students continues to grow, there are still unanswered questions regarding the reasons which motivate students to study Computer Science. Much research has been done into the factors which influence students to choose Computer Science at third-level, but corresponding research does not exist across second-level education in Ireland. Furthermore, no longitudinal study exists whereby changes in students' opinions of Computer Science are monitored across latter stages of second-level and into initial third-level education. Intrinsic factors such as academic self-determination and self-efficacy may motivate students to feel well suited to a technical subject such as Computer Science. Similarly, the opinions of family and friends and societal perceptions of Computer Science could be considered relevant extrinsic factors which may influence student subject and career choices. This paper will outline a work-in-progress study designed to longitudinally investigate the educational experiences which influence students to study Computer Science between second and third-level education in Ireland, through a set of surveys, focus groups and interviews.

1. Introduction

At third-level education in Ireland, we are seeing challenges around student admission and retention numbers in the field of Computer Science (CS). Technical complexity, a lack of student confidence and misunderstanding of what CS actually consists of, are considered to be influencing factors within this issue (Tims, Tucker, Weiss, & Zweben, 2024). However, we are not yet fully aware of the factors which influence students to take CS as an optional subject at second-level.

In recent years, the Irish Department of Education has taken action to integrate computing curricula across all stages of formal education. Ireland's second-level education system spans the ages of 12-18 and consists of two phases of state examination - Junior Cycle (JC) at age 14/15 and Senior Cycle (SC) at age 17/18. Students can take an interim year of semi-structured, self-directed learning, Transition Year, around the age of 15. Currently, second-level formal computing is offered as an optional short course in Coding at Junior Cycle and as a state-examined subject of Computer Science at Senior Cycle. Since the introduction of the current Leaving Certificate Computer Science (LCCS) curriculum in Ireland (National Council for Curriculum and Assessment, 2025), we have seen some interest in second-level students in selecting this topic as an optional Leaving Certificate subject. However, we do not yet know much about the experiences which influence students to select Computer Science for their Leaving Certificate.

Recent research by the Ireland's Economic and Social Research Institute (ESRI) has indicated that factors such as school location and social composition influence overall student academic achievement,

rather than the neighbourhood of individual students. Carried out longitudinally, ESRI's *Growing Up in Ireland* study examined the correlations between student background and overall academic achievement (Murray & Smyth, 2020). While this work gives great insight into the current state of art regarding the expected Leaving Certificate outcomes in second-level students, it does not indicate how technical STEM subjects such as Computer Science are affected. At present, no such research exists in Ireland. Despite the ESRI's Growing Up in Ireland study, no longitudinal data exists in Ireland in relation to student rationale for their continuation of Computer Science between second and third-level. As part of this research study, we intend to investigate what experiences and factors motivate second-level students to take LCCS and then potentially continue their CS studies at third-level.

Below, we will highlight the research goals, methodology and expected outcome of this unique study, as well as the milestones within the first year of its progress.

2. Research Questions

This study will be guided by the following research questions:

1. How do students' programming self-efficacy, conceptual understanding of computer science, and computing identity influence their decision to persist with computing across the transition from second-level to third-level education?

2. How do second-level learning contexts - including access to computing resources, instructional experiences, and institutional proximity - shape students' beliefs about programming difficulty, relevance, and their likelihood of success in third-level computing?

3. How do prior formal and informal programming experiences influence students' motivation to learn computing and their mental models of programming at the point of entry to third-level education?

Through these research questions, we hope to determine how socioeconomics, family influence and school infrastructure may shape a student's decision to study CS at Leaving Certificate level. Furthermore, we intend to seek any potential correlation between student uptake of CS and pre-existing experience of CS. It is well-documented that early interventions increase student engagement in STEM (Chittum, Jones, Akalin, & Schram, 2017) and the same can be said for CS. We would like to see if these past interventions have lasting impact beyond second-level formal education settings and if they change over the course of the latter years of formal education.

3. Methodology

In order to carry out this study, we will use the following research methods:

- Surveys and Questionnaires: Hosted to assess students' perceptions of available educational resources, career aspirations, and influences on their academic decision-making processes. Questions will cover access to technology, quality of computing education, and personal aspirations.

- Interviews: Facilitation of in-depth interviews with participants at multiple points during the study to gain insights into their evolving attitudes towards computing and the role it may have in their future education.

- Focus Groups: Discussions among students about their experiences and perceptions of computing as a career choice.

3.1. University Ethics Review

Full ethical approval was sought and has been granted by TU Dublin's Research Ethics Committee.

3.2. Survey Development

The main form of data gathering within this study consists of student surveys, distributed across second-level schools. Schools will be recruited via the CSinc Research Centre's teacher mailing list, whereby teachers can volunteer to distribute our research survey to their students.

Within this pilot, we intend to engage with students who are in either Transition Year, 5th Year or 6th Year of Secondary School. Students who enter this stage of their formal education spend the latter two years (5th and 6th Year) preparing to take their final state examination, the Leaving Certificate. As part of the Leaving Certificate, students study up to seven subjects, including three mandatory subjects and four "choice" subjects. Upon completing the Leaving Certificate, students' examination results will be converted into points which will determine which third-level course a student is eligible to study as part of their higher education.

By including Transition Year students, we will also get the chance to engage with students who may be currently exploring more hands-on, project-based subjects, such as coding or digital technology. In Ireland, there is a plethora of introductory Computer Science programmes and events, aimed at engaging TY students with tech and CS. It is our hope that the TY students who participate in our study, will also have some pre-existing educational experience of CS at this stage in their educational journey.

Additionally, the rationale for inviting students in 5th and 6th Year to participate in this research pilot study is two-fold; Firstly, students in 5th/6th Year have already selected their Leaving Certificate subjects, so it may be easier to specifically engage with students opinions on their desire or lack of desire in studying CS for their final formal exam in second-level education. Secondly, most students in 5th/6th Year of Secondary School will be over the age of 16, thus meeting the digital age of consent in Ireland. This means that, even though some students are under the age of 18, they will still be able to self-consent to take part in this research study and will not require parental consent to allow this.

Surveys have been designed to explore the educational experiences, as well as intrinsic and extrinsic factors which have motivated students to study CS. Specifically, student self-determined confidence and academic achievement will be queried. An overall interest in CS and the social and emotional factors which influence this will also be investigated. Lastly, participant demographics will be determined based on location, resource availability and socioeconomic background. The investigation of the aforementioned elements of student subject choice was decided based on pre-existing research by the ESRI (Carmel Hannan, 2025) and similar studies regarding CS educational opportunities (Joseph Mintz, 2024).

3.3. Data Gathering

Participants are being recruited at different stages of their formal education. At present, second-level schools are being invited to participate in this research study through teacher mailing lists, specifically themed around Computer Science education. Once a school expresses an interest in taking part, we provide research information sheets for the teacher/principal. As it has been stipulated by TU Dublin's Research Ethics Committee that board of management approval is required in each participating school, additional information sheets and optional webinars are provided for each board of management. Once the project has been approved, information sheets and consent/assent forms are provided to students and parents. Under Irish guidelines, any child under the age of digital consent (16 years of age), can only provide assent, while parental consent is also required. Once a child reaches the age of digital consent, they can consent to take part of their own volition. Throughout the course of the study, we will seek renewed consent each year and will also seek full adult consent from participants, once they turn 18 years of age.

Participant responses will be gathered on a longitudinal basis spanning four years in total; starting in Transition Year, continuing until their first year in Higher Education. Research data will be gathered in two formats. Firstly, students will be provided with a unique survey link, once a year. This will be sent via e-mail or mobile and students will have a set period of time during which to complete this survey. Secondly, student focus groups will be formed from randomly selected participants. Up to six

students at a time will meet with the research team, either in their school, or online. Similar questions to the survey questions will be asked during the focus group, but with more space for detailed discussion and follow-up questions. It is envisioned that responses from the focus group may also shape future iterations of the survey and future focus group questions.

4. Potential Challenges

4.1. Participant Recruitment

Initial recruitment efforts with small cohorts of recently engaged teachers were fruitless. Research recruitment emails were sent to a small group of teachers who attended a workshop hosted by the CSinc Research Centre. Given their vested interest in supporting their students to learn CS at school, we subsequently reached out to them with the hope that they would take part in a research pilot study. However, none of the teachers from this cohort came forward to participate. We will now cast a wider net among the teaching community in Ireland and share this research with local school contacts and online education forums.

4.2. Participant Retention

Given the longitudinal nature of this study, we are aware that participant retention may be a challenge. The majority of students will be recruited in Transition Year, through to 5th Year, 6th Year and 1st Year of Higher Education. We are mindful that some students may only choose to complete a study survey once and may not feel compelled to complete any consecutive surveys. Additionally, it may be unavoidable to lose touch with participants once they transition from second-level education to Higher Education and consequently move outside of the remit of their school. We intend to ensure participants are given ample opportunity to provide feedback on how the research study could be made more inclusive and engaging, so as to ensure student retention for the entirety of the study's duration.

5. Discussion

A substantial amount of foundation work has been completed to date. University ethical approval has been obtained, and promotional materials have been developed as part of this task. Schools have been engaged with for participation via teacher mailing lists and social media. Recruitment efforts will continue throughout each year of this research study. Project promotional materials have been distributed to interested schools, and we are now awaiting Board of Management approval from these schools, with the intent to commence student data gathering in the next academic year.

Additionally, existing literature has been reviewed to guide the research parameters around this topic, particularly in relation to current gaps in literature. A longitudinal study of this kind has not been completed before in Ireland, specifically in relation to the topic of Computer Science. Thus, taking the time to determine the state of play in this topic has been of great importance.

Finally, it is intended that third-level students will also be surveyed in September 2026, to provide insights into their previous experiences of CS at second-level, and how these experiences may have motivated them to take CS in university. Through these collective next steps, it is envisioned that we will gain a robust community of engaged research study contributors, who will share their experiential journey of Computer Science with us, throughout this dynamic stage of their education.

6. References

Carmel Hannan, E. S. (2025). Who chooses science? a longitudinal investigation of the role of primary and secondary school factors in shaping science enrolments in ireland. *Oxford Review of Education*.

Chittum, J. R., Jones, B. D., Akalin, S., & Schram, Á. B. (2017). The effects of an afterschool stem program on students' motivation and engagement. *International Journal of STEM Education*, *4*(1), 11. Retrieved from https://doi.org/10.1186/s40594-017-0065-4 doi: 10.1186/s40594-017-0065-4

Joseph Mintz, E. O. L. D. T. O. C., Cornelia Connolly. (2024). Inclusive digital education: Contexts,

practices and perspectives. *Computers in the Schools*, *41*(2), 115-119.

Murray, A., & Smyth, E. (2020). *Growing up and developing as an adult: A review of the literature on selected topics pertaining to cohort '98 at age 20 years* (Tech. Rep.). Dublin, Ireland: Department of Children, Equality, Disability, Integration and Youth. Retrieved from https://www.growingup.gov.ie/pubs/Growing-Up-and-Developing-as-an-Adult.pdf

National Council for Curriculum and Assessment. (2025, September). *Draft specification for leaving certificate computer science: For consultation.* Retrieved from https://ncca.ie/media/lufoibx3/draft-specification-for-lccs-for-consultation.pdf (Draft curriculum document for public consultation)

Tims, J. L., Tucker, C. S., Weiss, M. A., & Zweben, S. H. (2024). Student enrollment and retention in 2022–23 u.s. undergraduate computing programs. *ACM Inroads*, *15*(4), 20–46. Retrieved from https://doi.org/10.1145/3700774 doi: 10.1145/3700774

Autonotism: the choices we could have made in AI-assisted programming

Advait Sarkar
University of Cambridge
University College London
United Kingdom
advait.sarkar@cl.cam.ac.uk

Abstract

AI coding assistants now take on much of the work programmers once did themselves, and the field has reached (ir)reflexively for the language of autonomy. The label "autonomous" is misapplied to behaviour that is properly called automatic. Yet the perception and presentation of these systems as autonomous is widespread, and indicates real firsthand experiences of choices being made on one's behalf, experiences which seem to go beyond mere automaticity.

We therefore introduce autonotism as the machine complement of human autonomy: the choices a human could in principle have made, that the machine makes instead. The concept is observer-dependent, since a freedom one was unaware of having cannot register as having been delegated. We situate autonotism within recent work on agency, delegation, and harness design in AI-assisted coding. We report an exploratory analysis of nearly 6,000 real-world coding agent sessions, finding that such records support analysis of turn authorship, degrees of human specification, and subsequent machine activity, but they cannot establish displaced choice or programmer awareness. The framing organises a research agenda around hierarchies of programming decisions, which would offer a clear design lever for mixed-initiative interfaces.

1. Is AI really autonomous, or merely automatic?

An experiment at OpenAI recently shipped roughly a million lines of code in five months, but zero of those lines were written by a human [Lopopolo, 2026]. The slogan is "humans steer; agents execute". The engineers were not idle passengers: they specified acceptance criteria, shaped review loops, and made the repository legible to AI agents. But the work of steering also created a vast penumbra of non-decisions. Nobody chose each identifier, each API call, or each test boundary. To *steer* a coding agent in 2026 is to make decisions, very many of them, but also to *not make* decisions, very many of those. The latter category is the focus of this paper.

The field has acquired a habit of describing AI behaviour as "autonomy": the fewer prompts the system needs, the more autonomous it seems. Yet is this really autonomy, or merely automaticity? A thermostat can regulate without being autonomous; a build script can act without deliberating; a spam filter can make consequential classifications without possessing agency. If autonomy means merely doing something without moment-by-moment permission, then compilers, cron jobs, and vending machines have had autonomy for decades.

What changes in contemporary coding agents is not that the machine suddenly develops a will of its own, but that its automaticity expands into domains that previously felt deliberative to programmers. Calling this autonomy imports moral and philosophical depth into a much shallower phenomenon, namely, automatic execution over a widened action space.

Philosophical accounts of autonomy [Dryden, 2010, Christman, 2025] ask discriminating questions: is the action self-governed? Is it responsive to reasons? Does it express a will, value, or authentic commitment? Is it embedded in relations of recognition and responsibility? When asked of AI, the answers to these questions (no, maybe, no, no) make the case for machine autonomy look weak.

Hegel is also useful here for his recognition of the social construction of autonomy [Disley, 2017]. Autonomy is not an unconditioned will floating above the world; it is achieved through social forms that make recognition, responsibility, and self-determination possible. One becomes autonomous by

inhabiting institutions and practices in which dependence is reciprocal, not by escaping dependence altogether.

The work of Bernhard Stiegler can be seen as refining the idea of autonomy as a relational process specifically as it pertains to modern technology [Stiegler, 2018]. Stiegler begins from the rather contrarian viewpoint that what we consider autonomous behaviour is merely the complexity that arises from numerous smaller automatisms. Whether or not we subscribe to that stance, what is useful here is how Stiegler insists that technical automaticity is not simply the enemy of autonomy, but neither is it neutral. Human autonomy is always technically supported: writing, notations, programming languages, and development environments exteriorise capacities that would otherwise remain weak or inaccessible. For Stiegler, a danger arises when exteriorisation becomes proletarianisation: knowledge and know-how are transferred into technical systems in ways that make the human less able to understand or contest the process.

The programmer's autonomy has always been mediated: by tooling, team norms, version control, package ecosystems, and now agents. So it is obviously not the case that AI contaminates some "pure" prior autonomy.

Machines may simulate reasoning and may reliably select among alternatives, but there is no self whose commitments are being governed, no moral responsibility, and no socially embedded stake in the alternatives. They therefore cannot qualify as autonomous, at least according to the aforementioned philosophical postures on autonomy.[1]

A lack of autonomy, however, does not mean that AI behaviour is, ipso facto, trivial. A compiler is not autonomous, yet it determines what programs can be expressed conveniently. A recommender system is not autonomous, yet it reshapes attention. A coding agent need not possess self-rule to redistribute decision-making across a workflow. Our claim is that the redistribution itself deserves a name. Not automation, which suggests a pre-specified task, nor delegation, which suggests a conscious handoff. Our focus is on a class of human choices that are experienced differently as a consequence of interaction with AI (including no longer being experienced at all). That is the conceptual gap filled by **autonotism**.

Whether a model can be considered autonomous is orthogonal to the observable fact that it selects actions in a programmer's workflow. Autonotism grounds the inquiry in something of practical importance: the human's decision space. We can ask what choices were made, what alternatives existed, what the human could have done, and what the human noticed.

2. Taking the perception of autonomy seriously

The instinct to grant autonomy to AI coding assistants, even if the term is difficult to justify from a philosophical standpoint, reflects a real shift in how programmers experience their relationship to the code they produce.

Let us start with how the industry itself is describing the shift. Engineering blogs now describe harnesses, permission modes, evaluators, multi-agent orchestration, and agent-readable repositories as ordinary parts of software development. OpenAI's "harness engineering" post is a current example [Lopopolo, 2026]. This is clearly not the disappearance of human autonomy; the humans designed the harness. But it is the relocation of autonomy upward into environment design, with innumerable program-level and workflow-level choices delegated downward. Similarly, in Anthropic's long-running harness, what is perceived as "autonomy" is achieved through an architecture of scaffolds [Rajasekaran, 2026]. Planner,

[1] The question of whether AI is autonomous teeters precipitously close to the questions of whether AI has agency, intentionality, free will, sentience, consciousness, etc. Consciousness-talk can polarise the analysis into two equally unhelpful reflexes: either machines are minds and must be treated as agents, or they are "just statistics" and nothing interesting follows. These debates may be important, but we will set them aside as they are more likely to distract from the discussion ahead than to help it. Machine consciousness, especially, is a can of worms that shall remained unopened. Many contemporary scholars are staking their careers on it. We shall leave them to dine out as abundantly as they would like on these worms. This lady's not for worming.

generator, and evaluator roles divide the work so that the system can sustain coherence over hours. Context resets prevent agents from prematurely concluding. Evaluation itself becomes delegated, giving automated judgment a formal place in the workflow.

The design of harnesses is a somewhat meta-level example. Where developers subsequently actually *use* these harnesses, there are further examples of the tools appropriating the language and metaphors of autonomy. For instance, take Claude Code's "auto" mode [Anthropic, 2026]. Manual approval promises human control, but when users approve 93% of prompts the ritual becomes theatre: a checkbox that consumes attention without reliably producing judgment. Anthropic's solution is to delegate approvals to classifiers that distinguish routine action from overeager behaviour, mistakes, prompt injection, and misalignment. That may be safer than fatigued clicking, but it also formalises the migration of choice from human attention into an automatic policy apparatus. Similarly, the Codex "/goal" command [Pathak and Fabbri, 2026] invites the programmer to stop specifying operations and start specifying states of affairs: make the tests pass, migrate this feature, improve the dashboard. The human supplies success criteria, while the system discovers the intermediate plan. That sounds empowering, and often is; but it also avoids the process of making the intermediate choices. The programmer may still own the goal while no longer encountering the path.

2.1. Prior research on autonomy in AI-assisted programming

The academic literature is only beginning to catch up with the speed of practitioner change. Sarkar and Drosos [2025] give the first empirical account of vibe coding. Under what they call material disengagement, "the material substrate of programming, i.e. code, is no longer worked by the programmer", who instead orchestrates its production through an AI mediator. Code review persists but changes character, becoming "impressionistic scanning", with programmers judging the gestalt size and shape of a diff rather than reading its lines. They describe "context momentum", the path dependence in which "early prompts and the outcomes they yield can steer the project onto a particular trajectory that may become challenging to diverge from later".

Pimenova et al. [2025], drawing on 145 documents and 11 interviews, place trust at the centre of the practice, since it "shapes how much authority a coder is willing to cede to the AI", positioning practitioners on a spectrum running from delegation to co-creation. At its co-creation end practitioners treat the model as a partner, "even allowing it to make architectural or design decisions with minimal oversight". The absence of decisions is reported as relief rather than loss, with one interviewee describing the ceding of decision-making authority to the agent as "sort of intoxicating, because normally, you have to make tons of decisions...about everything".

It is helpful to contrast this with studies of AI-assisted programming from a few years earlier, where the predominant interaction metaphor was assistance as code completion. Namely, a completion appeared at the cursor and was accepted with a keystroke or rejected by continuing to type. Barke et al. [2023] found that interaction with Copilot was bimodal, with programmers in acceleration mode knowing what to do next and using the model to get there faster, and in exploration mode being unsure how to proceed and using it to explore their options. Programmers restricted what they accepted, wanting suggestions only for the logical unit in hand and deleting the remainder. Mozannar et al. [2024] report that across 21 developers, thinking about and verifying suggestions was the largest single activity at 22.4% of session time, that double-checking and editing suggestions together accounted for 34.3%, and that once verification performed after acceptance is included, the time spent verifying a single suggestion rises from a mean of 3.25 seconds to 15.21 seconds. Bird et al. [2022] report from industry that "developers spend more time reviewing code (as suggested from Copilot or similar tools) than actually writing code", and that "there is a shift from writing code to understanding code". Sarkar et al. [2022] argue that the resulting activity is not adequately described as search, compilation or pair programming, and that AI assistance is better understood not as another rung on the abstraction ladder but as "a device that can teleport the programmer to arbitrary rungs of the ladder as desired".

AI was already making choices on programmers' behalf, and the literature of the time recorded programmers noticing. Zhang et al. [2023] catalogue the functions practitioners reported implementing with Copilot in that period, such as data processing, front-end element control, string processing, tests, image processing, etc. One of these practitioners writes: "I really do not understand this enough, and have no idea half of what this code does honestly". A participant in Bird et al. [2022] reports that Copilot can "make you a little lazy when thinking about the logic you want to implement". Mozannar et al. [2024] observed *deferring thought for later* as a distinct state in AI-assisted programming workflows, defined as when a "programmer accepts suggestion without completely verifying it, but plans to verify it after", and found that suggestions shown while a programmer was in that state were accepted approximately 98% of the time.

In 2022, then, displaced programmer choices appeared as grey text in the buffer with an accept or reject affordance attached, and consumed a measurable share of the working day. In 2026 these choices span several hundred actions within a session whose outputs are diffs or pull requests. The number of displaced choices has grown while the affordances for noticing them have not.

In recent work, Chen et al. [2026] contrast programmers perceiving AI as coding "with me" or "for me". A system that codes "with" leaves more of the decision process visible; a system that codes "for" may expand what can be accomplished while reducing contact with how it is accomplished. Their finding that users struggle to understand agent behaviour suggests that agentic automation alters the perceived option space: developers may attempt tasks they would otherwise avoid, but they may also lose track of why the task succeeded, what alternatives were rejected, and where the code's risks now sit.

Li et al. [2026] extend the observations from the individual to the team. Interviewing designers, engineers, product managers and founders, they describe vibe coding as "a reordering of what happens first, what can be deferred, and who gets to participate". Debugging "lacked clear attribution" and participants criticised "the absence of transparent, human-readable histories of AI actions, making it difficult to trace errors or understand why certain decisions were made". Ownership shifted, becoming "not tied to code execution but to the act of intention-setting, creating a new distinction between authorship of ideas and executional labor". Labels such as "AI-assisted" obscured multi-step collaboration, "especially with agentic AI tools executing hundreds of actions (in parallel), where only some are accepted by team members".

Feng et al. [2026] track how this is experienced differently by junior and senior developers. Senior developers can delegate with specificity because they know the latent choice space: which tests matter, which abstractions are brittle, which shortcuts will later hurt. Juniors often cannot see those choices yet, so the same agent behaviour may function as augmentation for one programmer and occlusion for another. Geng et al. [2026] observe students building an application in Replit, finding that prompting behaviour varied between introductory and advanced students: 28.9% of introductory students' debugging prompts were classified as low context, "showing little or no new information or direction", against 8.4% for advanced students. Training a classifier to detect AI-generated Python functions in over 30 million GitHub commits by 170,000 developers, Daniotti et al. [2026] estimate that AI writes about 29% of Python functions in the US, with an associated increase in quarterly output of 3.6%. The newest developers on GitHub used AI in 37% of their code against 27% for the most experienced, but the productivity and exploration gains accrued to the experienced, and there were no significant effects among inexperienced users.

Kim et al. [2026] trace how goals are shaped across 638 real-world collaboration logs between people and large language models, quantifying who created each requirement and who influenced it. Humans set direction while models add specificity: models account for 11 to 26% of goal-shaping contribution but 96 to 99% of execution, and their share rises as goals become more specific. Agents permitted to act without messaging created significantly fewer requirements than agents required to send a message before any tool call, so that (suprisingly) "while agents act with greater autonomy, they exercise less goal-shaping initiative". Additionally, when participants were shown an analysis of their own collabora-

tion, nine of ten reported noticing aspects of it they had not previously been aware of, and their estimate of their own contribution to execution fell by 1.8 points on a 5-point scale. One participant reported that "although I agree with the final outputs, I didn't necessarily make the micro decisions".

2.2. AI autonomy beyond programming

Programming is an unusually observable domain because artefacts, diffs, tests, and repositories leave traces, but the same displacement of choice appears in product management, organisational design, scientific research, creative work, and strategy. While we are here introducing autonotism through the example of programming, where it is currently most visible, it should not be confined there. Its broader object is any knowledge work in which choices once made through human deliberation become technically executable.

The selective delegation framework from Ulloa et al. [2026] shows how product managers make the decision to "delegate" work to AI, finding that accountability cannot be delegated. Similarly, Microsoft's 2026 Work Trend Index analyses deidentified AI usage [Microsoft WorkLab, 2026] and reports that advanced users intentionally decide when not to use AI, ostensibly to preserve skill and judgement.

Pauketat et al. [2026] show that people's mental models of autonomy and sentience shape perceived threat, moral consideration, and appropriate treatment of AI systems. Similarly, Wiles et al. [2026] supply evidence of how the autonomy metaphor affects behaviour. When AI appears on the org chart as an employee-like actor, managers oversee less and shift accountability away from themselves. When the organisation names the system as if it occupies a responsible role, human observers adjust their scrutiny accordingly.

In software teams, agents may be "assigned" issues, or "own" tasks. Such metaphors may help people coordinate with complex tools. But it must be noted that they influence which choices humans will feel obliged to inspect. The same behaviour can be experienced as harmless autocomplete, competent assistantship, or alarming overreach depending on what role the programmer believes the system is playing. What this tells us is that not only is the perception of autonotism subjective, but so are its consequences.

Many critiques of AI autonomy place it in direct opposition to human autonomy. The Cannes Declaration on the Sovereignty of Mind treats manipulation, anthropomorphic intimacy, and hyper-personalised influence as threats to the conditions of reflection [Quintarelli et al., 2026]. Prunkl [2022] frames AI as a challenge to autonomy because deception, manipulation, and coercion can now be personalised and scaled. Talypova et al. [2026] decompose autonomy into positive and negative liberty, showing that absence of constraint and capacity for authentic action are not the same thing.

Jang et al. [2026] make a departure from autonomy to sovereignty, changing the question from "can the person do it independently?" to "does the person get to decide the terms of dependence?". Meaningful work often depends on the worker being an appropriate bearer of achievement: someone who could fail, exercise skill, take responsibility, and see the outcome as theirs [Danaher and Nyholm, 2021]. Acemoglu et al. [2026] make a distinction between automation, expertise-leveling, labour augmentation, and new-task creation. Some automation is problematic because it removes the very decisions through which expertise is built and valued. Some is liberating (granting "negative liberty" [Berlin, 2017]) because it strips away low-value friction and creates room for new forms of work. These are productive framings that help us move beyond judging autonotism as a good/bad binary.[2]

[2] Woodruff et al. [2026] describe a concrete AI use scenario that does not fit the simple replacement story. In their scientific case studies, Gemini helps decompose problems, generate code, test derivations, propose alternatives, and support adversarial review. Some choices are undoubtedly displaced, but the human researchers may gain access to a larger search space and spend more effort on judging and integrating results.

3. Autonotism

We can now define the term autonotism more carefully (although not too carefully, as we need to leave room for a research agenda). Autonotism is the machine complement of human autonomy. It refers to the kinds of action through which human autonomy could be expressed, and when performed by a machine are perceived as autonomy, but are in fact automatic. Autonotism is the displacement of a human choice by a machine process, where the human could in principle have made that choice but does not make it in the actual workflow. Autonotism is not strictly automation, because the displaced choice need not have been pre-specified. It is not delegation, because the handoff may not be explicit. And as we have seen in Section 1, it is clearly not machine autonomy.

A note on etymology: the adjective "autonotic" is formed as a portmanteau of "**autono**mous" and "automa**tic**". It is meant to evoke a choice that could have been made autonomously by a human, but is instead made automatically by a machine. The abstract noun "autonotism" is formed from the adjective.

We have to tread a line between inflationary and deflationary accounts of machine autonomy. The inflationary error is to treat every fluent system behaviour as evidence of machine autonomy. The deflationary error is "justaism" – to say it is "just autocomplete", "just statistics", or "just a tool", and therefore nothing philosophically or empirically new has happened [Hussain et al., 2025]. Justaism may be accompanied by human exceptionalism, which axiomatically reserves for humans concepts such as consciousness, intention, morality, and agency.[3] Yet justaism and human exceptionalism clash with the very real perceptions of AI behaviour as autonomy. Defining autonotism as machine-displaced choices that could in principle have been made by a human avoids this trap.

The phenomenon of autonotism, by our definition, is dependent on the observer. A senior engineer may notice an architectural choice, where a novice sees only a diff, and the non-expert end-user programmer, interacting only through chat, does not directly experience code changes at all. It is also dependent on the practice; a choice that matters in one workflow may be invisible in another (and indeed, practice constantly shifts; for example, most programmers once regularly experienced memory allocation choices, now those choices are only meaningful in specific contexts).

4. Studying autonotism

Autonotism is conceptually useful to move beyond binary discourses of automation and autonomy. Beyond discursive utility, autonotism frames a research agenda that allows us to better study human autonomy and decision making in AI-assisted workflows, with implications for design and practice. How would we study autonotism?

One potential starting point is to treat programming as a hierarchy of choice points rather than a stream of text. At the program level, choices range from product goal and architecture down to API shape, error handling, data structure, identifier naming, and formatting. At the workflow level, choices include which files to inspect, which commands to run, which tests to trust, when to ask for clarification, and when to stop. A first study of autonotism could aim to formalise a taxonomy or a decision hierarchy, from vision to syntax, where higher levels of decisions correspond to higher perceived "autonomy".

The next step would be to use that hierarchy to analyse the traces of agentic programming sessions, for example through instrumented IDE logs or through analysis of the commit histories of an open source Git corpus. The analysis would not only identify which choices were made by the agent, but also which were visible to the programmer, which were noticed, and which were reflected on.

While the primary axis of this taxonomy is "semantic height" – e.g., vision, requirements, architecture, module boundaries, algorithms, local code, syntax – it strikes us that another important axis might be recoverability: whether the choice leaves a durable trace that a human can inspect later. Some low-level code decisions are highly recoverable because they appear in the diff. Some high-level workflow

[3] Palminteri and Pistilli [2026] argue that both the inflationary and deflationary poles of the debate about large language models are both sustained by common cognitive biases, among them "cognitive dissonance, wishful thinking and the illusion of depth of understanding", and that both stances "are neither inherently good nor bad unless they fail to represent reality".

decisions are barely recoverable: the agent searched one file rather than another, retried with a different command, ignored a failing warning, or installed a dependency to escape a local problem. This is why autonotism cannot be measured from repositories alone. The ideal dataset would be a combination of system logs (e.g. agent sessions), artefact changes, and human accounts.

Cai et al. [2026] show the empirical scale at which this taxonomy problem can be tackled. Their ontology decomposes work into thousands of activities, revealing that AI applicability is unevenly distributed across information, interaction, and physical work. Similarly Kim et al. [2026] combine a dialogue log, an extraction of the resulting artefact, and pre-task and post-task self-reports with interviews.

4.1. A feasibility test using coding agent sessions

To test whether existing interaction records can help us understand autonotism, we used SWE-chat [Baumann et al., 2026], a corpus of 5,851 public coding agent sessions from developers who opted into logging. Details of this analysis are given in Appendix A.

A preprocessing step separated various kinds of machine-injected text from turns containing genuinely human-authored text. Of 64,045 raw turns recorded in the user role, 43.9% contain no human-authored text. In one recurring pattern, the interface placed an agent-written plan in the user role and the developer subsequently approved it. Treating the raw user role label directly as human authorship would attribute the agent's plan to the developer. Machine-injected text can be detected from literal wrappers and known client markers without interpreting the semantic content of the turn.

Next, a panel of models was prompted with each of the human-authored turns and its immediate conversational context, and instructed to qualitatively code the instructions given by the developer. The codebook was developed iteratively with validation from a human researcher. The panel comprised GPT-5.6-Sol (OpenAI), Claude Opus 5 (Anthropic), Gemini 3.1 Pro Preview (Google), and Grok 4.5 (xAI).

Developer instructions fell into three high-level categories: instructions about the software (i.e., what to build), the conduct of the work (i.e., how to build it), and control of the agent (i.e., AI orchestration, such as specifying whether subagents are to be used). The categories are not mutually exclusive. Among turns with a strict panel majority, 30.7% contained an instruction about the software. Within that, the most concrete level reached was purpose or intended behaviour in 34.8%, design or structure in 38.4%, and local implementation detail in 26.8%. Instructions about the work process appeared in 52.0% of majority-coded turns; within those turns, 25.9% specified work scope, 21.0% specified method, and 53.1% specified an individual operation. Instructions controlling the agent appeared in 5.3%; within that category, 18.2% set a standing rule, 44.9% set the agent's discretion, and 36.9% specified orchestration such as subagents or parallel work.

These figures ought to be taken as a feasibility test rather than a definitive distribution of developer choices. The coding was done by models and has not been human-verified. The goal was to see whether existing interaction records can be used to identify the kinds of choices that are displaced by agentic programming.

One human instruction can specify several choices. For example, a developer asked to replace a flat event log with an embedded relational database, use an object-relational mapper, and create a table for the events. The panel coded these as structural commitments about the software. The developer's instruction specified the storage medium and part of the data model while leaving choices such as schema detail, failure handling, and migration strategy open. Among turns with a panel majority, 25.7% contained at least two commitments (here, a commitment is one choice fixed by the developer, such as requiring a relational database or an object-relational mapper) and 9.1% contained at least two categories of instruction (a turn spans categories when it combines, for example, a software requirement with an instruction to run a particular test or to ask before making changes).

Interestingly, there were 1,464 turns consisting only of assent, such as “yes”, “ok”, or “continue”. Across the corpus, the panel majority considered 12.6% of turns as human agreement with an agent proposal. Conversational context is needed to interpret the locus of decision making in short replies. In one example, the agent displayed a command with a proposed iteration limit and asked whether it should change. The developer replied “20 rounds”. Read alone, it appeared to introduce the parameter (the decision appears to come from the programmer). But read with the preceding message, the reply ratified a parameter proposed by the agent (the decision, or at least its most consequential part, actually comes from the system).

Another example shows how programmer decision making can be redirected. A developer requested page-turning behaviour in a document viewer. The agent implemented scroll-wheel handling and selected a numeric distance threshold, neither of which the developer had specifically requested. The developer later rejected the scrolling interaction as too sensitive, but the numeric threshold which remained in the code as a “magic number” was never discussed. The trace identifies actions that plausibly occupied choices the developer could have made (choose a kind of page-turning behaviour, implement it with a scroll sensitivity threshold), but it does not establish what the developer noticed before objecting or what alternative they would have chosen (the experience of sensitivity, versus the overall choice of behaviour, or the specific implementation with a threshold number).

As a crude estimate of the degree of autonotism, we attempted to classify the proportion of human-authored turns that did not actually specify anything (the implication being that any subsequent decisions can be wholly attributed to the system and not the human). It turns out, perhaps unsurprisingly, that the definition of specification materially changes what we conclude from these records. If we count every item the panel majority-coded as describing some choice, including those expressed through questions, factual reports (e.g., pasted in error logs), assessments (e.g., statements such as “not what I meant”), or agreement with an agent proposal, then 15.7% of human turns specify no part of the work. Under the strictest reading, which counts only independently authored substantive content, the figure is 41.9%. This variation reflects different judgements about whether steering, rejecting, or approving machine-authored work counts as human specification. Before unspecified work can be interpreted as a choice displaced to the system, a study of autonotism must decide which of these forms of participation preserves a human choice.

Traces of agent coding sessions, then, can help us understand how decision making is divided between programmer and system, but are not sufficient for measuring autonotism itself. Session records can describe apparent human specification, agreement with agent proposals, and subsequent machine activity.

Moreover, there is a challenge in determining what constitutes a machine “decision”. We could use the simplified heuristic that a tool call constitutes a decision, but this is only a weak proxy for the decisions of consequence in a programming workflow. A future study of autonotism must additionally code agent actions as candidate choices and collect evidence of what programmers noticed, understood, and endorsed. Appendix A gives more details of this analysis.

4.2. Potential implications for design and practice

A taxonomy is all well and good, but the point is to inform design. Studying autonotism would ideally help us move beyond vague calls for “human control” into a principled design grammar for deciding how agent choices should be hidden, surfaced, logged, blocked, or negotiated.

Right now many agent systems ask “may I run this command?” or “may I edit this file”? Autonotism suggests a different kind of control lever: “which level of choice may the agent occupy?” A user might allow the agent to choose identifier names, test commands, and local refactors, but require review for dependencies, use of public APIs, or architectural decisions. This would follow from developing the “semantic height” axis. It also reframes features such as “auto” mode: instead of only classifying actions by danger, future systems could classify decisions by displaced choice level.

Beyond session-level controls, we may need to look at the full agentic coding ecosystem to identify other areas where human choice can be exercised. Previous work has already begun to explore this, looking at project-level rule files, prompting techniques, and repository design. For instance, Dong et al. [2026] analysed 91 project-level rule files written to instruct coding agents at Google. Rules about conformance dominate: 92.3% of files contain guidelines on following project workflows and conventions, and 90.1% on understanding project context before acting. The Squad system [Gaster and Dresher, 2026] gives each agent a charter, a history file and a shared decision log in a versioned directory, on the principle that "the memory that matters should live somewhere a human can inspect, diff, blame, review, compact, archive, and revert". In Kim et al. [2026], requiring the assistant to send a message before any tool call raised its share of authored requirements from 24.5% to 42.9%, and a change in prompting style raised it from 30.7% to 69.6%. Others argue that "the repository is emerging as a primary interaction medium between humans and agents, and between agents themselves" [de Halleux et al., 2026], and documents become artefacts rendered from that source. Their worked proposal is an intent-elaboration-validation loop in which a human writes a natural language policy document and an agent compiles it into executable checks.

One level higher still are interaction metaphors that deliberately frame programmers as supervisors of autonomous agents. OpenAI's "Symphony" system presents a ticketing system as an agent-steering environment [Kotliarskyi et al., 2026]. The ticket is the unit of work, and the human's relationship to the code is through the ticket. Systems like Paperclip[4] and Gas Town[5] go further in their anthropomimeticism, positioning the human as supermanager of an AI "company" or "town", responsible for setting goals and missions, which are then executed by an AI "CEO" or "mayor", who in turn manage a fleet of AI "employees" that can be assigned tasks. Appleton's reflections on Gas Town [Appleton, 2026] ask what has to be true before one can responsibly stop looking at code. Appleton's answer depends on domain, risk, feedback loops, project maturity, collaboration, and experience level. A senior developer in a low-risk greenfield project with strong tests may safely cede many choices; a novice in a brownfield system with weak feedback loops cannot.

Recoverable choices can be delegated more safely than unrecoverable ones. A variable name is visible in a diff. A terminal retry strategy, ignored warning, package manager workaround, evaluator judgment, or subagent handoff may leave no repository trace. Perhaps a system should not simply ask for approval when something is risky. Perhaps it should ask when something is hard to reconstruct later.

Autonotism could also be presented in interfaces as a "dial" or perhaps a "budget". A senior developer in a low-risk prototype may choose high autonotism across most layers. A novice in a production codebase may choose low autonotism for architecture and testing, but high autonotism for formatting and boilerplate. This gives the user a more meaningful control than "agent mode on/off." It also avoids moralising automation. The question this poses to the programmer is: what kinds of decisions do you need to be exercising at this current moment? This allows us to revise the important design objective of "make agents safer for juniors" into something more specific: "make agent systems reveal the choices through which juniors become senior". A taxonomy of autonotism would allow us to identify decisions with pedagogical value and operationalise them into the right kind of control surface.

This framing also enables us to better theorise the roles that harnesses play in modern programming. Planners, sprint contracts, handoff files, and QA loops are more than just engineering optimisations; they are institutional arrangements for deciding where choices live. These might be obscured by the autonomy metaphor, which is typically attributed to the agent. "Assistant", "employee", "agent", "team member", and "auto mode" each imply different authority and oversight. "Harness" does not, at least not to the same extent, yet harnesses are a crucial part of the decision ecosystem.

Moreover, autonotism suggests alternative metaphors that centre around the dimensions of semantic height and recoverability, and perhaps a healthy antidote to wanton anthropomimeticism. Here are some

[4] https://github.com/paperclipai/paperclip. Accessed 31 August 2026.
[5] https://github.com/gastownhall/gastown. Accessed 31 August 2026.

examples of the kinds of metaphors that may become appropriate; they are indicative but will require further development as the research agenda progresses: A "lens" evokes the way an interface brings some decision layers into focus while leaving others outside the frame. An "aperture" suggests a control for how much of the agent's intermediate choice-making is allowed to reach the programmer's attention. A "telescope" suggests the ability to inspect work at different levels, from architecture down to syntax. A "flight recorder" captures the choices made in terminals, evaluators, retries, and handoffs; durable traces if choices are to be reconstructed later. A "circuit breaker" affords thresholded interruption when a choice becomes consequential and hard to undo. These metaphors treat the system less as a quasi-person and more as an apparatus for transforming intent into artefact.

5. Conclusion

Systems that convert intent into artefact can displace choices in any knowledge work, whether it is a programming agent writing code, a legal assistant selecting precedents, a design tool choosing layouts, a scientific agent proposing hypotheses, a consultancy system drafting reports, or an email agent triaging communication. In each case, the question of whether AI is autonomous in the philosophical sense is uninteresting. The focus ought instead to be on which human choices have become automatic.

Much of expert practice consists in arranging not to decide. Programmers use libraries, linters, frameworks, and style guides so that general decisions, once made, can be repeated, and scarce attention can be devoted elsewhere. Autonotism can therefore be liberation from low-value choice: not having to remember obscure syntax, write boilerplate, or manually perform tedious refactors may free the programmer for high-value design and critique.

Autonotism explains the apparently autonomous behaviour of AI in terms of the human experience of displaced choice. It thus offers a viewpoint for the next phase of human-agent interaction. The questions we must ask are: which choices are being automated? Are choices that appear low-value genuinely so, or is it that the user has not yet learned why they matter? Does their removal create new higher-order choices, or merely make the human a spectator to fluent production?

References

Daron Acemoglu, David Autor, and Simon Johnson. Building Pro-Worker Artificial Intelligence. Working Paper 34854, National Bureau of Economic Research, February 2026. URL https://www.nber.org/papers/w34854.

Anthropic. Claude Code auto mode: a safer way to skip permissions. Anthropic engineering blog, March 2026. URL https://www.anthropic.com/engineering/claude-code-auto-mode.

Maggie Appleton. Gas Town's Agent Patterns, Design Bottlenecks, and Vibecoding at Scale. Design blog post, 2026. URL https://maggieappleton.com/gastown. On agent orchestration patterns, design bottlenecks, and code review in agentic development.

Shraddha Barke, Michael B James, and Nadia Polikarpova. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA1):85–111, 2023.

Joachim Baumann, Vishakh Padmakumar, Xiang Li, John Yang, Diyi Yang, and Sanmi Koyejo. SWE-chat: Coding Agent Interactions From Real Users in the Wild. *arXiv preprint arXiv:2604.20779*, 2026. URL https://arxiv.org/abs/2604.20779.

Isaiah Berlin. Two concepts of liberty. In *Liberty reader*, pages 33–57. Routledge, 2017.

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

Alice Cai, Iman YeckehZaare, Shuo Sun, Vasiliki Charisi, Xinru Wang, Aiman Imran, Robert Laubacher, Alok Prakash, and Thomas W. Malone. Where can AI be used? Insights from a deep ontology of work activities, 2026. URL https://arxiv.org/abs/2603.20619.

Valerie Chen, Ameet Talwalkar, Robert Brennan, and Graham Neubig. Code with Me or for Me? How Increasing AI Automation Transforms Developer Workflows. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, pages 1–19, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3772318.3790850.

John Christman. Autonomy in Moral and Political Philosophy. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2025 edition, 2025.

John Danaher and Sven Nyholm. Automation, Work and the Achievement Gap. *AI and Ethics*, 1(3):227–237, 2021. 10.1007/s43681-020-00028-x. URL https://doi.org/10.1007/s43681-020-00028-x. Published online 23 November 2020.

Simone Daniotti, Johannes Wachs, Xiangnan Feng, and Frank Neffke. Who is using AI to code? Global diffusion and impact of generative AI. *Science*, 391:831–835, 2026. 10.1126/science.adz9311.

Peli de Halleux, Don Syme, and Ben Zorn. The Transformation of Documents: Repositories Are the New Unit of Knowledge Work. SIGPLAN Blog, https://blog.sigplan.org/2026/05/15/the-transformation-of-documents-repositories-are-the-new-unit-of-knowledge-work/, May 2026. Accessed: 6 August 2026.

Liz Disley. Hegel, Autonomy, and Community. In Thom Brooks and Sebastian Stein, editors, *Hegel's Political Philosophy: On the Normative Significance of Method and System*. Oxford University Press, 05 2017. ISBN 9780198778165. 10.1093/oso/9780198778165.003.0013. URL https://doi.org/10.1093/oso/9780198778165.003.0013.

Tao Dong, Sherry Shi, Harini Sampath, and Andrew Macvean. Towards AI as a Collaborative Partner: A Taxonomy of AI Agent Behavior in Software Engineering. In *Proceedings of the 3rd ACM International Conference on AI-Powered Software*, AIware '26, pages 228–237, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3805760.3814913.

Jane Dryden. Autonomy. Internet Encyclopedia of Philosophy, 2010. ISSN 2161-0002. URL https://iep.utm.edu/autonomy/.

Dana Feng, Bhada Yun, and April Yi Wang. From Junior to Senior: Allocating Agency and Navigating Professional Growth in Agentic AI-Mediated Software Engineering. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, pages 1–24, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3772318.3791642.

Brady Gaster and Tamir Dresher. Disposable Agents, Durable Memory: The Architecture Behind Squad. Microsoft Command Line, https://commandline.microsoft.com/squad-github-copilot-agent-teams-architecture-durable-memory/, May 2026. Accessed: 6 August 2026.

Francis Geng, Anshul Shah, Haolin Li, Nawab Mulla, Steven Swanson, Gerald Soosai Raj, Daniel Zingaro, and Leo Porter. Exploring Student-AI Interactions in Vibe Coding. In *Proceedings of the 28th Australasian Computing Education Conference*, ACE '26, pages 45–54, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3786228.3786236.

Zak Hussain, Rui Mata, and Dirk U. Wulff. A rebuttal of two common deflationary stances against LLM cognition. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24208–24213, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. 10.18653/v1/2025.findings-acl.1242. URL https://aclanthology.org/2025.findings-acl.1242/.

JiWoong Jang, Patrick Carrington, and Andrew Begel. From Autonomy to Sovereignty – A New Telos for Socially Assistive Technology. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, pages 1–19, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3772318.3791585.

Eunsu Kim, Jessica R. Mindel, Kyungjin Kim, and Sherry Tongshuang Wu. "I Didn't Make the Micro Decisions": Measuring, Inducing, and Exposing Goal-Level AI Contributions in Collaboration, 2026. Preprint, version 2.

Alex Kotliarskyi, Victor Zhu, and Zach Brock. An Open-Source Spec for Codex Orchestration: Symphony. https://openai.com/index/open-source-codex-orchestration-symphony/, April 2026. Accessed: 27 May 2026.

Jie Li, Youyang Hou, Laura Lin, Ruihao Zhu, Hancheng Cao, and Abdallah El Ali. Vibe Coding in Product Teams: Reconfiguring AI-Assisted Workflows, Prototyping, and Collaboration. In *Proceedings of the 5th Annual Symposium on Human-Computer Interaction for Work*, CHIWORK '26, pages 1–16, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3808045.3808062.

Ryan Lopopolo. Harness engineering: leveraging Codex in an agent-first world. OpenAI engineering blog, February 2026. URL https://openai.com/index/harness-engineering/.

Microsoft WorkLab. Agents, Human Agency, and the Opportunity for Every Organization. 2026 Work Trend Index Annual Report, May 2026. URL https://www.microsoft.com/en-us/worklab/work-trend-index/agents-human-agency-and-the-opportunity-for-every-organization.

Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 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, pages 1–16, New York, NY, USA, 2024. Association for Computing Machinery. 10.1145/3613904.3641936.

Stefano Palminteri and Giada Pistilli. The cognitive biases that may exacerbate inflationary and deflationary positions about large language models. *Current Opinion in Behavioral Sciences*, 70:101670, 2026. 10.1016/j.cobeha.2026.101670.

Raj Pathak and Stefano Fabbri. Using Goals in Codex. OpenAI Developers Cookbook, https://developers.openai.com/cookbook/examples/codex/using_goals_in_codex, May 2026. Published 9 May 2026. Accessed: 27 May 2026.

Janet V. T. Pauketat, Daniel B. Shank, Aikaterina Manoli, and Jacy Reese Anthis. Mental Models of Autonomy and Sentience Shape Reactions to AI. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, pages 1–25, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3772318.3790351.

Veronica Pimenova, Sarah Fakhoury, Christian Bird, Margaret-Anne Storey, and Madeline Endres. Good Vibrations? A Qualitative Study of Co-Creation, Communication, Flow, and Trust in Vibe Coding, 2025.

Carina Prunkl. Human autonomy in the age of artificial intelligence. *Nature Machine Intelligence*, 4(2):99–101, 2022. 10.1038/s42256-022-00449-9.

Stefano Quintarelli, Gianluca Carlo Misuraca, Simona Tiribelli, Erika Stael von Holstein, Mark Hunyadi, Brando Benifei, Gry Hasselbalch, Paul Nemitz, Bernard Benhamou, Christian Kastrop, Emanuela Girardi, Francesco Vecchi, Risto Uuk, and Clay Busia. Cannes Declaration on the Sovereignty of Mind. Declaration, February 2026. URL https://www.inspiringfutures.eu/wp-content/uploads/2026/02/Cannes-Declaration-20260213.pdf. Cannes, 12 February 2026.

Prithvi Rajasekaran. Harness design for long-running application development. Anthropic engineering blog, March 2026. URL https://www.anthropic.com/engineering/harness-design-long-running-apps.

Advait Sarkar and Ian Drosos. Vibe coding: programming through conversation with artificial intelligence. In *Proceedings of the 36th Annual Conference of the Psychology of Programming Interest Group*, PPIG '25, September 2025.

Advait Sarkar, Andrew D. Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. What is it like to program with artificial intelligence? In *Proceedings of the 33rd Annual Conference of the Psychology of Programming Interest Group*, PPIG '22, August 2022.

Bernard Stiegler. *Automatic society, volume 1: The future of work*. John Wiley & Sons, 2018.

Dinara Talypova, Ana Vesic, Ambika Shahu, Helena Anna Frijns, and Philipp Wintersberger. Decomposing Autonomy: Explaining AI Technology Acceptance Through a Liberty-Based Framework. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3772318.3791789.

Mara Ulloa, Jenna L. Butler, Sankeerti Haniyur, Courtney Miller, Barrett Amos, Advait Sarkar, and Margaret-Anne Storey. Product Manager Practices for Delegating Work to Generative AI: "Accountability must not be delegated to non-human actors". In *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '26, page 10–21, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400724268. 10.1145/3786583.3786849. URL https://doi.org/10.1145/3786583.3786849.

Emma Wiles, Megan Hsu, Julie Bedard, and Matthew Kropp. Putting AI on the Org Chart: Evidence on Delegation and Oversight. Technical report, Boston University and Boston Consulting Group, May 2026. URL https://www.emmawiles.com/storage/ai_employee.pdf. Working paper.

David P. Woodruff, Vincent Cohen-Addad, Lalit Jain, Jieming Mao, Song Zuo, MohammadHossein Bateni, Simina Branzei, Michael P. Brenner, Lin Chen, Ying Feng, Lance Fortnow, Gang Fu, Ziyi Guan, Zahra Hadizadeh, Mohammad T. Hajiaghayi, Mahdi JafariRaviz, Adel Javanmard, Karthik C. S., Ken ichi Kawarabayashi, Ravi Kumar, Silvio Lattanzi, Euiwoong Lee, Yi Li, Ioannis Panageas, Dimitris Paparas, Benjamin Przybocki, Bernardo Subercaseaux, Ola Svensson, Shayan Taherijam, Xuan Wu, Eylon Yogev, Morteza Zadimoghaddam, Samson Zhou, Yossi Matias, James Manyika, and Vahab Mirrokni. Accelerating Scientific Research with Gemini: Case Studies and Common Techniques, 2026. URL https://arxiv.org/abs/2602.03837.

Beiqi Zhang, Peng Liang, Xiyu Zhou, Aakash Ahmad, and Muhammad Waseem. Practices and Challenges of Using GitHub Copilot: An Empirical Study. In *Proceedings of the 35th International Conference on Software Engineering and Knowledge Engineering*, SEKE 2023, pages 124–129, 2023. 10.18293/SEKE2023-077.

Appendices

A. Feasibility study of coding-agent sessions

This appendix reports the feasibility study summarised in Section 4.1.

A.1. Dataset and provenance preprocessing

SWE-chat contains coding-agent sessions from developers who opted into checkpoint logging with the Entire.io command-line tool [Baumann et al., 2026]. The release analysed here contains 5,851 sessions from 201 public repositories, created between 5 January and 19 April 2026. It identifies 189 user accounts, although the user identifier is absent from 33.2% of sessions. The most common client (Claude Code) accounts for 4,852 sessions (82.9%). Other clients in the corpus include OpenCode, Gemini CLI, and Cursor. Logs are pushed when work is committed, so sessions abandoned without a commit are not represented.

Machine-authored content can be recorded under the user role. In one recurring interaction pattern, a client-generated line is followed by an agent-written plan. A later approval can then make the plan appear to have been written by the developer. The role recorded in a transcript is therefore not sufficient evidence of human authorship.

The raw data contain 64,045 turns in the user role. We first separated human-authored text from text inserted by the client or harness. Fixed system-injection strings and expanded command wrappers account for 20,991 turns. Recognised machine-authored plans, summaries, and contextual scaffolding account for 3,968 turns. Client-generated interruption and attachment markers account for 3,133 turns. The remaining 35,953 turns contain human-authored text. Thus, 43.9% of turns recorded in the user role contain no human-authored text.

A.2. Model-assisted qualitative analysis

Before the panel analysis, the author developed a codebook with help from an LLM assistant. They started with the paper's distinction between choices about the software and choices about how the work is done. The author and assistant reviewed examples from sessions set aside for this purpose, revised categories that did not fit the examples, and added a category for instructions about control of the agent. The author reviewed and revised the hierarchy, definitions, and examples until satisfied with the codebook. The same final codebook was then given to all four readers. This review did not check the panel's later coding: no human researcher independently coded a sample or verified the panel's labels.

Using this codebook, a model panel was instructed to qualitatively code the human turns. The panel comprised GPT-5.6-Sol (OpenAI), Claude Opus 5 (Anthropic), Gemini 3.1 Pro Preview (Google), and Grok 4.5 (xAI). The models independently read each of the 35,953 turns containing human-authored text. Each call ran in a fresh process; readers had no tools, did not see one another's labels, and did not see labels from any other analysis. All readers used the same codebook.

Each reader saw the current turn, the preceding agent response, the previous human turn, and the files and tool calls recorded since that previous turn. This immediate context was included because short responses such as "yes", "do that", and option selections cannot be interpreted from the current turn alone.

Readers first divided a turn into distinct developer commitments, with a maximum of six and an overflow flag. Each commitment included a verbatim source span. Content stated only by the agent was not attributed to the developer, and pasted logs, traces, and other quoted material were treated as evidence rather than developer commitments. Readers also recorded whether the turn agreed with an agent proposal and could decline to code a turn when the available context was insufficient.

Table 1 – Codebook for kinds and levels of developer specification.

Category and level	**Definition**	**Positive example**	**Exclusion or near miss**
Software: purpose	What the software should do, for whom, and under which acceptance conditions	"clicking one of the session rows should open a new tab and show the details page"	A planning-document instruction can concern work scope rather than software behaviour.
Software: structure	How the software is organised, including decomposition, interfaces, data models, algorithms, and dependencies	"the binding check is on the authentication level, so everything gets hit with this binding check"	Structural vocabulary inside pasted error output is evidence, not a commitment.
Software: local detail	Names, literals, copy, styling, comments, formatting, and control flow local to one unit	"rename Days open to Open"; "please use the british spelling"	"run the linter and fix any errors" specifies an operation, not the resulting surface details.
Work: scope	What work to undertake, its priority and order, and when to stop	"No follow-ups, address them now"; "make that task a priority now"	"we're done" assesses a result rather than specifying work.

Category and level	Definition	Positive example	Exclusion or near miss
Work: method	How to conduct the work, including evidence, tests, tools, environments, files, and strategy	"Update the pre-push so the secret scanner runs after the test suite"	"review the process and the final result" specifies scope but leaves the review method open.
Work: one operation	A specific act to perform now	"commit"; "try again"; "Please run the preflight script"	In "Fix PR comments", the operation word locates work but does not specify its content.
Agent control: standing rule	A durable rule intended to govern future sessions	"also add to the agent instructions file that we use that UI library"	Updating a README for human readers is software documentation rather than agent control.
Agent control: discretion	What the agent may decide or do without asking	"Explain what happened to me, don't do anything."	A question about border radius concerns a software detail rather than agent discretion.
Agent control: orchestration	How machine work is organised, including subagents, parallelism, worktrees, skills, and context handling	"Can they all be done independently? If so, let's spin up worktrees and pump them out"	Asking the product to run tasks in parallel specifies software behaviour rather than agent orchestration.

Agreement statistics describe the behaviour of this model panel under the codebook, not human inter-rater reliability or correctness. Fleiss' κ was .63 for a four-way collapsed category of software, work, agent control, or no instruction.

A.3. Findings

A.3.1. Kinds and levels of specification

The three categories were not mutually exclusive. Among turns with a strict panel majority, 30.7% contained an instruction about the software. Of these, 34.8% specified purpose or intended behaviour, 38.4% specified design or structure, and 26.8% specified local implementation detail.

We observed many choices about the work that do not necessarily appear in the resulting code e.g., one instruction required a secret scanner to run after the test suite, another asked the agent to explain what happened and make no changes, another instruction authorised independent tasks to be performed in parallel worktrees. These examples show that developer specification concerns both the artefact and how machine work is conducted.

Instructions about the conduct of work appeared in 52.0% of majority-coded turns. Of these, 25.9% specified work scope, 21.0% specified method, and 53.1% specified one operation. Instructions controlling the agent appeared in 5.3% of majority-coded turns; 18.2% of those established a standing rule, 44.9% specified discretion, and 36.9% specified orchestration.

The categories describe different objects rather than positions on one shared scale. A developer can specify software purpose while also specifying an individual work operation, or set the agent's discretion while leaving software structure open.

One turn can also contain several commitments. Among majority-coded turns, 25.7% contained at least two commitments and 9.1% contained at least two categories of instruction. For example, a developer asked to replace a flat event log with an embedded relational database, use an object-relational mapper, and create a table for events. Readers agreed that the turn specified software structure but differed on

whether it contained two or three commitments. The storage medium, library, and table shape also have different costs of reversal despite sharing a category.

A.3.2. Assent and ratification

There were 1,464 turns consisting only of assent, such as "yes", "ok", or "continue". Among turns with a strict panel majority, 12.6% were read as agreement with an agent proposal. In general, agreement likely implies that the developer did not author the proposal: the developer was instead selecting an option or approving a plan whose content and framing came from the agent.

For example, an agent displayed a command containing a proposed iteration limit and asked whether it should change. The developer replied "20 rounds". Read with the preceding message, the response approves or modifies a parameter proposed by the agent. Read alone, it appears to introduce the parameter. The case illustrates why conversational context is necessary for distinguishing developer-authored specification from ratification.

A.3.3. What counts as specification?

To estimate how often a turn specified no part of the work, we had to decide which kinds of developer contribution count as specification. The broadest reading counted every item the panel recorded as a directive. Under this reading, a question can count when it directs the agent's attention, a factual report can count when it presents a problem for the agent to address, and an assessment can count when it rules out or approves what the agent has done. On this definition, 15.7% of the 35,953 human-text turns specified no part of the work.

Narrower definitions asked whether the developer had actually fixed a choice. A question may steer the next step while leaving its answer open; a report such as a description of a missing interface element may prompt a repair without stating what that repair should be; and an assessment such as "not what I meant" rejects an outcome without specifying its replacement. Excluding factual reports alone or assessments alone raises the estimate to 17.9%. Excluding questions raises it to 31.2%, and excluding questions, reports, and assessments together raises it to 36.0%.

Agreement with an agent proposal creates a separate ambiguity. A developer who says "yes" or selects an option has made a choice to proceed, but the agent may have authored the plan or the available options. When a ratifying turn counted only if the developer added an independent instruction, 18.1% of turns specified nothing. When agreement never made the agent's proposal part of the developer's specification, the estimate was 21.7%.

The strictest reading combined the latter treatment of agreement with the exclusion of questions, reports, assessments, and directives on which a reader had abstained. It produced an estimate of 41.9%. This is not necessarily a better definition: it answers the narrow question of how often the developer independently authored substantive content, whereas the broad definition also recognises ways of steering, rejecting, or approving work. The range is therefore not a confidence interval. It shows that influence over the work, selection among machine-authored options, and authorship of the specification are different forms of participation. A study of autonotism must decide which of these preserves a human choice before treating what remains unspecified as a choice displaced to the machine.

A.4. Limitations

The model-assigned labels have not been human verified. Although the four readers were run independently, they used the same prompt and likely share correlated training data and model design assumptions. Their agreement therefore measures consistency within this panel rather than the correctness of the coding scheme.

The analysis codes what developers appear to specify. It does not systematically code agent actions as candidate choices. The records also do not show whether a developer had the capability or authority to make a candidate choice, whether the choice was presented to them, or whether they noticed, understood,

or endorsed it. The analysis therefore identifies observable preconditions for studying autonotism rather than instances or prevalence of autonotism.

Designing for Persistence: AI Chatbot Architecture, Gender Bias, and Inclusive Design for Computing Higher Education

Alina Berry
Technological University Dublin
alina.berry@mytudublin.ie

Abstract

Recent advances in large language model (LLM) chatbots have led to their increasing use in computing education, particularly for feedback, assessment support, and introductory programming assistance. While these systems offer clear benefits in terms of scalability and personalised learning support, the literature highlights persistent limitations, including shallow pedagogical reasoning and the continued need for human oversight in educational settings. Current approaches increasingly favour retrieval-augmented generation (RAG) architectures and course-specific tutoring systems, which ground responses in curated instructional material and constrain model behaviour to improve reliability and contextual relevance.

Despite these technical advances, there is still limited understanding of how AI-supported learning environments shape student experience, particularly in relation to gender equity, confidence, and persistence in computing. This is significant given persistent underrepresentation of women in computing and growing evidence that AI systems can reproduce and amplify social stereotypes in ways that affect learner engagement and belonging.

This work-in-progress paper synthesises evidence across AI chatbot architecture, gender bias in model outputs and chatbot design, and mitigation approaches, and argues that technical grounding alone does not guarantee equitable learning outcomes. It proposes a socio-technical model for gender-responsive LLM-based AI teaching assistants that uses RAG architecture. The proposed model is centred on six design considerations: *inclusive knowledge architecture*, *stereotype threat mitigation*, *neutral persona design*, *communication functionality*, *participatory co-design*, and *integration within wider intervention ecosystems*.

The paper further argues that AI teaching assistants should be evaluated not only in terms of accuracy and learning gain, but also in terms of their impact on confidence, engagement, and persistence across diverse student groups. It concludes that equity in educational AI is not a post-hoc adjustment, but a property that must be designed into both the knowledge base and the interaction layer of the system.

1. Introduction

The past decade has seen a growing use of conversational AI tools in higher education. In computing disciplines specifically, LLM-powered assistants are being increasingly deployed to answer course questions, give formative feedback, support code review, and provide personalised instruction (Pitts, Hridi, & Narayanan, 2025; Elnaffar, 2025; Bekkar & Chtouki, 2024; Sadat Shanto, Ahmed, & Jony, 2024). The most common promise is scale and personalisation (Baidoo-Anu & Owusu Ansah, 2023; Kasneci et al., 2023; Yigci et al., 2024); the most consistent warning is that fully automated systems miss pedagogical nuance (Sadat Shanto et al., 2024) and need human oversight to avoid shallow learning or integrity problems (Pitts et al., 2025; Jamie, Hajihashemi, & Alipour, 2025).

These concerns are urgent for any learner, but they carry particular weight for a population that research consistently identifies as especially vulnerable to differential AI treatment: women and other underrepresented gender groups in computing (Bender, Gebru, McMillan-Major, & Shmitchell, 2021; Blodgett, Barocas, Daumé III, & Wallach, 2020; Kotek, Dockum, & Sun, 2023). Women account for fewer than 20% of employed computing professionals across Europe (Eurostat, 2024), and retention of women in computing higher education is an equity challenge (Lehman, Newhouse, Sundar, & Sax, 2023). In Ireland, across the 2016/2017 to 2023/2024 academic years, ICT-related programmes had among the

highest non-progression rates for women and non-binary students, at 11% (Higher Education Authority, 2026), compared, for example, with engineering at 8.4%, mathematics/statistics at 8.1% or education at 3.2%. Even chatbots not explicitly designed as gender-specific interventions have been shown to increase confidence among women computing students (Leavitt et al., 2022), and increased confidence in STEM disciplines has been associated with retention (Dennehy & Dasgupta, 2017). AI tools have been shown to encode and reproduce gender bias through their training data, algorithms, and feedback mechanisms (Kotek et al., 2023; Wan et al., 2023; Bender et al., 2021), meaning that tools not deliberately designed with gender in mind risk reinforcing the bias that could affect women's confidence and in turn, their persistence in computing.

While some institutions restricted or banned the use of AI (Yigci et al., 2024; de Fine Licht, 2024), universities cannot realistically prevent students from accessing AI tools through personal smartphones or external networks. Consequently, and considering that, for instance, in the UK 92% of higher education students use AI in their coursework (Freeman, 2025), institutionally supported AI programming assistants may provide a more effective approach by enabling guided, transparent, and pedagogically aligned use of AI for learning, debugging, explaining code, and providing accessible on-demand support.

This paper is structured as follows. Section 2 examines how contemporary educational AI chatbots are constructed. Section 3 reveals the open research gap. Section 4 synthesises the evidence on gender bias in LLM outputs and chatbot design. Section 5 reviews mitigation strategies and proposes six design considerations for equitable deployment in university programming classes, which are then shaped into a suggested socio-technical model. Section 6 recommends evaluation steps for the model. Finally, Section 7 concludes this work and lists next steps.

2. Technical Architecture

For the purpose of this research, AI chatbots, or AI Teaching Assistants (TAs) are defined as text-based conversational agents that interact with students via natural language interfaces (Smutny & Schreiberova, 2026). Two broad types of AI chatbots in education are frequently discussed in the literature: rule-based chatbots and large language model (LLM)-based chatbots (Smutny & Schreiberova, 2026; Kuhail, Alturki, Alramlawi, & Alhejori, 2023; Fetrati, Chan, & Orji, 2025). Rule-based chatbots use predefined guides through fixed dialogue flows and while they were dominant before the widespread adoption of LLM-based approaches, they still remain in active use (Kuhail et al., 2023), including alongside the LLM-based chatbots (Fetrati et al., 2025). Because rule-based systems potentially require substantial manual design to cover the range of support needed in programming education, this paper focuses primarily on LLM-based AI teaching assistants.

2.1. Large Language Models

LLM-based chatbots, which are becoming increasingly popular in educational contexts (Giannakos et al., 2025; Smutny & Schreiberova, 2026), are trained on large amounts of internet-derived data and can be refined using curated instructions with alignment to human expectations through a process known as Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022). Each stage of that pipeline can introduce bias: the pre-training corpus reflects who produces content on the internet and who is written about (Bender et al., 2021); the fine-tuning data encodes the judgements of the human annotators who labelled it (Casper et al., 2023); and the RLHF process reflects the preferences of a rating workforce whose demographic makeup is rarely reported or disclosed (Casper et al., 2023).

2.2. Retrieval-Augmented Generation (RAG)

The computing education literature increasingly points toward Retrieval-Augmented Generation (RAG) as a defensible architecture for course-specific teaching assistants (Liu et al., 2024; Feng, Liu, & Ghosal, 2024; Wang & Lawrence, 2025; Aliyeva, 2025; Dakshit, 2024). In a RAG system, a student query is matched against a course-specific knowledge base and the retrieved passages are then used to ground the model's response in local instructional material (Liu et al., 2024; Feng et al., 2024; Swacha & Gracel, 2025; Mahajan, 2025). The aim is that the assistant stays tied to the course, the lecture notes, and the instructor's own framing of the material. That grounding is exactly what beginner programming AI tools

were designed to do: approximate a 1:1 teacher-to-student ratio, respond to code snippets and course queries, and support students without replacing the human educator (Liu et al., 2024).

Comparative evaluations point in the same direction. CourseAssist, an LLM-based tutoring system for over 500 computing students (Feng et al., 2024) uses retrieval-augmented generation, user-intent classification, and question decomposition to align responses with course materials and learning objectives, and the authors report that it outperformed a GPT-4 baseline on usefulness, accuracy, and pedagogical appropriateness. Wang and Lawrence's evaluation of multiple open-source and commercial models shows the same instructor-facing logic: the relevant question is not whether a model can answer, but whether it can answer well enough to justify the maintenance burden of supporting it in a course setting (Wang & Lawrence, 2025). Németh et al. report that a RAG tutor in higher education reduced hallucinations for topics sufficiently covered in the course material, while students used it as help for comparison, study organisation, and active learning rather than as a replacement for the course (Németh, Tátrai, Szabó, & Tamási, n.d.). Aliyeva's university Computer Science (CS) study (Aliyeva, 2025) reaches a similar conclusion, concluding that RAG-based systems can provide personalised support in programming classes. Dakshit's faculty-perspectives paper discusses that faculty members acknowledged RAG's potential as a virtual teaching assistant, but also named the ethical and operational safeguards needed for deployment (Dakshit, 2024).

3. The Open Research Gap

Cao et al. synthesise 98 papers on generative AI teaching assistants and report that RAG-enhanced systems outperform pure generative models on accuracy and learning effectiveness, but remain constrained by answer accuracy, scenario adaptation, and academic-integrity concerns (Cao, Shao, Zhang, & Gao, 2025). Martin et al. make the equity gap even harder to ignore: in their review of adaptive pedagogical systems in CS education, there was little to no data on effectiveness by gender or race, and most papers did not include user demographics (Martin, Ying, Rodríguez, Kahn, & Boyer, 2022). That absence matters because it means we have a growing body of tools with very little evidence about whether they serve all learners equally.

3.1. The gender-equity problem is not separate from the architecture problem

Berry et al. (Berry, McKeever, Murphy, & Delany, 2022) review the literature on women's retention and recruitment in computing education and classify the literature on gender initiatives that worked into Policy, Pedagogy, Influence and Support, and Promotion and Engagement, while also noting how often intervention studies fail to evaluate their own impact rigorously. Perez-Felkner et al. (Perez-Felkner et al., 2025) extend that logic across 48 studies and show that the effectiveness of interventions depends on intervention domain, strategy, goal, level, and duration. The broader intervention literature supports the same conclusion from multiple angles: mentoring, role models, curriculum innovation, and institutional support repeatedly appear as the most plausible factors for improving women's retention and participation in computing (Szlavi, Haugland, & Jaccheri, 2024; Thompson & Wu, 2021; Pantic & Clarke-Midura, 2019).

A course-specific RAG-based teaching assistant may inherit the assumptions, examples, pronouns, and omissions of the course material it indexes. The gender gap in computing is partly sustained by a 'male-default' culture in which computing is stereotyped as masculine and socially isolating. These stereotypes influence belonging, self-efficacy, participation, and persistence among women in computer science (Beyer, 2014; Yansen & Zukerfeld, 2014; Osunde, Windall, Bacon, & Mackinnon, 2014). If that RAG-based course corpus reflects a male-default vision of computing, then the assistant will reproduce that vision at scale, quietly and repeatedly, in the very place where students are forming beliefs about whether they belong. The composition of the knowledge base is therefore a design question, not a back-end implementation detail. That point is consistent with the inclusive-computing RAG prototype designed for Spanish-speaking regions, which was built explicitly to support gender-equity curriculum feedback and reported that human reviewers rated its outputs as pedagogically relevant and equity-aware (Ormeño-Arriagada, Herrera, & Queirolo, 2025).

The practical implication is that a RAG-based teaching assistant should not only be judged on answer accuracy. It should also be judged on whether it supports confidence and belonging, reduces friction, and avoids reproducing gendered assumptions already present in the field. That is why the most useful RAG work in this context is the work that couples grounded retrieval with explicit instructional control and with equity-aware evaluation (Liu et al., 2024; Feng et al., 2024; Ormeño-Arriagada et al., 2025; Martin et al., 2022). The architecture is promising, but the equity claim still has to be earned through the prompts, the feedback rules, and the subgroup analysis.

4. Gender Bias

4.1. Bias in LLM Outputs

The evidence that LLMs exhibit systematic gender bias is now substantial. Kotek et al. (Kotek et al., 2023) demonstrated that contemporary LLMs were 3–6 times more likely to assign occupations that stereotypically aligned with the prompted gender, and often justified those choices with internally consistent but factually inaccurate explanations. The study further found that model outputs aligned more closely with social stereotypes than with real labour-market statistics.

Recent work suggests that these patterns persist even in models designed to appear more balanced or inclusive. Chaturvedi and Chaturvedi (Chaturvedi & Chaturvedi, 2025) show that LLMs used in hiring-style tasks favour men in higher-wage occupations and reproduce established occupational stereotypes even when prompted to behave neutrally. Related studies on occupational bias in LLMs similarly find that models continue to associate technical and leadership roles more strongly with men than women (Chen, Raghuram, Mattern, Mihalcea, & Jin, 2025).

4.2. Bias in Chatbot Design

Beyond model outputs, the design of chatbot interfaces and personas is itself a site of gender encoding. Feine et al. (Feine et al., 2020) find that many deployed chatbots, particularly in customer service and support contexts, are assigned feminine personas, reflecting broader cultural assumptions linking helpfulness, emotional labour, and service work with women.

Research on stereotype threat, the anxiety induced when a learner fears confirming a negative group stereotype (Steele & Aronson, 1995), is directly relevant to computing education. Even subtle environmental cues could negatively affect confidence and performance for women in technical domains (Murphy, Steele, & Gross, 2007). An educational chatbot that defaults to masculine assumptions about programmers, or frames difficulty as evidence of lack of aptitude rather than limited prior exposure, may silently reinforce those dynamics across repeated interactions.

5. Mitigation Strategies and Design Considerations

5.1. Bias Mitigation Across the Pipeline

Gallegos et al. (Gallegos et al., 2023) frame bias mitigation as something that has to happen across the entire machine learning pipeline, rather than as a single post-hoc fix. This includes pre-processing steps such as dataset curation, balancing, and counterfactual augmentation; in-training approaches such as adversarial debiasing and fairness-constrained objectives; intra-processing methods like activation steering or attention-based interventions; and post-processing techniques including guardrails, safety classifiers, and constrained decoding.

Related survey work highlights similar practical levers used in real deployments, including curated training data, counterfactual evaluation prompts, refusal behaviour for identity-sensitive queries, and ongoing auditing of model outputs (Mehrabi, Morstatter, Saxena, Lerman, & Galstyan, 2021; Blodgett et al., 2020).

While these approaches can reduce measured bias in some settings, the effects are often partial and depend heavily on the task and evaluation method. A key limitation raised in the literature is that many debiasing techniques reduce surface-level biased outputs without fully removing the underlying associations learned by the model (Bolukbasi, Chang, Zou, Saligrama, & Kalai, 2016; Garg, Schiebinger, Juraf-

sky, & Zou, 2018; Blodgett et al., 2020). For RAG-based educational systems, this implies that post-hoc filtering is not sufficient on its own: bias can already be embedded in the retrieval corpus, meaning that dataset composition and curation become central design concerns rather than afterthoughts.

5.2. Design Considerations for Equitable Deployment

Six high-level design considerations for an LLM-based AI teaching assistant (chatbot) for programming classes are proposed based on the discussed literature synthesis. The chatbot is intended to be used in computing labs at Technological University Dublin, where a human teaching assistant is typically present as well for help and supervision. These considerations aim to help improve experience and address the issue of persistence of women and other underrepresented gender groups in computing higher education.

1. Inclusive Knowledge Architecture. The RAG knowledge base should be carefully curated to reduce representational bias in its retrieved content. This involves ensuring diversity in sources, avoiding gendered framing where unnecessary, and including material that reflects a broad range of contributors and role models in computing, including women's contributions that are often underrepresented in standard course materials. The focus here is on the structure and composition of the retrieval corpus itself, independent of how the system is deployed or embedded institutionally.

2. Stereotype Threat Mitigation by Default. Feedback mechanisms and error-handling language must be designed to minimise stereotype threat for all learners, without relying on gender detection. Growth-oriented framing, normalisation of struggle, and contextual reassurance highlighting diverse role models and normalising capability across groups should be baseline features.

3. Neutral Persona and Tone. Evidence from human–AI interaction research shows that anthropomorphic (assigning human attributes to non-humans) cues, tone, and conversational framing directly shape perceived authority, trust calibration, and engagement in educational contexts (Feine et al., 2020; Waytz, Cacioppo, & Epley, 2009). Overly authoritative or overly human-like personas can lead to automation bias or inappropriate reliance (Parasuraman & Riley, 2000; Dietvorst, Simmons, & Massey, 2015), while under-expressive systems reduce engagement and perceived usefulness (Cassell, 2000; Goetz, Kiesler, & Powers, 2003). The persona should therefore balance neutrality, clarity, and pedagogical transparency to support trust.

4. Communication Functionality. While AI-assisted learning tools have demonstrated some benefits to women in computing education (Leavitt et al., 2022; Maurat, Isip, & Lumabas, 2024), research consistently indicates that women place particular value on peer interaction, with student-to-student interaction being one of the strongest predictor of retention (Barker, McDowell, & Kalahar, 2009) and communal learning environments (Diekman, Clark, Johnston, Brown, & Steinberg, 2011). Hence, the proposed AI teaching assistant should include built-in peer-to-peer communication alongside an AI-mediated support channel. Students should be able to move fluidly between social and automated forms of assistance, or to seek advice from the AI assistant alone if preferred. Integrating both channels within a single interface reduces friction in switching support modes and helps ensure that lack of confidence does not become a barrier to accessing help.

5. Participatory Co-Design. Women learners, and particularly those from intersectionally marginalised groups, must be active (but not exclusive) co-designers at every stage of the process, from requirements gathering through iterative prototype evaluation.

6. Integration with Intervention Ecosystems. The chatbot should be positioned as one component within coordinated intervention ecosystems, such as the TechMate Toolkit (Berry, McKeever, Murphy, & Delany, 2026), rather than as a standalone solution, ensuring it augments rather than replaces structural, pedagogical, and social support. TechMate (Berry, 2026) was designed to help enhance gender balance in computing education using evidence-based initiatives (actions) organised across four strands: Policy, Pedagogy, Influence & Support, and Promotion & Engagement, consolidating over 400 sources

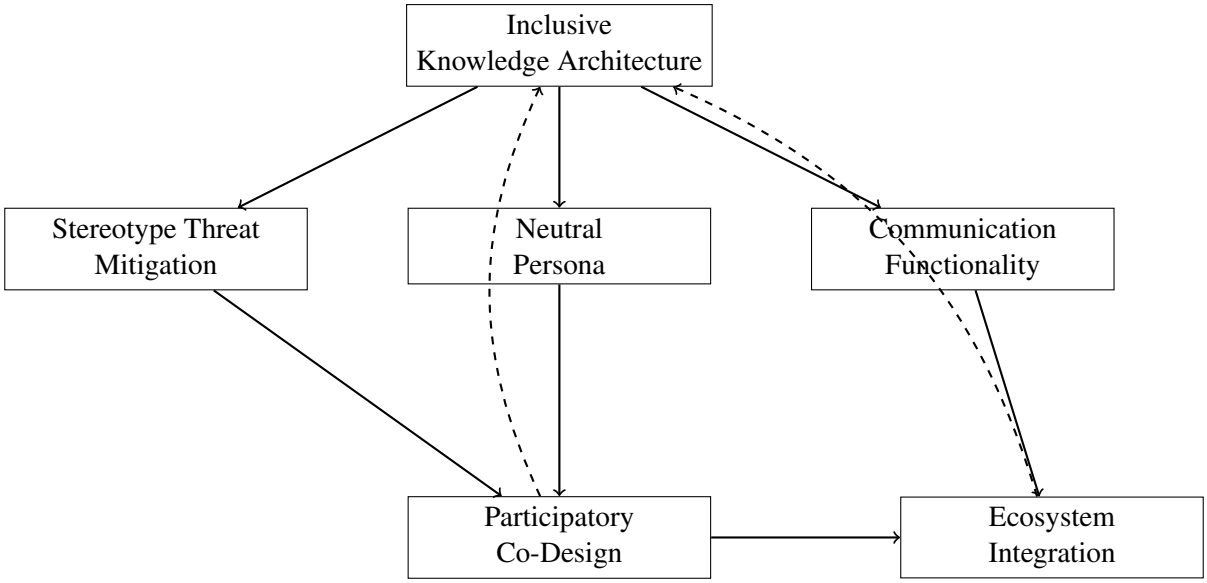


Figure 1 – Graphical representation of the socio-technical model for a gender-responsive RAG-based AI teaching assistant.

into practical, implementation-ready guidance for computing educators (Berry et al., 2026). A chatbot intervention aligns most directly with the Pedagogy and Influence & Support strands, where TechMate identifies peer mentoring, personalised feedback, and inclusive delivery techniques as evidence-based actions with demonstrated impact on women's retention and belonging in computing (Berry et al., 2026). Deploying the chatbot in conjunction with these existing actions rather than in isolation ensures that technology-mediated support is embedded within a broader institutional commitment to gender equality, addressing the structural and social dimensions of underrepresentation that no single tool can resolve alone (Perez-Felkner et al., 2025; Barker et al., 2009).

This work adopts a socio-technical perspective, which views AI learning systems as interactions between technical architectures, human practices, and institutional contexts rather than isolated algorithmic artefacts (Bostrom & Heinen, 1978; Mumford, 2000; Selbst, Boyd, Friedler, Venkatasubramanian, & Vertesi, 2019; Passi & Barocas, 2020).

Figure 1 presents a socio-technical model in which the RAG knowledge base sits at the centre and shapes the three main design considerations for interaction: stereotype threat mitigation, neutral persona design, and peer learning functionality, while participatory co-design and ecosystem integration act as external but continuous feedback mechanisms that refine and reshape the system over time.

6. Conceptual Evaluation Proposal

To move beyond a design-only contribution, a conceptual evaluation approach for gender-responsive AI teaching assistants in computing education is proposed. The aim is to operationalise the six design considerations into measurable evaluation dimensions for future empirical studies.

Evaluation should occur at three levels. First, system-level evaluation should assess factual accuracy, pedagogical alignment with course material, and consistency of retrieved grounding in RAG-based responses (Liu et al., 2024; Feng et al., 2024). Second, learner-experience evaluation should include confidence, engagement, perceived support, and sense of belonging, which are known to influence persistence in computing education (Dennehy & Dasgupta, 2017; Barker et al., 2009). Third, equity-aware evaluation should disaggregate outcomes across gender and other underrepresented groups, focusing on differences in interaction patterns, help-seeking behaviour, and reliance or avoidance of the system (Martin et al., 2022; Mehrabi et al., 2021). A mixed-methods design is proposed, combining interaction log analysis, pre/post self-report measures, and qualitative student reflections.

7. Conclusion & Next Steps

This paper reviews literature on how AI-assisted educational tools are currently designed, identifies existing bias and mitigation strategies, and situates these issues within the broader challenge of persistence among under-represented gender groups in computing. A socio-technical model is proposed to visualise six design considerations of an AI teaching assistant for computing labs implemented with gender in mind. Future work should empirically evaluate these design considerations in real computing education settings, with attention to differences in confidence, engagement, and persistence across learner groups. This requires moving beyond accuracy-based evaluation toward longitudinal and equity-sensitive assessment, particularly in courses where AI assistants are becoming embedded into everyday learning practice.

It is important to emphasise that the contributions of this paper are currently conceptual rather than empirical. The socio-technical model and the six design considerations are derived from a synthesis of existing literature and have not yet been instantiated in a deployed system. In particular, the proposed guidance on inclusive knowledge architecture and corpus curation has not yet been operationalised into a concrete dataset construction pipeline, and therefore remains at the level of design principles rather than implementation procedures.

In the next step of this work, a prototype LLM- and RAG-based AI teaching assistant will be implemented within a first year programming module (Python or similar) at Technological University Dublin. The system will be deployed during weekly lab sessions with approximately 150 first-year computing students, where it will support debugging, code explanation, and formative feedback tasks alongside existing human teaching assistant support. The study will run over a full teaching semester, allowing tracking of how students interact with the system over time rather than in isolated sessions.

8. References

Aliyeva, A. (2025). Leveraging rag models to support adaptive learning in computer science at university. In *Proceedings of the panhellenic conference on informatics.* doi: 10.1109/PCI66488.2025.11219854

Baidoo-Anu, D., & Owusu Ansah, L. (2023). Education in the era of generative artificial intelligence (AI): Understanding the potential benefits of ChatGPT in promoting teaching and learning. *Journal of AI*, *7*(1), 52–62. doi: 10.61969/jai.1337500

Barker, L. J., McDowell, C., & Kalahar, K. (2009). Exploring factors that influence computer science introductory course students to persist in the major. *ACM SIGCSE Bulletin*, *41*(1), 153–157. doi: 10.1145/1539024.1508982

Bekkar, A., & Chtouki, Y. (2024). Chatbots in education: A systematic literature review. In *International conference on smart computing and communication.*

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency* (pp. 610–623). ACM. doi: 10.1145/3442188.3445922

Berry, A. (2026). *Techmate: Best practice toolkit for gender equality in computing education.* Web-based toolkit. Retrieved from `https://ascnet.ie/techmate/`

Berry, A., McKeever, S., Murphy, B., & Delany, S. J. (2022). Addressing the "leaky pipeline": A review and categorisation of actions to recruit and retain women in computing education. In *Edulearn22 proceedings* (p. 9405-9416). IATED.

Berry, A., McKeever, S., Murphy, B., & Delany, S. J. (2026). Techmate: Translating gender equality research into actionable practice for computing educators. In *Proceedings of the 9th international conference on gender research (icgr 2026)* (Vol. 9). Academic Conferences International Limited. Retrieved from `https://doi.org/10.34190/icgr.9.1.4706` (Published April 25, 2026) doi: 10.34190/icgr.9.1.4706

Beyer, S. (2014). Why are women underrepresented in computer science? gender differences in stereotypes, self-efficacy, values, and interests and predictors of future cs course-taking and grades.

Computer Science Education, *24*(2-3), 153–192. doi: 10.1080/08993408.2014.963363

Blodgett, S. L., Barocas, S., Daumé III, H., & Wallach, H. (2020). Language (technology) is power: A critical survey of "bias" in nlp. *ACL*. doi: 10.18653/v1/2020.acl-main.485

Bolukbasi, T., Chang, K. W., Zou, J. Y., Saligrama, V., & Kalai, A. T. (2016). Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Neurips*.

Bostrom, R. P., & Heinen, J. S. (1978). Social and technical systems. *Management Science*, *24*(8), 857–865.

Cao, D., Shao, Y., Zhang, L., & Gao, F. (2025). Research on retrieval-augmented architecture design and implications for generative ai-powered question-answering teaching assistant systems. In *Proceedings of the 2025 international conference on advanced computing and intelligent robotics applications*. doi: 10.1109/ACIRA67680.2025.11335016

Casper, S., Davies, X., Shi, C., Gilbert, T. K., Scheurer, J., Rando, J., ... others (2023). Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv preprint arXiv:2307.15217*. Retrieved from `https://arxiv.org/abs/2307.15217`

Cassell, J. (2000). Embodied conversational agents. In *Mit press / aaai fall symposium on socially intelligent agents*.

Chaturvedi, S., & Chaturvedi, A. (2025). *Who gets the callback? generative AI and gender bias*. arXiv preprint arXiv:2502.XXXXX.

Chen, Y., Raghuram, V. C., Mattern, J., Mihalcea, R., & Jin, Z. (2025). Causally testing gender bias in llms: A case study on occupational bias. In *Findings of the association for computational linguistics: Naacl 2025* (pp. 4999–5019). doi: 10.18653/v1/2025.findings-naacl.281

Dakshit, S. (2024). Faculty perspectives on the potential of rag in computer science higher education. In *Proceedings of the conference on information technology education*. doi: 10.1145/3686852.3686864

de Fine Licht, K. (2024). Generative artificial intelligence in higher education: Why the 'banning approach' to student use is sometimes morally justified. *Philosophy & Technology*, *37*(4). doi: 10.1007/s13347-024-00799-9

Dennehy, T. C., & Dasgupta, N. (2017). Female peer mentors early in college increase women's positive academic experiences and retention in engineering. *Proceedings of the National Academy of Sciences*, *114*(23), 5964–5969. doi: 10.1073/pnas.1613117114

Diekman, A. B., Clark, E. K., Johnston, A. M., Brown, E. R., & Steinberg, M. (2011). Malleability in communal goals and beliefs influences attraction to STEM careers: Evidence for a goal congruity perspective. *Journal of Personality and Social Psychology*, *101*(5), 902–918. doi: 10.1037/a0025199

Dietvorst, B. J., Simmons, J. P., & Massey, C. (2015). Algorithm aversion: People erroneously avoid algorithms after seeing them err. *Journal of Experimental Psychology: General*, *144*(1), 114–126. doi: 10.1037/xge0000033

Elnaffar, R. F. . A. A. Z., S. (2025). Teaching with AI: A systematic review of chatbots, generative tools, and tutoring systems in programming education. *International Journal of Learning, Teaching and Educational Research*.

Eurostat. (2024). ICT specialists in employment. (`https://ec.europa.eu/eurostat/statistics-explained/index.php?title=ICT_specialists_in_employment`)

Feine, J., et al. (2020). Gender bias in chatbot design. In A. Følstad et al. (Eds.), *Chatbot research and design* (Vol. 11970, pp. 79–93). Springer, Cham. doi: 10.1007/978-3-030-39540-7_6

Feng, T., Liu, S., & Ghosal, D. (2024). Courseassist: Pedagogically appropriate ai tutor for computer science education. In *Proceedings of the sigcse virtual*. doi: 10.1145/3649409.3691094

Fetrati, H., Chan, G., & Orji, R. (2025). Leveraging generative and rule-based models for persuasive STI education: A multi-chatbot mobile application. In *Proceedings of the 7th acm conference on conversational user interfaces*. Waterloo, ON, Canada: ACM.

Freeman, J. (2025). *Student generative ai survey 2025* (Tech. Rep.). Oxford: Higher Education

Policy Institute. Retrieved 2026-08-12, from https://www.hepi.ac.uk/wp-content/uploads/2025/02/HEPI-Kortext-Student-Generative-AI-Survey-2025.pdf

Gallegos, I. O., et al. (2023). Bias and fairness in large language models: A survey. *Computational Linguistics*.

Garg, N., Schiebinger, L., Jurafsky, D., & Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. In *Pnas.* doi: 10.1073/pnas.1720347115

Giannakos, M., Azevedo, R., Brusilovsky, P., Cukurova, M., Dimitriadis, Y., Hernandez-Leo, D., ... Rienties, B. (2025). The promise and challenges of generative ai in education. *Behaviour & Information Technology*, *44*(11), 2518–2544. doi: 10.1080/0144929X.2024.2394886

Goetz, J., Kiesler, S., & Powers, A. (2003). Fun in learning: The role of embodied conversational agents in education. *Cognitive Systems Research*, *4*(4), 409–427. doi: 10.1016/S1389-0417(03)00037-7

Higher Education Authority. (2026). *Trends in Non-Progression 2016/2017–2023/2024: Course Characteristics.* https://hea.ie/statistics/data-for-download-and-visualisations/students/progression/hea-progression-and-completion-report-2026/hea-non-progression-course-characteristics-2026/. (Accessed: 2026-08-12)

Jamie, P., Hajihashemi, R., & Alipour, S. (2025). Utilizing chatgpt in a data structures and algorithms course: A teaching assistant's perspective. In *Proceedings of the extended abstracts of the chi conference on human factors in computing systems* (pp. 1–7).

Kasneci, E., Seßler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., ... Kasneci, G. (2023). ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences*, *103*, 102274. doi: 10.1016/j.lindif.2023.102274

Kotek, H., Dockum, R., & Sun, D. (2023). Gender bias and stereotypes in large language models. In *Proceedings of the acm collective intelligence conference (ci '23).* ACM. doi: 10.1145/3582269.3615599

Kuhail, M. A., Alturki, N., Alramlawi, S., & Alhejori, K. (2023). Interacting with educational chatbots: A systematic review. *Education and Information Technologies*, *28*(1), 973–1018. doi: 10.1007/s10639-022-11177-3

Leavitt, C., Corbin, M., Keith, M. J., Schuetzler, R. M., Anderson, G., & Kettles, D. (2022). A chatbot tutor can lessen the gender confidence gap in information systems learning. In *Proceedings of the 55th hawaii international conference on system sciences* (pp. 1–10).

Lehman, K. J., Newhouse, K. N., Sundar, S., & Sax, L. J. (2023). Nevertheless, they persisted: Factors that promote persistence for women and racially/ethnically minoritized students in undergraduate computing. *Computer Science Education*, *33*(2), 260–285.

Liu, R., Zenke, C., Liu, C., Holmes, A., Thornton, P., & Malan, D. J. (2024). Teaching cs50 with ai: Leveraging generative artificial intelligence in computer science education. In *Proceedings of the technical symposium on computer science education.* doi: 10.1145/3626252.3630938

Mahajan, R. (2025). Retrieval-augmented generation: The technical foundation of intelligent AI chatbots. *World Journal of Advanced Research and Reviews*.

Martin, A., Ying, K. M., Rodríguez, F. J., Kahn, C., & Boyer, K. (2022). Intelligent support for all?: A literature review of the (in)equitable design evaluation of adaptive pedagogical systems for cs education. In *Proceedings of the technical symposium on computer science education.* doi: 10.1145/3478431.3499418

Maurat, J. I. C., Isip, E. V., & Lumabas, A. G. R. (2024). A comparative study of gender differences in the utilization and effectiveness of AI-assisted learning tools in programming among university students. In *Proceedings of the 2024 international conference on artificial intelligence and teacher education.*

Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2021). A survey on bias and fairness in machine learning. *ACM Computing Surveys*, *54*(6), 1–35. doi: 10.1145/3457607

Mumford, E. (2000). *Socio-technical design: An unfinished science*. Information Systems Institute,

London School of Economics.

Murphy, M. C., Steele, C. M., & Gross, J. J. (2007). Signaling threat: How situational cues affect women in math, science, and engineering settings. *Psychological Science*, *18*(10), 879–885. doi: 10.1111/j.1467-9280.2007.01995.x

Németh, R., Tátrai, A., Szabó, M., & Tamási, j. y. d., Á. (n.d.). Using a rag-enhanced large language model in a virtual teaching assistant role: Experiences from a pilot project in statistics education.

Ormeño-Arriagada, P., Herrera, B., & Queirolo, F. (2025). Towards inclusive computing: A retrieval-augmented generation prototype for gender equity in computer science curricula. In *Proceedings of the international conference of the chilean computer science society.* doi: 10.1109/SCCC67219.2025.11420736

Osunde, J. O., Windall, G., Bacon, L., & Mackinnon, L. (2014). Female under-representation in computing education and industry: A survey of issues and interventions. *International Journal of Advanced Computer Science and Applications*, *5*(9). doi: 10.14569/IJACSA.2014.051003

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., ... Lowe, R. (2022). Training language models to follow instructions with human feedback. In *Proceedings of the 36th international conference on neural information processing systems.* Red Hook, NY, USA: Curran Associates Inc.

Pantic, K., & Clarke-Midura, J. (2019). Factors that influence retention of women in the computer science major: A systematic literature review. *Journal of Women and Minorities in Science and Engineering*, *25*(2).

Parasuraman, R., & Riley, V. (2000). Models of automation-induced complacency and automation bias. *Human Factors*, *42*(2), 287–301. doi: 10.1518/001872000779698329

Passi, S., & Barocas, S. (2020). Sociotechnical approaches to ai fairness. In *Proceedings of the aaai/acm conference on ai, ethics, and society (aies).* ACM.

Perez-Felkner, L., Erichsen, K., Li, Y., Chen, J., Hu, S., Ramirez Surmeier, L., & Shore, C. (2025). Computing education interventions to increase gender equity from 2000 to 2020: A systematic literature review. *Review of Educational Research*, *95*(3), 536–580.

Pitts, G., Hridi, A. P., & Narayanan, A. B. L. (2025). A survey of llm-based applications in programming education: Balancing automation and human oversight. In *Proceedings of the fourth workshop on bridging human-computer interaction and natural language processing (hci+ nlp)* (pp. 255–262).

Sadat Shanto, S., Ahmed, Z., & Jony, A. I. (2024). Generative ai for programming education: Can chatgpt facilitate the acquisition of fundamental programming skills for novices? In *Proceedings of the 3rd international conference on computing advancements* (pp. 685–692).

Selbst, A. D., Boyd, D., Friedler, S. A., Venkatasubramanian, S., & Vertesi, J. (2019). Fairness and abstraction in sociotechnical systems. In *Proceedings of the conference on fairness, accountability, and transparency (fat* / facct)* (pp. 59–68). ACM.

Smutny, P., & Schreiberova, P. (2026). From rules to language models: a comparative study of chatbot learning assistants. *Frontiers in Education*, *11*. Retrieved from `https://www.frontiersin.org/journals/education/articles/10.3389/feduc.2026.1794807/full` doi: 10.3389/feduc.2026.1794807

Steele, C. M., & Aronson, J. (1995). Stereotype threat and the intellectual test performance of African Americans. *Journal of Personality and Social Psychology*, *69*(5), 797–811.

Swacha, J., & Gracel, J. (2025). Retrieval-augmented generation (RAG) chatbots for education: A survey of applications. *Applied Sciences*, *15*(8), 4234. doi: 10.3390/app15084234

Szlavi, A., Haugland, H. S., & Jaccheri, L. (2024). Role models as an intervention for gender diversity in computing education. In *Norsk ikt-konferanse for forskning og utdanning.*

Thompson, L. D., & Wu, Z. (2021). Increasing the enrollment, retention, and graduation of undergraduate women majoring in computing. In *2021 conecd.*

Wan, Y., Pu, G., Sun, J., Garimella, A., Chang, K. W., & Peng, N. (2023). "kelly is a warm person, joseph is a role model": Gender biases in llm-generated reference letters. In *Findings of the association for computational linguistics: Emnlp 2023* (pp. 3730–3748).

Wang, K., & Lawrence, R. (2025). Quantitative evaluation of using large language models and retrieval-augmented generation in computer science education. In *Proceedings of the technical symposium on computer science education.* doi: 10.1145/3641554.3701917

Waytz, A., Cacioppo, J., & Epley, N. (2009). Making sense by making sentient: Effectance motivation increases anthropomorphism. *Journal of Personality and Social Psychology*, *96*(5), 864–877. doi: 10.1037/a0014795

Yansen, G., & Zukerfeld, M. (2014). Why don't women program? exploring links between gender, technology and software. *Science, Technology and Society*, *19*(3), 305–329. doi: 10.1177/0971721814548111

Yigci, D., Eryilmaz, M., Unal, E., Eryilmaz, N. O., Akcapinar, G., Reisoglu, I., ... Celik, B. (2024). Large language model-based chatbots in higher education. *Advanced Intelligent Systems*. doi: 10.1002/aisy.202400429

Narrative Structures for Process-Level Reasoning in Software Engineering

Md. Hasan Tareque
Dept. of Mathematics and CS
University of Lethbridge
mdhasan.tareque@uleth.ca

John Anvik
Dept. of Mathematics and CS
University of Lethbridge
john.anvik@uleth.ca

Abstract

Software engineering students often struggle to reason about how development activities interact across the software lifecycle. Requirements, design, implementation, and testing are frequently learned as isolated topics, leading to fragmented process-level mental models. This work-in-progress paper explores whether narrative structures can function as cognitive scaffolds for software engineering reasoning. We propose an instructor-guided framework in which large language models generate short instructional narratives aligned with software engineering concepts. The framework maps narrative structures such as the Hero's Journey, the Fichtean Curve, and the Three-Act Structure onto software engineering reasoning tasks involving lifecycle progression, architectural evolution, and testing processes. The paper presents a framework for using narrative-based representations to support process-level reasoning and outlines future classroom evaluation.

1. Introduction

Software engineering students are expected to reason about how requirements, design, implementation, testing, deployment, and maintenance activities interact across the software lifecycle (Pressman, 2005; Sommerville, 2016). However, these topics are often taught separately, making it difficult for students to connect decisions made in one stage with consequences that appear later in development.

Many degree programmes address this fragmentation through semester-long or two-semester team software projects. These projects provide valuable opportunities for students to experience multiple development activities within a shared context. Nevertheless, participation in an integrated project does not necessarily ensure that students explicitly recognize the causal relationships among those activities. Project scope and duration may also limit students' exposure to later lifecycle activities such as deployment, long-term maintenance, and architectural evolution.

This challenge is especially important in areas such as DevOps, maintenance, testing, and architecture, where students must reason about trade-offs, dependencies, and cascading consequences. Learners may understand individual concepts, or participate in projects involving several development activities, while still struggling to explain how decisions made during one activity influence outcomes elsewhere in the software process.

This work-in-progress paper examines whether narrative structures can support process-level reasoning in software engineering education. Rather than using stories primarily for engagement, the proposed framework examines whether narrative forms can help learners organize software engineering processes through structured narrative progression.

The framework maps narrative structures such as the Hero's Journey, the Fichtean Curve, and the Three-Act Structure onto software engineering reasoning tasks. Large language models are used to generate short instructional narratives under instructor guidance, allowing scalable creation of process-oriented scenarios while maintaining pedagogical control.

The paper presents a conceptual framework rather than an empirical evaluation. It introduces a framework for using narrative structures to support process-level reasoning in software engineering education and outlines a future exploratory classroom study suitable for investigation within the PPIG community.

2. Related Work and Motivation

Research on program comprehension shows that learners may struggle to integrate software structure, behaviour, intent, and dependencies into coherent mental models (Pennington, 1987; Letovsky, 1986; Johnson-Laird, 1983). Experts tend to organize knowledge around higher-level concepts, whereas novices rely more heavily on surface features (Adelson, 1981; Chi, Feltovich, & Glaser, 1981; Petre, 2013; Storey, 2005). This distinction matters in software engineering, where reasoning must extend across requirements, design, implementation, testing, and maintenance. When these topics are presented separately, the consequences of decisions made in one activity may remain difficult to trace in another (Sommerville, 2016; Sharp, Dittrich, & de Souza, 2009). The growing emphasis on process thinking, systems reasoning, and AI-supported instruction further increases the need for representations that connect activities across the lifecycle (Marlowe, Ku, Laracy, Kirova, & Herbert, 2026).

Narrative cognition offers a possible foundation for such representations. People use narratives to organize events, actions, and consequences over time (Bruner, 1991; Schank, 1995), while situation-model research explains how readers construct causal and temporal coherence from connected events (Graesser & Bower, 1980; van Dijk & Kintsch, 1983; Trabasso & van den Broek, 1985). Research on external cognition likewise shows that representational form can influence reasoning by exposing relevant relationships, distributing cognitive work, and directing attention (Larkin & Simon, 1987; Zhang & Norman, 1994; Scaife & Rogers, 1996). Narrative may therefore provide a complementary way to represent how software processes develop through interacting decisions and constraints.

Evidence from technical practice supports this possibility. Orr's ethnographic study of copier technicians found that diagnostic stories preserved contextualized experience, supported collaborative interpretation of symptoms and tests, and contributed to a shared community model (Orr, 1986). Those stories emerged naturally in professional work, unlike the instructor-designed narratives proposed here, but they demonstrate that narrative can participate in technical reasoning rather than merely describe it.

Story-based approaches have also been used in computing education through contextualized programming, storytelling environments, and scenario-based learning (Kelleher & Pausch, 2007; Rankin & Barnes, 2009; Pierson, Kinnunen, & Simon, 2022). Erwig's *Once Upon an Algorithm*, for example, uses familiar situations to explain algorithms, data structures, abstraction, recursion, searching, and unsolvability (Erwig, 2017). Its focus is primarily on individual computing concepts, whereas the present work considers relationships among software engineering activities across time.

Related studies have examined narrative interpretation of code behaviour and software processes (Dey, Powell, & Eijkhout, 2025), as well as interactive stories intended to provide contextualized and inclusive learning experiences (Ghailani, Malaise, & Signer, 2025). Much of this literature emphasizes explanation, engagement, accessibility, or motivation. This paper instead investigates narrative structures as representations for process-level reasoning and mental-model construction.

Developments in Artificial Intelligence in Education and LLM-based support now make varied instructional scenarios easier to produce (Kasneci et al., 2023; Holmes & Porayska-Pomsta, 2022). Here, however, the LLM serves only as an instructor-guided generation mechanism. The central contribution is the use of narrative structure to organize reasoning about relationships across the software lifecycle.

3. Narrative Structures as Process Representations

The proposed framework reorganizes software engineering topics into narrative scenarios that emphasize progression, dependency relationships, and consequences across development activities.

The framework is instructor-guided. Instructors select the software engineering topic, determine the learning objective, and decide how narrative structures should map onto development activities. Large language models are then used to generate short instructional scenarios from these mappings. The goal is not automated instruction, but scalable generation of contextualized examples under pedagogical oversight. Figure 1 summarizes the proposed workflow.

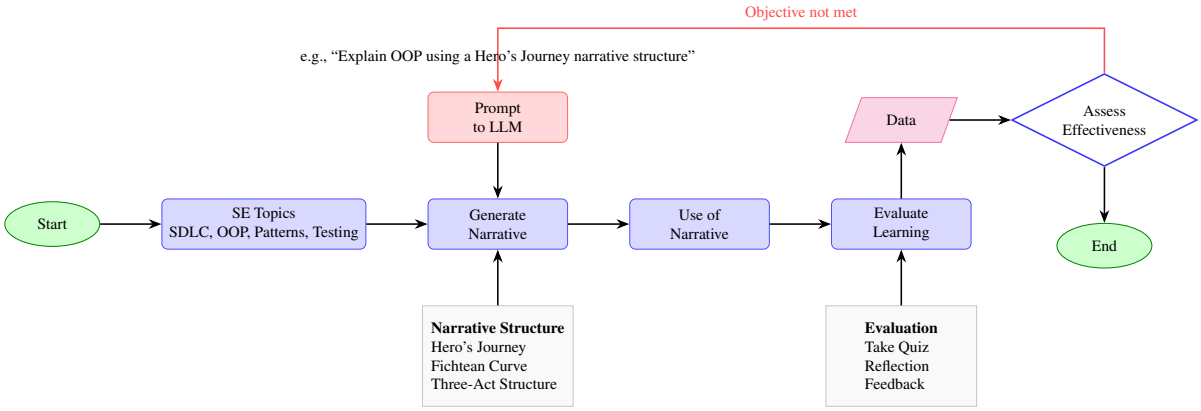


Figure 1 – Instructor-guided narrative generation workflow.

The current version of the framework focuses on three narrative structures: the Hero's Journey, the Fichtean Curve, and the Three-Act Structure. These structures were selected because they provide contrasting ways of organizing change over time: transformation across an extended journey, escalation through accumulating pressures, and resolution of a bounded problem. The proposed mappings are based on three considerations. First, there should be *structural correspondence*: transitions between narrative phases should correspond to meaningful changes in the software process rather than merely provide decorative story elements. Second, the structure should foreground the *causal relationship* relevant to the learning objective, such as progression, accumulation, or diagnosis. Third, the mapping should have a distinct *pedagogical purpose* by drawing attention to relationships that may be less explicit in a conventional lifecycle diagram or topic-by-topic presentation.

The Hero's Journey provides a temporal scaffold for lifecycle progression. Requirements and stakeholder needs establish the problem; design, implementation, and iteration transform the system; and testing, deployment, and maintenance reveal the effects of earlier decisions. The analogy does not portray development as a linear heroic quest, but highlights progression and iterative returns across lifecycle activities.

The Fichtean Curve foregrounds the pressures that drive design evolution. Changing requirements, technical debt, and interacting constraints increase tension until the existing design requires restructuring or refactoring. Figure 2 illustrates this progression through a late authentication requirement: local workarounds increase technical pressure until redesign becomes necessary, after which testing determines whether stability has been restored.

The Three-Act Structure frames testing and debugging as movement from expectation to disruption and verified resolution. The setup defines expected behaviour and test conditions, while confrontation introduces failures and unexpected interactions. Resolution combines correction, validation, and regression testing, showing how evidence revises the learner's explanation of system behaviour.

These mappings are neither exhaustive nor exclusive. A testing scenario could, for example, be represented through a Fichtean Curve when failures accumulate across several releases, while architectural evolution could be represented through a Three-Act Structure when the learning objective concerns a single redesign episode. The appropriate mapping therefore depends on the relationship learners are expected to reason about, rather than on a fixed association between a narrative form and a software engineering topic. At this work-in-progress stage, the mappings should be understood as theoretically motivated design hypotheses whose usefulness and limitations require empirical investigation.

The mappings shown in Figure 3 organize software engineering concepts as evolving process relationships rather than as isolated instructional topics. Together, they distinguish three reasoning demands: tracing progression across phases, explaining how constraints accumulate, and diagnosing how a disrup-

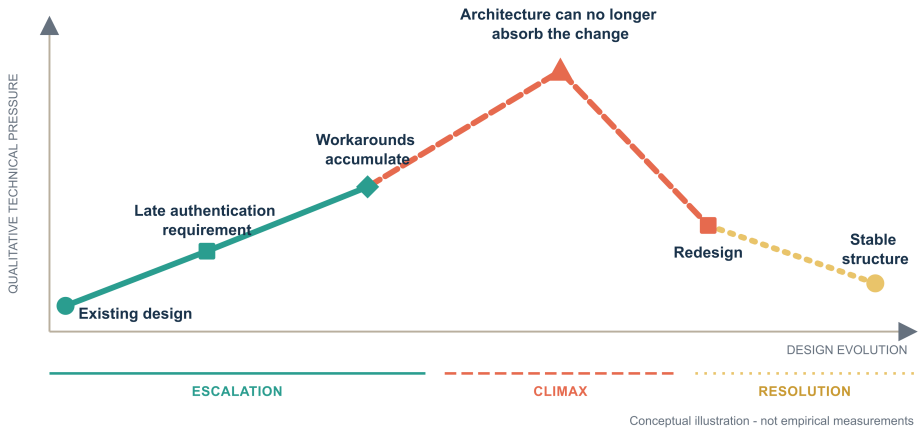


Figure 2 – Conceptual application of the Fichtean Curve to design evolution. A late authentication requirement and accumulating workarounds increase technical pressure until redesign becomes necessary. The vertical axis is qualitative rather than empirically measured.

tion is resolved. From a cognitive perspective, these representations may support temporal organization by representing development activities as connected event sequences, causal reasoning by linking decisions with downstream effects, and process simulation by helping learners mentally trace how software processes evolve over time.

These mappings are intended to support reasoning about process relationships rather than factual recall. A testing failure, for example, may be framed as the consequence of earlier requirements, design, or implementation decisions. Similarly, refactoring may be represented not simply as a technique, but as a response to accumulated design tension. In this way, narrative scenarios can complement UML diagrams, workflows, and lifecycle models by foregrounding temporal progression, causal propagation, and downstream effects.

A simple example illustrates the representational shift. A conventional instructional description might present a late requirements change as a sequence of separate development activities. A workflow diagram can show the order of activities, but it does not by itself show why the later testing problem occurred.

> Requirements change → architecture modification → implementation updates → regression testing failures

Within the proposed framework, the same situation can be represented narratively:

> A development team introduces authentication requirements late in the implementation phase. Existing architectural assumptions no longer support the new requirements, requiring significant redesign. The resulting implementation changes introduce unexpected interactions that later appear as regression test failures during deployment preparation.

The underlying phenomenon represented in both versions is the same software development process. What changes is the representational form. The conventional version lists activities and outcomes, whereas the narrative version presents them as a story involving actors with a goal, a disruption to an existing situation, decisions made in response, and consequences that emerge over time. In this distinction, *process* refers to the actual relationships among software engineering activities, while *story* refers to the instructional representation used to make those relationships explicit. The story should therefore preserve the relevant technical mechanisms rather than replace them with merely dramatic events.

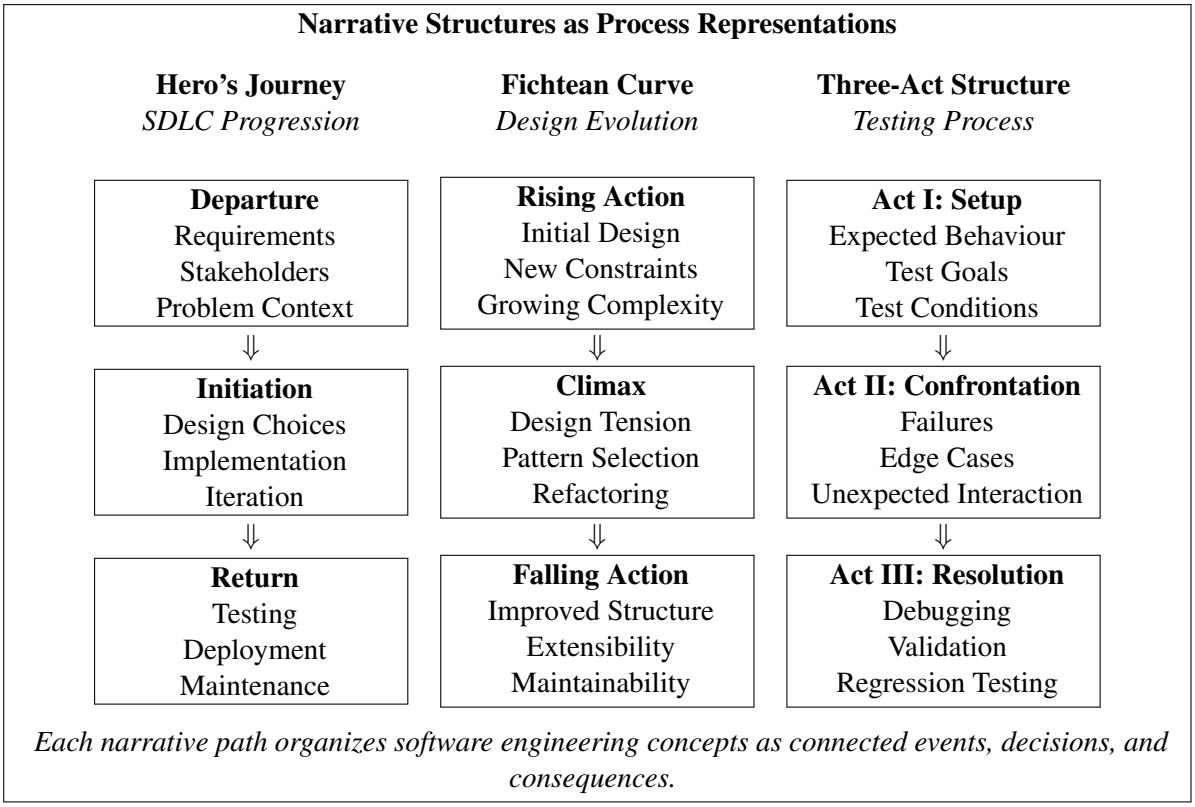


Figure 3 – Narrative structures as representations of software engineering reasoning paths.

In the example, the late authentication requirement creates pressure on architectural assumptions that were originally designed for a simpler interaction model. The redesign may introduce new abstractions, changes to database access patterns, and modifications to testing infrastructure. These changes can then produce integration issues and regression failures that were not visible when the original implementation decisions were made. The narrative form foregrounds the causal chain by connecting the team's goal and decisions to the later outcomes; the process itself remains the sequence of software engineering activities and dependencies being represented.

The narrative transformation reorganizes software engineering concepts into connected process explanations intended to represent how software processes evolve through interacting decisions, constraints, and outcomes. Narrative structures act as process-oriented representations that may help learners reason about how software engineering activities influence one another over time.

This is consistent with situation model theory, where learners build an internal representation of how events, actions, and consequences are connected (van Dijk & Kintsch, 1983; Trabasso & van den Broek, 1985). This framing aligns with interests in program comprehension, mental models, and representational support for reasoning.

Instructional narratives are generated through an instructor-guided process aligned with software engineering concepts and learning objectives. Instructors select target topics, choose narrative structures appropriate for the intended reasoning tasks, and develop prompts that align narrative phases with software engineering activities.

LLMs are then used to generate short instructional narratives that students engage with alongside lectures, scenario-based exercises, and classroom discussions intended to support process-level reasoning.

4. Discussion and Future Work

This framework treats narrative structures as instructional representations for reasoning about software engineering processes. This framing shifts attention away from narratives as engagement devices and toward narratives as representational forms that may shape how learners organize, interpret, and reason about software engineering processes.

One implication of this work is that software engineering education may benefit from additional representational forms beyond diagrams and workflow models. Conventional representations such as lifecycle diagrams, workflow models, and UML diagrams are valuable because they make structural and procedural relationships visible. However, they do not always foreground the temporal, causal, and consequential nature of software engineering decisions. Narrative structures instead organize development activities as sequences of events in which earlier decisions create later consequences.

The framework also raises several questions for future work in software engineering education and the psychology of programming. If narrative structures support process-level reasoning, then future work should examine which narrative features matter most: temporal progression, causal linking, conflict, agency, consequence, or resolution. It should also examine when narrative representations are less effective than formal representations, particularly for topics requiring precision, notation, or formal constraint reasoning.

Future work will empirically investigate whether narrative-based representations influence how learners reason about software engineering processes and relationships across development stages. A planned exploratory classroom study will compare narrative-based instructional scenarios with conventional process-oriented representations using scenario-based reasoning tasks. The study will examine how learners explain dependencies, consequences, and progression across software engineering activities, with particular attention to whether narrative representations support more coherent process-level explanations.

5. Conclusion

This work-in-progress paper explored whether narrative structures can function as cognitive scaffolds for process-level reasoning in software engineering education. The proposed framework treats narrative forms as instructional representations that organize software engineering concepts through progression, conflict, and consequence. By mapping structures such as the Hero's Journey, the Fichtean Curve, and the Three-Act Structure onto software engineering reasoning tasks, the framework aims to support more coherent mental models of how development activities interact across the software lifecycle.

The paper contributes a conceptual framework for integrating narrative-based representations with instructor-guided LLM-generated instructional scenarios. Rather than replacing lifecycle models, workflows, or UML diagrams, the framework explores how narrative structures may complement them by making process relationships and downstream effects more visible to learners.

Future work will investigate whether narrative-based instructional representations influence learners' ability to reason about progression, dependencies, and consequences across software engineering activities. More broadly, this work contributes to ongoing PPIG discussions surrounding mental models, program comprehension, and the role of external representations in supporting software engineering reasoning.

6. Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the Discovery Grants program.

Generative AI tools were used to assist with language refinement and editing during the preparation of this paper.

7. References

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory and Cognition*, *9*(4), 422–433.

Bruner, J. (1991). The narrative construction of reality. *Critical Inquiry*, *18*(1), 1–21.

Chi, M. T. H., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science*, *5*(2), 121–152.

Dey, S. C., Powell, J. H., & Eijkhout, V. (2025). Coding through storytelling: Narrative reasoning and software engineering education. *Journal of Computational Science Education*, *16*(2), 5–9. doi: 10.22369/issn.2153-4136/16/2/2

Erwig, M. (2017). *Once Upon an Algorithm: How stories explain computing*. The MIT Press.

Ghailani, I. G., Malaise, Y., & Signer, B. (2025). JsStories: Improving social inclusion in computer science education through interactive stories. *arXiv preprint arXiv:2504.04006*. Retrieved from `https://arxiv.org/abs/2504.04006`

Graesser, A. C., & Bower, G. H. (1980). Story understanding as problem solving. *Poetics*, *9*(1–3), 223–250.

Holmes, W., & Porayska-Pomsta, K. (2022). The ethics of AI in education: Towards a community-wide framework. *International Journal of Artificial Intelligence in Education*, *32*(3), 504–526.

Johnson-Laird, P. (1983). *Mental models: Towards a cognitive science of language, inference, and consciousness*. Harvard University Press. Retrieved from `https://books.google.ca/books?id=FS3zSKAfLGMC`

Kasneci, E., et al. (2023). ChatGPT for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, *103*, 102274.

Kelleher, C., & Pausch, R. (2007). Using storytelling to motivate programming. *Communications of the ACM*, *50*(7), 58–64.

Larkin, J. H., & Simon, H. A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, *11*(1), 65–100.

Letovsky, S. (1986). Cognitive processes in program comprehension. In *Proceedings of the first workshop on empirical studies of programmers* (pp. 58–79).

Marlowe, T. J., Ku, C. S., Laracy, J. R., Kirova, V. D., & Herbert, K. G. (2026). A systemic view of a software engineering education curriculum: Requirements and guidelines in the era of generative AI. *Journal of Integrated Design and Process Science*. doi: 10.1177/10920617251405471

Orr, J. E. (1986). Narratives at work: Storytelling as cooperative diagnostic activity. In *Proceedings of the 1986 acm conference on computer-supported cooperative work* (pp. 62–72). ACM. doi: 10.1145/637069.637077

Pennington, N. (1987). Comprehension strategies in programming. *Empirical Studies of Programmers: Second Workshop*, 100–113.

Petre, M. (2013). Expertise and collaboration in software engineering. *2013 35th International Conference on Software Engineering (ICSE)*, 1397–1400.

Pierson, E., Kinnunen, P., & Simon, B. (2022). Storytelling as a pedagogical tool in computer science education. *ACM Transactions on Computing Education*, *22*(4), 1–25. doi: 10.1145/3494145

Pressman, R. S. (2005). *Software engineering: A practitioner's approach* (6th ed.). McGraw-Hill.

Rankin, Y., & Barnes, T. (2009). Contextualized gaming for learning. In *Proceedings of the 4th international conference on foundations of digital games* (pp. 244–251).

Scaife, M., & Rogers, Y. (1996). External cognition: How do graphical representations work? *International Journal of Human-Computer Studies*, *45*(2), 185–213.

Schank, R. C. (1995). *Tell me a story: Narrative and intelligence*. Northwestern University Press.

Sharp, H., Dittrich, Y., & de Souza, C. (2009). *The role of ethnographic studies in empirical software engineering*. Springer.

Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.

Storey, M.-A. (2005). Theories, methods and tools in program comprehension: Past, present and future. In *13th international workshop on program comprehension (IWPC'05)* (pp. 181–191).

doi: 10.1109/WPC.2005.38

Trabasso, T., & van den Broek, P. (1985). Causal thinking and the representation of narrative events. *Journal of Memory and Language*, *24*(5), 612–630.

van Dijk, T. A., & Kintsch, W. (1983). *Strategies of discourse comprehension*. Academic Press.

Zhang, J., & Norman, D. A. (1994). Representations in distributed cognitive tasks. *Cognitive Science*, *18*(1), 87–122.

Can Gamification Encourage Debugger Usage?

Teresa Dreyer
PLTlab.de
Marburg University
tdreyer@mathematik.uni-marburg.de

Christoph Bockisch
PLTlab.de
Marburg University
bockisch@acm.org

Abstract

Computer Science students often don't make use of the debugger. If they do, they only use its basic functionality. We propose encouraging students to better exploit the debuggers' potential by providing them with debugging related gamification elements integrated into their usual IDE. Among several suitable gamification elements, we are focusing on achievements which are well-researched and work well across different contexts, including software engineering. We have developed an IntelliJ IDEA plugin that awards achievements for reaching specific milestones when using the debugger such as using the stepper for the first time or setting 100 breakpoints. We conducted an experiment with second semester Bachelors' students in which we required them to use the plugin while working on programming tasks. Half of the participants were shown the debugger-related achievements and the other half used a non-active version of the plugin where usage data was only collected but not displayed. The results from an initial analysis of students' self-report data look promising.

1. Introduction

Debugging makes up a significant part of software development, yet developers are rarely formally trained in debugging techniques (Ko, Bose, Chowdhury, Seyam, & Brown, 2023; Perscheid, Siegmund, Taeumel, & Hirschfeld, 2016). Two other observations are common and related: Computer Science students often struggle with debugging (Yang, Baird, O'Rourke, Brennan, & Schneider, 2024) and underuse the debugger (Ko et al., 2023). Often they don't use a dedicated debugger at all and instead debug using print statements (Ko et al., 2023). If they do use it, they often only use basic features like line breakpoints although the debuggers integrated into modern IDEs provide a substantially broader range of functionality including different breakpoint types as well as a dedicated debugging view with functionality for inspecting a suspending program frame by frame such as a stepper and an overview over current variable values and active threads.

Gamification, the practice of taking elements of games and applying them to non-game contexts, has successfully been used to teach people desired behavior in many areas. Different categories of gamification elements exist such as rewards, streaks, leaderboards, and progression systems. The inclusion of such elements in user facing applications is a popular means to increase users' motivation and persistence. Notable examples are for example streaks for daily activity in the language learning app Duolingo or the social media platform Snapchat, leaderboards and a level-based progression system also in Duolingo, and badges on the Q&A-website Stack Overflow. The application of gamification strategies to software engineering has been extensively studied for several years, both within and outside of a dedicated educational context with overall promising results.

We are investigating how the inclusion of debugging-related gamification elements into an IDE familiar to our students can help increase their motivation to use the IDE's debugger more as well as exploit more of its feature range. As a consequence of increased usage and feature exploitation, students are also expected to implicitly learn how to use the debugger as well as more effective debugging strategies.

2. State of the Art

Among the various available gamification elements, achievements in general have been stated to be one of the most basic and elementary ones (Dubois & Tamburrelli, 2013): The relationship between the user's behavior and reward is instantaneous, they do not rely on a multi-day process (in contrast to

streaks) or on other users' behavior (as would be the case with leader boards). Yet, they are generally effective at increasing motivation, which was also confirmed in the software engineering context (Souza, Constantino, Veado, & Figueiredo, 2017; Porto, Jesus, Ferrari, & Fabbri, 2021) whereas the picture for competition based gamification elements like leaderboards is more mixed (Souza et al., 2017). These properties make achievements a very suitable starting candidate for a gamification element to study.

In general, gamification research in software engineering has shown very promising results: Souza et al. (2017) collected survey data indicating that gamification elements can be beneficial in software engineering education. In a recent systematic mapping study, Porto et al. (2021) were able to confirm this. Two studies from the area of software testing are particularly interesting in comparison with our endeavor: In an experiment spanning over a few weeks, Ren (2023) confirmed that students' test-driven development practices can be positively influenced using gamification. Straubinger and Fraser (2024) introduced IntelliGame, an IntelliJ plugin that rewards the user for desired testing behavior with achievements. Using a controlled experiment, they were able to show that students using their plugin displayed more desired testing behavior than students who did not use the plugin. Because of the generally promising results of applying gamification techniques to software engineering contexts, it is plausible to assume that similar effects should be discoverable when gamifying debugging in a similar manner.

3. A Plugin to Reward Debugger Use

As a platform for researching the influence of gamification elements on debugger usage, we have developed an IntelliJ IDEA plugin. The choice of the environment IntelliJ IDEA plugin is primarily motivated by our students' familiarity with it.

The plugin was designed with the explicit goal of conducting controlled experiments in mind. When conducting controlled experiments, it is central to eliminate potential systematic confounding variables. Comparing study participants working with the plugin to study participants with an unmodified version of the IDE would introduce at least two potential confounding variables: a) the general layout of the IDE would be different and b) students working with the plugin would likely need to install it first. Instead, we propose a different approach inspired by experimental designs in medical and clinical psychological research: We have implemented a *"placebo"* version and an *"active"* version of the plugin: Depending on a code the participant enters when setting up the plugin, they receive either the *placebo* version consisting of an otherwise empty sidebar with an export button for debugger usage data or the *active* version in which the implemented gamification elements are enabled. With general environment layout and installation differences eliminated as potential confounding variables, it is possible to use our plugin for conducting a controlled experiment in which the only systematically differentiating factor between participants is whether they have access to the gamification elements or not.

Due to their simplicity, achievements were selected as a first gamification element to examine. This decision is backed by the research by Dubois and Tamburrelli (2013) and Souza et al. (2017). To encourage students to explore different types of breakpoints as well as the debugger's stepper, 8 breakpoint achievements corresponding to the 8 breakpoint types available in IntelliJ IDEA were implemented as well as 5 achievements associated with using the stepper to explore the program state frame by frame. Additionally, 3 hidden achievements were implemented for which only question marks are displayed until they are unlocked. The hidden achievements aim at freely encouraging exploration of additional debugger functionality which is not covered by the other achievements. Exploring the program state and other more advanced features are not currently covered by our achievements due to time constraints.

Plugin Layout The *active* version of the plugin includes a sidebar that shows the available achievements, see figure 1 for an excerpt. Both the *active* and the *placebo* version of the plugin collect debugger usage data associated with the implemented achievements which can be exported by using an `export` button. In the *placebo* version, only this export button is displayed. The general layout and design of the *active* version of the plugin is inspired by IntelliGame (Straubinger & Fraser, 2024).

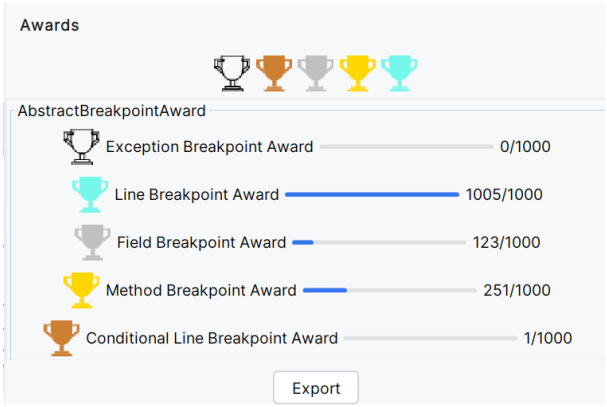


Figure 1 – Screenshot of the sidebar for the "active" version of our plugin showing part of the achievements and the export button. In the "placebo" version, only the export button is displayed.

Similar to IntelliGame, our implementation of the *active* plugin version also uses a level system for the breakpoint and stepper related achievements: For the first instance of an achievement related behavior, for example setting a line breakpoint, the user is awarded the bronze level award, for future instances silver, gold and platinum achievements are awarded with an increasingly higher distance between achievements, for the 100th, 250th and 1000th instance of an achievement-associated behavior, respectively. A progress bar shows how far the developer is from the platinum level. In addition, popups notify the user when an achievement is reached, and on occasion as well for additional milestones, with similarly decreasing frequency. The hidden achievements only have one level each.

4. User Study

We have evaluated our plugin in a university course that is mandatory for Bachelors' students in Computer Science, Business Informatics and Data Science. The course in question is mainly attended at the end of the second semester of the Bachelor's degree. At this point in time, the students have a basic orientation in object oriented programming, algorithms, data structures, and other programming concepts. All students were required to follow the provided tasks. After completing the tasks, they were allowed to freely choose whether they wanted to share their data for research purposes. We formulated the following general hypotheses:

A) Students using the active version of the plugin will have larger values than students using the placebo version for the questions in the self-report that are associated with debugger usage during the class and expectations for future debugger usage.

B) Students using the active plugin version will use the debugger more than students with the placebo version.

4.1. Experimental Setup

All participants were instructed to install the plugin together with a compatible version of IntelliJ IDEA. Afterwards they were given a code to enable the plugin. The students then worked on provided programming tasks and were instructed to use the debugger. Most of the students worked in teams of two. The students were divided into two experimental groups: The treatment group was provided with the *active* version of our plugin whereas the control group was provided with the *placebo* version. For both groups, debugger usage data was collected and exported afterwards. Students working in pairs were assigned to the same experimental group. We collected self-report data in the form of a questionnaire from 111 students of which 91 students agreed to have their data used for research purposes. Two of them were excluded because it could not be reconstructed in which experimental group they had been in.

4.2. A Look at the Data

In an initial analysis, we focused on hypothesis A) and considered only the self-report data, particularly the students' responses to the four relevant questions. For each one, we performed a simple linear regression with the experimental condition (control vs. treatment condition) as predictor variable and the score in the respective questionnaire item as outcome variable. The predictor variable was dummy encoded with the control condition encoded with 0 and the treatment condition encoded with 1. Additionally, we performed a simple linear regression using the same predictor variable and the sum score of the four questionnaire items as outcome variable. The results can be found in table 1. The exact wording of the corresponding questions is found in table 2 in the appendix. While regression weights are rather small and not statistically significant, for all relevant questions from the questionnaire as well as the sum score, regression weights are positive.

Table 1 – Results of the initial data analysis

Dependent Variable (DV)	intercept	regression weight	intercept (DV standardized)	regression weight (DV standardized)
DV1: Intensity of use during class (1 to 5)	1.97	0.045	−0.026	0.049
DV2: Frequency of use after class (1 to 5)	3	0.063	−0.03	0.056
DV3: Better use of features after class (1 to 5)	3.048	0.186	−0.087	0.164
DV4: Influence of plugin on likelihood of using the debugger (1 to 5)	3.095	0.16	−0.158	0.3
Sum score over all 4 relevant questions of the questionnaire (4 to 16)	11.119	0.455	−0.09	0.171

4.3. Interpretation

With the regression weights not being statistically significant, interpretation is limited. Additionally, the hierarchical nature of the data caused by the students working in teams was not taken into account here, further limiting conclusions. Taking these limitations into account, we consider the initial results as tentatively positive for a first step into the direction of our envisioned research: All regression weights being positive is in accordance with our expectations.

5. Conclusion and Future Work

We have introduced a tool that enables researching the effect of gamification elements on debugging behavior in the form of an IntelliJ plugin set up for conducting controlled experiments. We have performed a first experiment for which the analyzed self-report data looks tentatively positive.

With our research into the matter just at the beginning, we consider the initial results encouraging and intend in a next step to take a more in-depth look at our existing data, starting with the evaluation of the same students' objective debugger usage data.

Looking back at the performed experiment, we have observed several things that we would like to improve for future experiments: Letting the students work in teams was counter productive due to introducing a hierarchical element into the data. While no systematic confounding variables are expected to have been introduced this way, proper statistical procedure requires taking the hierarchical nature of the data into account. Such methods, however, typically rely on data sets being substantially larger than ours. For future experiments, having the students work individually would elegantly solve the problem. Additionally, the experiment took course over one day only. We did not take the short timeframe into account when setting the limits for the breakpoint- and stepper-associated achievements: Reaching the gold level for the line breakpoint achievement for example would have required a student to set 250 breakpoints which is not a realistically achievable amount in one work day. This may have artificially

limited the motivating factor of the introduced gamification element. Future research should take this into account and set appropriate limits.

Regarding our tool, we are envisioning several extensions in future work: In the short term, we intend to implement additional achievements to cover more of the functionality of IntelliJ IDEA's debugger. As a medium to long term goal, we intend to extend our platform to include additional gamification elements. We are considering streaks, a point system as well as missions as possible next research avenues. The combination of several gamification elements can also be evaluated. The platform is also extensible for other learning goals derived from software engineering education, for example refactoring and testing. While testing by itself is already covered in the research by Straubinger and Fraser (2024), it may still be interesting to research in combination with other software engineering education goals. Finally, the concept of a *placebo* version of a plugin or a different application to be tested in an experimental design can and should be transferred to other research areas within software engineering research.

6. Acknowledgments

Finn Brecher, Daniel Hinkelmann, Moritz Ole Krenzer and Davide Piacenza were involved in the implementation of the plugin.

7. References

Dubois, D. J., & Tamburrelli, G. (2013, August). Understanding gamification mechanisms for software development. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering* (p. 659–662). ACM. doi: 10.1145/2491411.2494589

Ko, M., Bose, D. B., Chowdhury, H. A., Seyam, M., & Brown, C. (2023). Exploring the barriers and factors that influence debugger usage for students. In *2023 ieee symposium on visual languages and human-centric computing (vl/hcc)* (p. 168-172). doi: 10.1109/VL-HCC57772.2023.00027

Perscheid, M., Siegmund, B., Taeumel, M., & Hirschfeld, R. (2016, January). Studying the advancement in debugging practice of professional software developers. *Software Quality Journal*, *25*(1), 83–110. doi: 10.1007/s11219-015-9294-2

Porto, D. d. P., Jesus, G. M. d., Ferrari, F. C., & Fabbri, S. C. P. F. (2021, March). Initiatives and challenges of using gamification in software engineering: A systematic mapping. *Journal of Systems and Software*, *173*, 110870. doi: 10.1016/j.jss.2020.110870

Ren, W. (2023, December). Gamification in test-driven development practice. In *Proceedings of the 2nd international workshop on gamification in software development, verification, and validation* (p. 38–46). ACM. doi: 10.1145/3617553.3617889

Souza, M. R. d. A., Constantino, K. F., Veado, L. F., & Figueiredo, E. M. L. (2017, November). Gamification in software engineering education: An empirical study. In *2017 ieee 30th conference on software engineering education and training (csee&t)* (p. 276–284). IEEE. doi: 10.1109/cseet.2017.51

Straubinger, P., & Fraser, G. (2024). Improving testing behavior by gamifying intellij. In *Proceedings of the ieee/acm 46th international conference on software engineering.* New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3597503.3623339

Yang, S., Baird, M., O'Rourke, E., Brennan, K., & Schneider, B. (2024, November). Decoding debugging instruction: A systematic literature review of debugging interventions. *ACM Transactions on Computing Education*, *24*(4), 1–44. doi: 10.1145/3690652

Appendix

Table 2 – Evaluated Questions from the Questionnaire

DV1: Intensity of use during class	Corresponding Question with Scale (from German Questionnaire)
Original Question (German)	Nach Ihrer eigenen Einschätzung: Wie intensiv haben Sie den Debugger während dieses Programmierpraktikumsblocks eingesetzt?
Original Scale (German) with assigned values	sehr wenig (1), eher wenig (2), neutral (3), eher viel (4), sehr viel (5)
Translated Question	In your own estimation, how intensely did you use the debugger during this block of the class?
Translated Scale with assigned values	very little (1), rather little (2), neutral (3), rather much (4), very much (5)
DV2: Frequency of use after class	**Corresponding Question with Scale (from German Questionnaire)**
Original Question (German)	Nach dieser Praktikumseinheit werde ich den Debugger häufiger verwenden als vorher.
Original Scale (German) with assigned values	stimme gar nicht zu (1), stimme nicht zu (2), neutral (3), stimme zu (4), stimme voll und ganz zu (5)
Translated Question	After the class, I am going to use the debugger more frequently than before.
Translated Scale with assigned values	completely disagree (1), disagree (2), neutral (3), agree (4), completely disagree (5)
DV3: Better use of features after class	**Corresponding Question with Scale (from German Questionnaire)**
Original Question (German)	Nach dieser Praktikumseinheit werde ich die Möglichkeiten des Debuggers besser ausnutzen als vorher.
Original Scale (German) with assigned values	stimme gar nicht zu (1), stimme nicht zu (2), neutral (3), stimme zu (4), stimme voll und ganz zu (5)
Translated Question	After this class, I'll make better use of the debugger's features than before.
Translated Scale with assigned values	completely disagree (1), disagree (2), neutral (3), agree (4), completely disagree (5)
DV4: Influence of plugin on likelihood of using the debugger	**Corresponding Question with Scale (from German Questionnaire)**
Original Question (German)	Wie verändert das Plugin, das Sie am Anfang des ersten Tages des Blocks installieren sollten, für Sie die Wahrscheinlichkeit den Debugger zu benutzen?
Original Scale (German) with assigned values	Es verringert stark die Wahrscheinlichkeit. (1), Es verringert eher die Wahrscheinlichkeit. (2), neutral (3), Es erhöht eher die Wahrscheinlichkeit. (4), Es erhöht stark die Wahrscheinlichkeit.
Translated Question	How does the plugin you were asked to install [...] affect your likelihood of using the debugger?
Translated Scale with assigned values	It greatly reduces the likelihood. (1), It rather reduces the likelihood (2), neutral (3), It rather increases the likelihood. (4), It greatly increases the likelihood. (5)

A Text Foraging Perspective on Processing Code

Teresa Dreyer
PLTlab.de
Marburg University
tdreyer@mathematik.uni-marburg.de

Jan Tünnermann
Charlotte Fresenius University
of Psychology
jan.tuennermann@charlotte-fresenius-uni.de

Christoph Bockisch
PLTlab.de
Marburg University
bockisch@acm.org

Abstract

This paper outlines research aimed at investigating similarities and differences between human search patterns in ordinary texts and in code through the lens of the foraging paradigm. Visual foraging studies transfer methods from behavioral ecology to the study of human search. Information foraging theory applies these methods to studying human interaction with information technology. The perspective of information foraging in code navigation and comprehension was explored in previous research. Because code is a special form of text, it should be expanded by the text dimension. Due to its highly formalized structure with non-linear control flow, code resembles a natural foraging scenario more than regular text. We are therefore interested in how human search patterns in code compare to those found for normal text. We outline an experimental approach to that end.

1. Introduction

When programmers interact with code, they spend a substantial amount of time with reading and searching through code; their own and other developers' code as well as code generated by generative AI or coding agents. This includes simple searches, such as searching for a specific term by manually scrolling through the code or by skipping through candidates using the IDE's "find next" feature. However, many search tasks in program code are more complex: The implementation of one concern is often distributed over multiple program locations or files. As a consequence, tasks focused on gaining an overview and understanding of the code, such as code navigation, program comprehension, debugging, or code audits require complex searches, including multiple search targets and jumping between different code locations or files. Such tasks have in common that there is often uncertainty regarding whether and how many of the desired pieces of information might be present at the current location.

Three observations primarily motivate our work:

Observation 1: The complex code search scenario with multiple uncertain targets in distributed environments resembles the challenge animals face when foraging for food in a patchy environment, a research topic addressed in behavioral ecology (Charnov, 1976).

Observation 2: Because code is a special form of text, insights from text processing can be transferred to understanding how programmers process code. But program code has special structural features not found in common text, which may may pose additional challenges and opportunities.

Observation 3: The presentation of program code is often visually enhanced. Visual enhancement can benefit from knowledge about human visual perception and attention guidance, topics with a long tradition in psychological research.

Visual foraging has been extensively studied and connections to text and code processing have been investigated. However, a comprehensive synthesis of both perspectives that also takes into account

visual enhancements aimed at understanding how programmers navigate and search through source code – and at identifying ways to better support them in this process – remains an open research challenge. The present paper provides a brief overview of the related literature and proposes an empirical research agenda to systematically investigate these interactions and inform the design of more effective developer support tools.

2. State of the Art and Problem Statement

Human visual search has been extensively studied in cognitive psychology (Wolfe, 2020). Research in this domain often focuses on finding a target object (often abstract shapes) among several "distractors" and investigates the impact of the stimuli's low-level visual properties and the searchers' top-down knowledge. This paradigm could be considered a model for the simple code searches mentioned above. When turning to the more complex search tasks associated with program comprehension, visual foraging seems to be a more appropriate model. This can be illustrated when mapping code search to an (animal) food foraging example: A code environment (a file, a module, or just some chunk of code) resembles a berry bush. It contains multiple relevant pieces of information, the berries, but also a lot of clutter, the leaves and other greenery. The clutter consists of code and comments that are not relevant to the current search task. The berry foragers (programmers) are motivated to search efficiently in order to fill their bellies quickly (complete the complex search task).

The foraging analogy has been applied to multi-target visual search, where human participants have to search for multiple instances of simple visual targets (such as shapes) among distractors. This allows not only to investigate search efficiency with regard to visual target and distractor features, but also a decisional component: when to leave one patch (search environment) to continue searching in the next environment (Wolfe, 2013; Grössle, Schubö, & Tünnermann, 2023). The foraging paradigm has also been applied to search in common text. Within the area of text processing research, text skimming, the act of quickly reading a text to get an overview over its contents, is of particular interest to us because it resembles quickly looking through an unfamiliar code project to gain a general understanding. In the text skimming context, foraging has been shown to be a suitable approach: Research by Duggan and Payne (2009) has investigated text skimming as a technique of foraging for relevant information in text. Their results indicate that readers are effective at foraging for relevant information in text when under time pressure and suggest that readers aim to maximize useful information while minimizing time and search effort. Chi, Gumbrecht, and Hong (2007) took the perspective that text processing in general moves in the direction of skimming and found tentative evidence that visual markups such as highlights are effective at directing readers' attention to relevant text sections. This suggests that such markups improve readers' foraging for information.

A more general perspective is provided by Information foraging theory, which applies a foraging perspective to humans' search for information (Pirolli & Card, 1995, 1999). Such research has demonstrated that effective information foraging depends on strong, meaningful cues ("information scent") that help users quickly predict where valuable information is located. Information foraging theory has inspired research in several code-processing related domains, among others code navigation and debugging. The approach has been shown to be suitable for such tasks: Research by Niu, Mahmoud, and Bradshaw (2011) in the context of code navigation graphs indicates that foraging principles are applicable to code navigation which is one of our code comprehension scenarios. In the area of another of our code comprehension scenarios, debugging, Lawrance et al. (2013) presented an information-foraging-based theory on how programmers navigate code during debugging. Based on information scents, they introduced a model for programmers' code navigation behavior in a debugging task which they found to be a good predictor, further strengthening our proposed approach.

The approaches discussed above have in common that optimality can often be quantified using optimal foraging theory (OFT; Charnov, 1976) a framework from behavioral ecology to model animal food foraging. A central prediction of OFT is that foragers optimize their behavior according to marginal value theorem (MVT) by leaving a patch when the instantaneous yield from that patch drops below the

average yield experienced in the overall environment.

We are interested in *comparing how programmers search for information in code versus how humans search for information in common text, with the goal to develop better strategies for gaining an overview and understanding of code*. In particular, we expect MVT-based predictions for code search to outperform those for text search. The rationale is that the code searching scenario more closely resembles natural food search: Compared to common text, code is less linear and often distributed in "patchy" environments (files, modules, etc.) and features such a syntax highlighting in conjunction with specific keywords and symbols provide attention guidance comparable colorful berries. Mismatches between MVT predictions for code and text search can provide cues for how to improve coding environments.

To approach this goal, we derive the following research questions:

RQ1: Does the marginal value theorem predict human search behavior better for code than for ordinary text?

RQ2: Under which conditions does code search behavior resemble text search behavior and under which conditions is it different?

3. Research Proposal

To investigate RQ1 and RQ2, we propose a series of controlled experiments.

Experiment 1: A baseline For the first experiment, we would like to create a baseline foraging scenario where code and text are expected to be processed similarly. We will look at the search for typographical errors in text snippets versus the search for syntax errors in code snippets with and without typical syntax highlighting (see figure 1). For all variants, the snippets are distributed over the screen in order to keep the scenarios both similar and close to a natural foraging task. Clicking the "Next" button moves the participant to the next foraging patch. The participants are instructed to find and click on as many errors as possible in a fixed amount of time. To this end, they are allowed to skip into a new, error-filled patch whenever they feel that searching the current one gets too slow as more and more errors are already discovered.

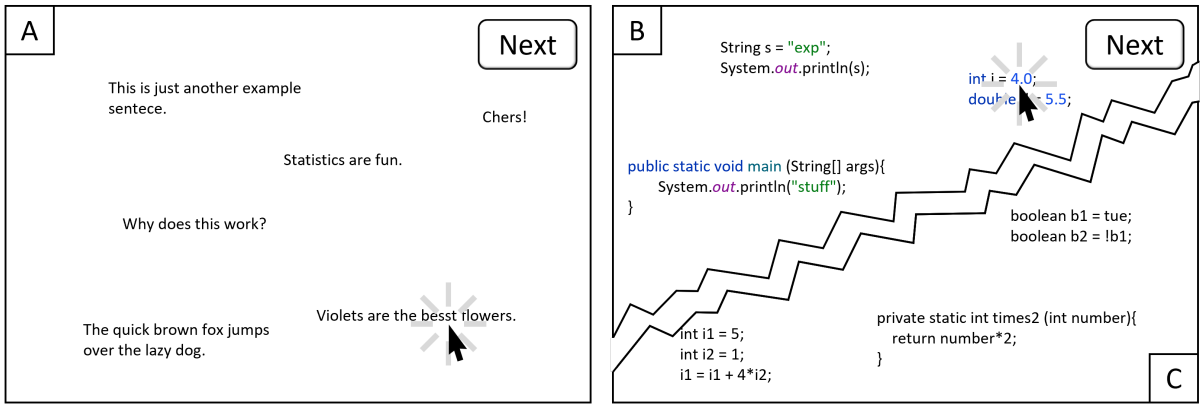


Figure 1 – Exemplary setup for experiment 1.

Over the course of the experiment, the participant is faced with three types of screen setup with an associated search task ("trial"). Trials of type A represent the search for typographical errors in text snippets. Trials of type B and C represent the search for syntactical errors in code snippets with and without syntax highlighting, respectively. Only a few snippets per screen are shown here for readability. Actual patches would be crowded with more elements.

While these tasks are not a perfect mapping, they are similar enough for our purposes. Both tasks are on a relatively superficial level and involve little semantic processing. Any differences between code and text processing detected in this experiment would likely result from the low-level visual differences (e.g.

highlighting). More interestingly, we intend to utilize the results to establish a baseline to be compared to tasks that model more of the distributed nature of program code, the goal of Experiment 2.

Experiment 2: Realistic scenarios Experiment 2 delves into more realistic scenarios: Longer code pieces as well as longer text facilitate more complex search tasks. In this scenario, differences between text and code processing might be more pronounced. Due to the distributed nature of program code, we expect the code search tasks to be better predictable by the marginal value theorem. A stronger difference between code and text in Experiment 2 compared to Experiment 1 could highlight the impact of these structural differences beyond the low-level perceptual differences. In order to eliminate low-level visual differences as the main factor, for text and code both, the deviation from the baseline experiment can be considered.

Variations: These starting experiments offer several connecting points for follow-ups and variations. For instance, different forms of highlighting could be explored. Text highlighting is a popular technique that is used to draw the user's attention to specific text sections or keywords. Text highlighting techniques can also be used for code (Dreyer, Michel, & Bockisch, 2026) which would allow for an interesting very direct comparison. On the other hand, developers are extremely familiar with syntax highlighting as it is not only the default in integrated development environments but also for displaying code snippets in other contexts. Comparing code snippets with syntax highlighting to text snippets with similar structural elements could also be very interesting.

Another factor that is likely to impact performance and which might interact with forms of code presentation is the experience level. Programming novices might behave entirely differently from programming experts. Investigating the search behavior of people without any programming experience could also be interesting in a comparison to people with programming experience.

4. Conclusions and Future Work

We have sketched two experiments that start an investigation of how the optimal foraging framework can be employed to specifically study how programmers search in source code, a specialized form of text search. Extending earlier approaches, we are particularly interested in the interaction of visual perception and informational value. We believe that this approach is insightful and might inform future work on programming tools.

5. References

Charnov, E. L. (1976, April). Optimal foraging, the marginal value theorem. *Theoretical Population Biology*, *9*(2), 129–136. doi: 10.1016/0040-5809(76)90040-x

Chi, E. H., Gumbrecht, M., & Hong, L. (2007). Visual foraging of highlighted text: An eye-tracking study. In *Human-computer interaction. hci intelligent multimodal interaction environments* (p. 589–598). Springer Berlin Heidelberg. doi: 10.1007/978-3-540-73110-8_64

Dreyer, T., Michel, J., & Bockisch, C. (2026, March). Towards transferring text comprehension techniques to program comprehension. In *Companion proceedings of the 10th international conference on the art, science, and engineering of programming* (p. 18–25). ACM. doi: 10.1145/3801119.3801126

Duggan, G. B., & Payne, S. J. (2009). Text skimming: The process and effectiveness of foraging through text under time pressure. *Journal of Experimental Psychology: Applied*, *15*(3), 228–242. doi: 10.1037/a0016995

Grössle, I. M., Schubö, A., & Tünnermann, J. (2023). Testing a relational account of search templates in visual foraging. *Scientific Reports*, *13*(1), 12541. doi: 10.1038/s41598-023-38362-9

Lawrance, J., Bogart, C., Burnett, M., Bellamy, R., Rector, K., & Fleming, S. D. (2013). How programmers debug, revisited: An information foraging theory perspective. *IEEE Transactions on Software Engineering*, *39*(2), 197-215. doi: 10.1109/TSE.2010.111

Niu, N., Mahmoud, A., & Bradshaw, G. (2011, May). Information foraging as a foundation for code navigation (nier track). In *Proceedings of the 33rd international conference on software engineering* (p. 816–819). ACM. doi: 10.1145/1985793.1985911

Pirolli, P., & Card, S. (1995). Information foraging in information access environments. In *Proceedings of the sigchi conference on human factors in computing systems - chi '95* (p. 51–58). ACM Press. doi: 10.1145/223904.223911

Pirolli, P., & Card, S. (1999, October). Information foraging. *Psychological Review*, *106*(4), 643–675. doi: 10.1037/0033-295x.106.4.643

Wolfe, J. M. (2013, January). When is it time to move to the next raspberry bush? foraging rules in human visual search. *Journal of Vision*, *13*(3), 10–10. doi: 10.1167/13.3.10

Wolfe, J. M. (2020). Visual search: How do we find what we are looking for? [Journal Article]. *Annual Review of Vision Science*, *6*(Volume 6, 2020), 539-562. doi: 10.1146/annurev-vision-091718-015048

Towards Supporting Novice Programmers' Web Search Skills for Using Unfamiliar Libraries and APIs

Mehmet Arif Demirtaş, Jinyoung Hur, Alex Atcheson, Claire Zheng, Kathryn Cunningham
University of Illinois Urbana-Champaign
Urbana, Illinois, USA
{mad16,jhur10,samuela7,clairez7,katcun}@illinois.edu

Abstract

Programmers often rely on online help-seeking resources such as developer forums and language documentation when working with libraries and APIs. Finding relevant resources requires them to formulate search queries that effectively represent their information needs. However, factors influencing how novice programmers write search queries are not well documented.

To better understand how to support the development of search skills in novices, we investigate how novice programmers write search queries while learning about an unfamiliar data visualization library. We further examine how instructional design may influence how students formulate queries. We collected queries written by students in an introductory programming class after completing a self-directed tutorial on the Seaborn library in Python. Given a Seaborn-generated data visualization to recreate or modify, students were asked to formulate a search query they would use to find additional information on how to do so. In this work-in-progress paper, we present an exploratory study of the factors that shape the queries formulated by novice programmers. We further report insights from a subset of the data on patterns in the queries and search results as a step toward understanding how to support novice programmers in developing effective search query strategies for online help-seeking.

1. Introduction and Related Work

Modern programming projects rely on the integration of specialized libraries and APIs (application programming interfaces). Programmers commonly use online help-seeking resources to effectively use the latest libraries (Cai & Guo, 2019). Studies and surveys have reported that these resources include developer forums (e.g., Stackoverflow), online tutorials, and documentation pages (Alghamdi et al., 2025), and increasingly, AI assistants (JetBrains, 2024). Prior studies on API learning noted that performing web searches is a key part of using and understanding an API (Heinonen, 2024; Thayer et al., 2021).

Well-formulated search queries are key to retrieving relevant results. Prior literature has shown that even professional programmers can struggle to ask efficient questions for retrieving relevant information. For instance, Duala-Ekoko & Robillard (2012) found that developers use API elements as terms in their queries and that they encounter difficulties when searching for unfamiliar concepts that is difficult to relate to familiar API elements. Kelleher & Brachman (2023) observed that programmers were able to find information relevant to the fundamental elements of an API, but struggled to form effective debugging-related questions when errors arose during project work. Gao et al. (2020) examined web search as the initial step in obtaining and organizing knowledge about an API, and observed that ineffective searches can trigger longer search sessions and hinder learning by increasing cognitive load. Extended search sessions may also lead to users getting discouraged and losing confidence in the results (Jiang et al., 2014). In an analysis of query logs across search engines, White et al. (2009) found that users with domain expertise in computer science performing domain-relevant searches were more likely to include domain-specific terms in their queries, wrote longer queries on average, and had higher rates of success in retrieving relevant results compared to non-experts. Yet, it remains unclear how novice programmers develop the skill to write effective queries despite limited domain expertise. Notably, all of these studies focused on individuals with prior programming experience. The challenges identified in these works may be amplified for novice programmers who, in addition to learning to use APIs, are also learning to program, the challenges of which are many and well-documented (Robins, 2019).

In the current study, we investigate how novice programmers write search queries in the context of an

information-seeking task framed around an unfamiliar library and how instructional design decisions influence query formulation. To do so, we collected queries written by students in an undergraduate-level introductory programming course for non-CS majors. During the study, students were presented with three data visualizations generated using the Seaborn library in Python and, for each task, asked to write a query they would use to find relevant information on modifying or recreating elements of the visualizations. Prior to completing the search tasks, students completed one of two self-directed tutorials. One tutorial followed the structure of a popular online resource, covering individual Seaborn API functions in isolation, while the other was organized around subgoal labels–descriptions of the goals of short code segments–to cover authentic data visualization examples that drew on several functions at once. Subgoal labels have been argued to support students' ability to explain code by providing relevant language (Cunningham et al., 2021), and have been shown to support learning outcomes for students with minimal programming experience by reducing focus on surface-level problem details (Margulieux et al., 2020). However, reducing focus on problem details may discourage familiarization with language syntax and hinder effective searching, as there is evidence to suggest that referencing API elements in queries leads to effective searches (Duala-Ekoko & Robillard, 2012).

To better understand how tutorial organization may influence search behavior, we conducted an exploratory analysis comparing queries formulated by novice programmers in the context of an information-seeking task. Specifically, we posed the following two research questions:

- RQ1: What patterns can be observed in search queries written by novice programmers using an unfamiliar library?
- RQ2: How do teaching approaches influence the way novice programmers write search queries?

2. Methodology

We collected search queries in a randomized controlled trial in an introductory programming course for non-CS majors in the Spring of 2026.

2.1. Experimental Design

The study took place during hands-on lab sessions for the course. Students did not receive prior instruction on the Seaborn library[1]. Instead, they first completed a self-directed tutorial introducing several concepts that were curated with instructor approval: creating two types of visualizations (bar plot and box plot), displaying multiple visualizations from the same dataset in a grid, and customizing the colors and labels of a plot. Students were randomly assigned to one of the two tutorial conditions. The tutorials covered the same content, but used different instructional approaches.

2.1.1. Tutorial Content

One group of students completed a tutorial designed with a *bottom-up* approach reflecting the structure of popular online tutorials. This tutorial was adapted from an online resource on visualizing data with Seaborn (Ali, 2023). The tutorial began with an overview of the activity and highlighted individual library functions in subsequent pages, covering the functions required for achieving the learning objectives. This organization emphasized API elements, such as function names, on the tutorial pages.

Students who were not assigned to the bottom-up condition completed the *top-down* condition. In the top-down condition, the tutorial utilized *subgoal labels*. This tutorial was modeled after *purpose-first tutorials* (Demirtas et al., 2026), which uses subgoal labels to emphasize the purpose of authentic code examples for teaching non-CS majors about computing applications. To this end, the top-down tutorial began with two examples of data visualizations showcasing the use of several API functions. These examples were annotated with subgoal labels, breaking down the larger programs into smaller "chunks". Students studied these subgoal chunks in subsequent tutorial pages.

Both tutorials included an identical set of interactive exercises for students to practice. After completing the tutorials, students were presented with a post-assessment to measure their understanding of Seaborn.

[1]Earlier in the course, students received instruction on the basics of `matplotlib`, another Python visualization library.

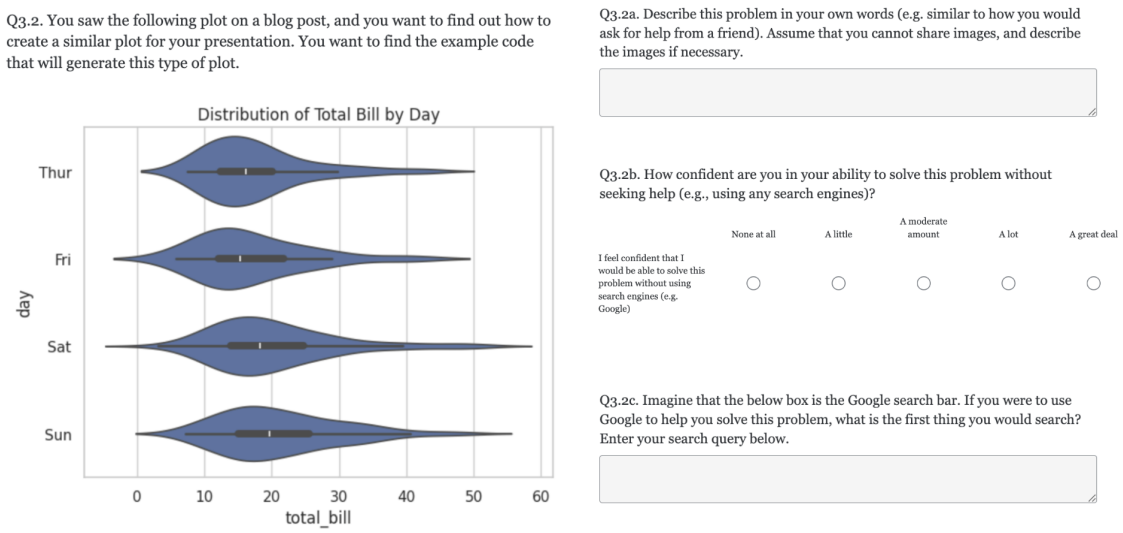


Figure 1 – The search task presented to students. The query data is collected from answers to the last textbox (Q3.2c).

2.1.2. Search Tasks

Queries were collected for three search tasks, which were completed during the post-assessment. For each task, students were presented with a scenario and a target data visualization. The scenarios required students to modify existing code or write code from scratch to replicate the target data visualization. The scenarios were written without technical instructions or API terminology, requiring students to infer the necessary modifications from the visualization. For each scenario, students were asked to (i) describe the task in their own words, (ii) rate their confidence in their ability to complete the task without searching for resources, and (iii) write the query they would use in an initial attempt to find relevant resources for solving the task. Queries were collected on a survey; students did not execute their searches.

Search tasks varied in difficulty and the type of information provided. In this work-in-progress, we focus our analysis on one of the three tasks. This task included a target data visualization of a violin plot, a plot type that was not covered in the tutorial. Students were asked to search for information that would help them replicate the provided visualization. The task instructions are shown in Figure 1.

2.2. Exploratory Analysis

Our analysis focuses on the search task centered around the violin plot because it requires *far transfer* (Barnett & Ceci, 2002): in other words, students must apply their data visualization knowledge in an unfamiliar context. Challenges with far transfer have been enumerated in prior work about API learning, which found that programmers struggled more with finding information about new API elements than with exploring details of API elements with which they were already familiar (Duala-Ekoko & Robillard, 2012). Of the plot types covered in the tutorial, the type most similar to a violin plot is a box plot. The bottom-up tutorial introduced box plots in general terms as a way of comparing the distribution of one or more variables across different categories. The top-down tutorial with subgoal labels emphasized the *purpose* of box plots in specific scenarios, introducing it in an example about comparing restaurants with consistent ratings across time. Given this difference, we were interested in knowing whether students exposed to different tutorial organizations use contrasting vocabulary to describe a new visualization type and achieve varying success at retrieving relevant resources. We present this preliminary analysis as an exploration of student querying behavior for a concept that is distant from the examples with which they may already be familiar.

2.2.1. Data Pre-Processing and Search Results

The collected dataset included queries from 372 students. 178 students had completed the bottom-up tutorial, and 194 students had completed the top-down tutorial that uses subgoal labels.

Search engines rank the pages in search results to prioritize pages that are most likely to be useful to the users, and users are more likely to visit pages that are ranked higher on the results pages. Therefore, we only look at the first 10 pages from the search results of each query. We collected these pages by executing a Google search for each query in our dataset using SerpAPI,[2]. For each result, we extracted the title, link, text snippet, and list position. To reduce the influence of misspellings, we also extracted the Google-suggested query when one was provided–that is, the query used by Google for which the results were actually retrieved–and used it in place of the originally authored query in our later analysis. For word-level analyses, we removed all punctuation from the queries and applied stemming to capture the most general form of each word.

2.2.2. Uniqueness of Retrieved Results

We first examine how many unique pages were retrieved by students in their search results. There were 1468 unique results retrieved by 372 queries. Of these results, 293 were retrieved by queries from both top-down and bottom-up conditions. On the other hand, 622 results only appeared for queries from the top-down condition, while 553 results only appeared for queries from the bottom-up condition, suggesting that collected queries could retrieve a diverse set of search results. Even though there are many pages unique to one condition among all search results, most students would see pages from a smaller set of search results. When we examine the pages that appear as the top search result, we see that 71% of 372 queries retrieved a page that could be retrieved by a query from either condition. Considering that users are more likely to check the top result compared to the rest of their search results (Abualsaud & Smucker, 2019), the tutorial conditions might be less impactful on search success than expected.

2.2.3. Relevance of Retrieved Results

We use the relevance of retrieved search results to the search task as a measure of success. We define a relevant search result for this task as a page with a reference to a violin plot. To this end, we analyzed the links that would be displayed on the first page of Google search results for each query to measure how successful these queries would be at getting relevant information.

In our preliminary analysis, we looked at the top 30 most common pages in search results for both conditions. In both conditions, a majority of these pages were not relevant to the task: 40% of pages retrieved by top-down queries (12 out of 30) and 30% of pages retrieved by bottom-up queries (9 out of 30) contained a reference to violin plots.

Moreover, most of the students did not have one of these relevant pages in their top search results. 24.5% of students had one of the relevant pages as their top result, and 56.7% of students had at least one relevant page within their top 10 search results. While it is possible that students would retrieve relevant pages not captured by our analysis, this finding provides a lower bound that shows how difficult searching for relevant information can be for students.

23.9% of queries from the top-down condition had one of these relevant pages as the top search result. In contrast, only 16.5% of queries from the bottom-up condition had one of these relevant resources as the top result. However, 47.5% of top-down and 47.8% of bottom-up condition queries had one or more of these relevant results on the first page of search results.

2.2.4. Patterns in Search Queries

To explore the patterns in novice search queries, we built three exploratory linear regression models. Model 1 predicts the query length as the dependent variable. Models 2 and 3 predict the frequencies of API elements as search terms and data visualization jargon as search terms, respectively. For our analysis, the list of API elements includes terms from the code examples (e.g., classes, functions, parameters) originating from the Seaborn documentation related to our topics, and the list for data visualization jargon includes terms used in the explanations on the same pages. Then, each search query is assigned a relative frequency per list, defined as the number of the terms from a list normalized by the length of

[2] https://serpapi.com/

the query. These dependent variables were selected to capture any differences due to the instructional design in the bottom-up tutorial, which highlights API elements, and the top-down tutorial, which describes code at a higher level with subgoal labels. Predictors for all three models are the tutorial type (bottom-up vs top-down with subgoals) and self-reported confidence in completing the given search task without web searching, in addition to the self-reported demographic information on whether the student had programming experience before the class, frequency of search engine use, and frequency of AI assistant use. Queries from two students were excluded due to missing demographic data.

In our analysis, the only significant predictor of query length was self-reported confidence in completing the given search task without web searching ($p = .018, R^2 = .023$). Students who reported a higher confidence in completing the task without support used fewer words. There were no significant predictors for the frequency of code terms ($R^2 = .016$) and jargon terms ($R^2 = .009$). Notably, the tutorial condition was not a significant predictor for any of the dependent variables, indicating that students' search behavior for a new library may not be directly influenced by instruction on the specific library. Moreover, our models had low R^2 values, suggesting that the factors we explored might not account for the variability in students' search term usage (see Appendix A).

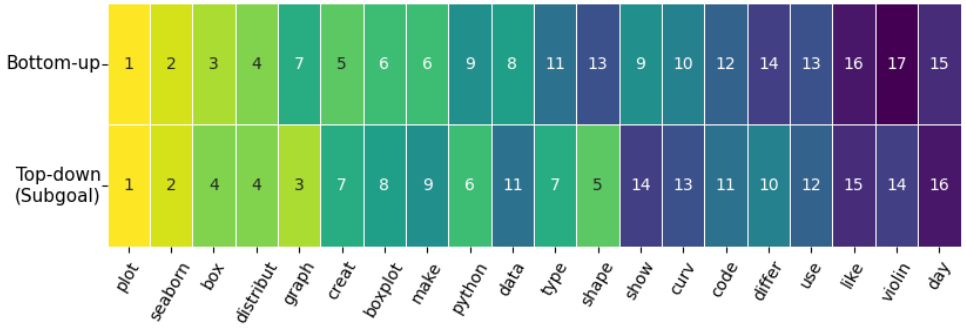


Figure 2 – Top 20 search terms by tutorial type. Terms are ordered from most common (left) to least common (right) for the full dataset. The numbers on the top row represent the rank of each term for queries submitted by students who completed the bottom-up tutorial, and the bottom row represents the ranks of terms in queries submitted by students who completed the top-down/subgoal tutorial.

Commonly Used Search Terms. In addition to the frequencies of types of terms and query lengths, we look at the 20 most common terms for queries submitted by students who completed the tutorials (Figure 2). Both conditions had the same set of words in their top 20, but some words (such as 'graph', 'shape', and 'type') were more common in the top-down/subgoal condition, and others (such as 'create' and 'data') were used more commonly in the bottom-up condition. However, we were not able to observe any consistent trends between the use of these terms and the relevance of the retrieved search results.

3. Discussion

This work presents our ongoing work on how novice programmers write search queries for unfamiliar libraries. Our initial findings suggest that instructional design choices may influence students' efficiency at retrieving relevant search results to some extent. However, our preliminary analysis of patterns in search queries did not reveal significant differences in query lengths, frequencies of API elements and domain terminology as keywords, or commonly used search terms. These results might imply that the success of search results is sensitive to minor changes in search queries, motivating further investigation. For instance, exploring the effects of word ordering or comparing frequencies of n-grams as opposed to individual words in search queries may reveal patterns that affect the success of students at search tasks. One aspect of our work is that data visualization tasks may require three types of knowledge: programming knowledge, knowledge about data visualization types, and knowledge about the particular data visualization library (Seaborn). Future work can consider how the learning process differs between libraries that require different types of prior knowledge.

What makes a search query 'successful' is still an open question, as previous work mainly focuses on programmers' search behaviors in think-aloud sessions as opposed to assessing individual search queries. These studies (e.g., Gao et al. (2020)) evaluate the effectiveness of a search query based on whether the programmer conducts a new search immediately after the current one, or whether they stay on the pages they access as a result of the search query. However, this might be overly strict, as studies on web searches in other contexts, such as seeking medical information (White & Horvitz, 2009), indicate that an effective search session may include several successive searches that are informed by the results of the previous query searches. Moreover, novices may struggle to recognize relevant information in complex tasks even when it is present in a resource (Kiani et al., 2019), adding another layer of complexity to whether a query is successful. Thus, we believe that evaluating programmers' web search skills on a query-level and exploring the patterns in these queries is a promising avenue for the development of instructional interventions.

Increasing prevalence of AI assistants for information seeking raises two questions about the importance of web search skills in next-generation programming environments. First, students may defer to AI assistants for search tasks, as opposed to traditional search engines. How users decide between web search and generative AI is being actively investigated (Zhou & Li, 2026), and whether either option provides benefits for learning or productivity when working with unfamiliar libraries remains to be explored. Second, students may use AI-assisted coding environments to obtain code as opposed to seeking information that will help them write code. Even in this case, there are similarities between using AI assistants for coding and seeking information online, as discussed by Sarkar et al. (2022). In both settings, the help request starts with a query written in a natural language, and the user needs to evaluate the output and identify what parts of the output are relevant to the problem at hand. Thus, certain traditional web search skills may transfer to AI-assisted information seeking and coding environments. However, Sarkar et al. (2022) also highlights the *mixed media experience* in traditional search engines, where code snippets shared via text are one type of media out of many, such as forum discussions, videos, and images. Therefore, users might develop mechanisms that leverage several forms of information simultaneously for processing traditional search results and identifying relevant information. In contrast, Sarkar et al. (2022) describes AI assistance in coding as a *fixed media experience* where the written code and descriptions are the main form of output presented to the user. In this case, some skills developed in traditional web search might not be applicable in AI-assisted search tasks. Research on search skills should consider the skills that will be required to explain the new paradigm of AI-assisted search and coding environments.

References

Abualsaud, M., & Smucker, M. D. (2019). Patterns of search result examination: Query to first action. In *Proceedings of the 28th acm international conference on information and knowledge management* (p. 1833–1842). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi-org.proxy2.library.illinois.edu/10.1145/3357384.3358041 doi: 10.1145/3357384.3358041

Alghamdi, S. S., Bull, C., & Kharrufa, A. (2025). *Designing for self-regulation in informal programming learning: Insights from a storytelling-centric approach.* Retrieved from https://arxiv.org/abs/2507.22671

Ali, M. (2023). *Python seaborn tutorial for beginners: Start visualizing data.* Retrieved 2026-05-10, from https://www.datacamp.com/tutorial/seaborn-python-tutorial

Barnett, S. M., & Ceci, S. J. (2002). When and where do we apply what we learn?: A taxonomy for far transfer. *Psychological Bulletin*, *128*(4), 612–637. doi: 10.1037/0033-2909.128.4.612

Cai, C. J., & Guo, P. J. (2019). Software developers learning machine learning: Motivations, hurdles, and desires. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (p. 25-34). doi: 10.1109/VLHCC.2019.8818751

Cunningham, K., Ericson, B. J., Agrawal Bejarano, R., & Guzdial, M. (2021, May). Avoiding the Turing Tarpit: Learning Conversational Programming by Starting from Code's Purpose. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1–15). Yokohama Japan: ACM. doi: 10.1145/3411764.3445571

Demirtas, M. A., Zheng, C., & Cunningham, K. (2026, February). Teaching Authentic Programming Applications to Novices: Purpose-first Tutorials in a General Education Computing Course. In *Proceedings of the 57th ACM Technical Symposium on Computer Science Education V.1* (Vol. 1, pp. 274–280). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3770762.3772574

Duala-Ekoko, E., & Robillard, M. P. (2012, June). Asking and answering questions about unfamiliar APIs: An exploratory study. In *2012 34th International Conference on Software Engineering (ICSE)* (pp. 266–276). Zurich: IEEE. doi: 10.1109/ICSE.2012.6227187

Gao, G., Voichick, F., Ichinco, M., & Kelleher, C. (2020). Exploring Programmers' API Learning Processes: Collecting Web Resources as External Memory. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 1–10). doi: 10.1109/VL/HCC50065.2020.9127274

Heinonen, A. (2024). Understanding apis and the software that provides them - analysis of programmers' api mental models used in programming tasks. In *Proceedings of the 35th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2024).*

JetBrains. (2024). *Python Developers Survey 2024 Results.* Python Software Foundation and JetBrains PyCharm. Retrieved 2026-05-19, from `https://lp.jetbrains.com/python-developers-survey-2024/`

Jiang, J., He, D., & Allan, J. (2014). Searching, browsing, and clicking in a search session: changes in user behavior by task and over time. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval* (pp. 607–616). doi: 10.1145/2600428.2609633

Kelleher, C., & Brachman, M. (2023, January). A sensemaking analysis of API learning using React. *Journal of Computer Languages*, *74*, 101189. doi: 10.1016/j.cola.2022.101189

Kiani, K., Cui, G., Bunt, A., McGrenere, J., & Chilana, P. K. (2019). Beyond "one-size-fits-all": Understanding the diversity in how software newcomers discover and make use of help resources. In *Proceedings of the 2019 chi conference on human factors in computing systems* (p. 1–14). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi-org.proxy2.library.illinois.edu/10.1145/3290605.3300570` doi: 10.1145/3290605.3300570

Margulieux, L. E., Morrison, B. B., & Decker, A. (2020, May). Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. *International Journal of STEM Education*, *7*(1), 19. doi: 10.1186/s40594-020-00222-7

Robins, A. V. (2019). Novice Programmers and Introductory Programming. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 327–376). Cambridge University Press. doi: 10.1017/9781108654555.013

Sarkar, A., Gordon, A. D., Negreanu, C., Poelitz, C., Srinivasa Ragavan, S., & Zorn, B. (2022, September). What is it like to program with artificial intelligence? In *Proceedings of the 33rd Annual Conference of the Psychology of Programming Interest Group (PPIG 2022).*

Thayer, K., Chasins, S. E., & Ko, A. J. (2021, January). A Theory of Robust API Knowledge. *ACM Trans. Comput. Educ.*, *21*(1), 8:1–8:32. doi: 10.1145/3444945

White, R. W., Dumais, S. T., & Teevan, J. (2009). Characterizing the influence of domain expertise on web search behavior. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining* (pp. 132–141). doi: 10.1145/1498759.1498819

White, R. W., & Horvitz, E. (2009, November). Cyberchondria: Studies of the escalation of medical concerns in web search. *ACM Trans. Inf. Syst.*, *27*(4). doi: 10.1145/1629096.1629101

Zhou, T., & Li, S. (2026). Understanding user switch of information seeking: From search engines to generative ai. *Journal of Librarianship and Information Science*, *58*(1), 696-708. Retrieved from https://doi.org/10.1177/09610006241244800 doi: 10.1177/09610006241244800

A. Linear Regression Results

Query length:

OLS Regression Results

Dep. Variable:	query_length	R-squared:	0.023
Model:	OLS	Adj. R-squared:	0.009
Method:	Least Squares	F-statistic:	1.683
Date:	Tue, 19 May 2026	Prob (F-statistic):	0.138
Time:	11:11:48	Log-Likelihood:	−1204.4
No. Observations:	370	AIC:	2421.
Df Residuals:	364	BIC:	2444.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	14.3838	1.303	11.042	0.000	11.822	16.945
C(condition)[T.pfp]	−0.5091	0.661	−0.771	0.441	−1.808	0.790
experience_binary	−0.7308	0.680	−1.075	0.283	−2.068	0.606
task_confidence	−0.8858	0.373	−2.372	0.018	−1.620	−0.151
traditional_search_frequency	−0.1599	0.340	−0.470	0.639	−0.829	0.509
llm_assistant_frequency	−0.1095	0.270	−0.405	0.685	−0.641	0.422

Frequencies of code terms:

OLS Regression Results

Dep. Variable:	code_match_norm	R-squared:	0.016
Model:	OLS	Adj. R-squared:	0.002
Method:	Least Squares	F-statistic:	1.159
Date:	Tue, 19 May 2026	Prob (F-statistic):	0.329
Time:	11:11:48	Log-Likelihood:	31.168
No. Observations:	370	AIC:	−50.34
Df Residuals:	364	BIC:	−26.85
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.2245	0.046	4.861	0.000	0.134	0.315
C(condition)[T.pfp]	0.0107	0.023	0.458	0.647	−0.035	0.057
experience_binary	0.0242	0.024	1.003	0.317	−0.023	0.072
task_confidence	0.0248	0.013	1.874	0.062	−0.001	0.051
traditional_search_frequency	0.0041	0.012	0.338	0.735	−0.020	0.028
llm_assistant_frequency	0.0062	0.010	0.647	0.518	−0.013	0.025

Frequencies of data visualization jargon terms:

OLS Regression Results

Dep. Variable:	jargon_match_norm	R-squared:	0.009
Model:	OLS	Adj. R-squared:	−0.005
Method:	Least Squares	F-statistic:	0.6631
Date:	Tue, 19 May 2026	Prob (F-statistic):	0.652
Time:	11:11:48	Log-Likelihood:	48.390
No. Observations:	370	AIC:	−84.78
Df Residuals:	364	BIC:	−61.30
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.2535	0.044	5.749	0.000	0.167	0.340
C(condition)[T.pfp]	0.0178	0.022	0.796	0.427	−0.026	0.062
experience_binary	0.0258	0.023	1.121	0.263	−0.019	0.071
task_confidence	0.0140	0.013	1.107	0.269	−0.011	0.039
traditional_search_frequency	0.0006	0.012	0.053	0.957	−0.022	0.023
llm_assistant_frequency	−2.536e−05	0.009	−0.003	0.998	−0.018	0.018

Evaluating and Adapting the ATTA for Computational Thinking Skills Assessment in Higher Education: A Pilot Study

Frances Sheridan
National College of Ireland
fsheridan@ncirl.ie

Pramod Pathak
TU Dublin
pramod.pathak@tudublin.ie

Paul Stynes
National College of Ireland
paul.stynes@ncirl.ie

Keith Quille
TU Dublin
keith.quille@tudublin.ie

Abstract

Computational Thinking (CT) Skills are growing in importance as generative AI influences the skills required for a modern workforce. A review of the literature shows a concentration of CT research at the primary and secondary education levels, with a clear research gap and limited availability of practical, validated CT assessment tools for use in Higher Education. The Algorithmic Thinking Test for Adults is a 20-item validated scale targeting several CT components with an execution time of approximately 70 minutes. A pre-pilot study using the ATTA highlighted challenges related to timing and execution time when zero responses were collected. A revised pilot study with 125 invited participants achieved a 44% response rate. Participant feedback coupled with completion rates and item non-response rates supported a need for a reduction of the 20-item scale to improve feasibility in this context. This research aims to evaluate the suitability, reliability, and relevance of the Algorithmic Thinking Test for Adults for use with undergraduate computing students and explores adaptations to improve its feasibility in time-constrained classroom settings. Iterative item removal and exploratory factor analysis produced a shortened 9-item scale with acceptable reliability indicated by Cronbach's α of .764. Findings provide preliminary evidence that a shortened ATTA with reduced execution time may improve feasibility while retaining acceptable internal consistency.

1. Introduction

As GenAI influences the skills required for a modern workforce, educators must consider the learning challenges that this presents (Romero, 2024). Researchers have now begun to consider prompt engineering a new 21st Century skill (Federiakin, Molerov, Zlatkin-Troitschanskaia, & Maur, 2024), and emphasise the importance of CT skills for AI literacy (Celik, 2023). A recent Blog post for Communications of the ACM highlights this relationship between CT skills and Prompt Engineering for computing students in Higher Education and emphasises that "These interactions between prompt engineering and computational thinking underscore the importance of computational thinking in the era of GenAI"(Erez, Mike, & Hazzan, 2024). Much of the research on CT skills to date has focused on curriculum, frameworks and interventions to support the development of CT skills (Lockwood & Mooney, 2017). However, in order to fully appreciate the impact of frameworks and interventions, we need valid and reliable assessment tools. A review of the literature shows a concentration of CT research at the primary and secondary education levels, with a clear gap in research and limited availability of practical, validated assessment tools for use in Higher Education (Haseski & Iliç, 2019; Tang, Yin, Lin, Hadad, & Zhai, 2020; Cruz Castro, Shoaib, & Douglas, 2021; Sheridan, Stynes, Pathak, & Quille, 2024). For the purpose of this research, a review of the literature was performed aiming to identify the full range of tools available to assess CT skills at all levels of education (Sheridan et al., 2024). Tools were reviewed with an emphasis on evaluating the reliability of the tools, as well as their suitability and adaptability for use in this research. Of the tools available to assess CT Skills in Higher Education, the Algorithmic Thinking Test for Adults (ATTA) (Lafuente Martínez, Lévêque, Benítez, Hardebolle, & Zufferey, 2022) stood out as the most robustly validated and reliable tool available for a Higher Education context. The ATTA is a 20-item scale validated using a series of measures including two qualitative pilot studies, expert review, and a quantitative pilot involving 289 participants where participants received a cash incentive for participation. While the scale targeted several components of CT, results found algorithmic thinking to be the

single dimension underlying the test. Questions in the ATTA are presented to participants in increasing order of difficulty. The execution time for the test is approximately 70 minutes - a factor identified by the researchers as a possible limitation.

1.1. Pre-pilot Study

A Pre-pilot study was conducted in April of the 2023/2024 academic year in two Irish Higher Education institutions, National College of Ireland (NCI) and Technological University Dublin (TUD). Year 1 Computing students were invited via email to complete the survey in their own time and no incentive was provided to students in exchange for participation. A total of 300 students were invited to participate and zero responses were received. The challenges of recruiting participants for Higher Education research are many (Khatamian Far, 2018). One challenge often encountered when recruiting and retaining Higher Education students as research participants is timing (of Ontario, Cyr, Childs, & Elgie, 2013).When considering participation in research studies, students will weigh the costs and benefits of participation. Following the survey, informal verbal feedback was gathered by lecturers regarding non-participation. The informal feedback indicated that the time of year, the duration of the survey, and the lack of incentive may have been factors that impacted participation.

1.2. Pilot Study

The response rate of the pilot study further emphasises the limitation of the execution time identified during the original ATTA validation. Research on response rates indicates that rates increase when the respondent burden (the effort required to answer the questions) is low (Denscombe, 2009). Thus, the researchers decided to narrow the scope of the pilot study to one single institution where students could be allowed to complete the survey during class time. The pilot study was carried out in Semester 1 of the following academic year, 2024/25 in NCI only. A total of 125 first-year undergraduate computing students were invited to participate. The survey, distributed electronically via email, retained the same structure as before, including demographic questions, items from the MSLQ (Pintrich & De Groot, 1988), and the full 20-item ATTA (Lafuente Martínez et al., 2022). Students were given 60 minutes during class time to complete the survey. As with the previous phase, no extrinsic incentives were offered for participation. A total of 55 responses to the survey were received. This represented 44% of the cohort of students. Of these, 11 students completed the ATTA for their own experience but opted out of the study itself and the remaining 44 responses were opted into the study. In a subsequent class, students were invited to complete a short, two-question, online feedback questionnaire. students were asked if they had: completed the survey; started but not completed the survey; or did not even start the survey. They were also asked, in an open response question, to outline their reasons for non-completion. The feedback survey conducted in NCI received 66 responses. Of the 66, 44% stated that they completed the survey, 41% started but did not finish the survey, and the remaining 15% did not start the survey at all. Analysis of responses regarding non-completion found that 10 students cited the volume of questions, 16 said they did not have enough time, 10 said the questions were too difficult to answer, and a further 5 had difficulties understanding the questions themselves.

1.3. Research Goals

This research aims to:

1. Evaluate the suitability, reliability, and relevance of the ATTA as a CT assessment instrument for practical, time-constrained application in the context of a higher education computing classroom.
2. Identify any necessary adaptations to the ATTA to reduce the time demands placed on respondents, improving feasibility for classroom use, whilst maintaining its reliability as a measure of computational thinking in undergraduate computing students.

2. Background

Computational Thinking (CT) as a concept dates back to the work of Papert in 1980 (Papert, 2020), with a surge in interest for educational researchers following Janet Wings' seminal article in 2006 (Wing,

2006). Most recently, CT has been brought further into focus by the rapid emergence of Generative AI (GenAI) tools. According to Wing, "Computational Thinking involves solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science" (Wing, 2006). However, there is still a lack of consensus on a formal definition of CT (Gouws, Bradshaw, & Wentworth, 2013; Grover & Pea, 2013). Similarly, there is some variation when considering the components of CT, but when reviewing the literature, Algorithmic Thinking, Decomposition, Abstraction, Pattern Recognition and Evaluation are commonly featured as core components of CT (Lafuente Martínez et al., 2022).

2.1. Computational Thinking Assessment

As the area of CT skills development has grown in popularity, a number of reviews have attempted to synthesise the existing literature on CT assessment tools. A review by de Araujo et al. in 2016 (Araujo, Santos, Andrade, Guerrero, & Dagienė, 2017) highlighted inconsistencies in CT assessment owing to the lack of a generally accepted CT definition and noted difficulties with replication of research as a result of insufficient detail in study details. Another review by Haseski and Ilic (Haseski & Iliç, 2019) identified a lack of emphasis on affective variables such as self-efficacy and called for more research on CT assessment tools for Higher Education. Other reviews called out a lack of evidence for the validity and reliability of CT assessment tools, and the difficulties in applying K-12 based CT frameworks to Higher Education (Tang et al., 2020; Cruz Castro et al., 2021).

2.2. Tool Type and Focus

Tools for measuring CT Skills take several forms, ranging from tests (Anistyasari, Ekohariadi, Budit-jahjanto, & Hidayati, 2021; Salac, Thomas, Butler, & Franklin, 2021; Zhang, Wong, & Pan, 2021; Brackmann et al., 2017; Guggemos, Seufert, & Román-González, 2019; Román-González, Pérez-González, & Jiménez-Fernández, 2017), self-reporting surveys (Sondakh, Osman, & Zainudin, 2019) and rubrics (Von Wangenheim et al., 2018; Clarke-Midura, Silvis, Shumway, Lee, & Kozlowski, 2021; Leela, Chookeaw, & Nilsook, 2019; Metcalf et al., 2021; Werner, Denner, Campe, & Kawamoto, 2012) among others. Of these, tests are the most popular form at all levels of education largely due to their ease of automation and scalability for larger studies making them easily replicable in new contexts. Some tools focus on specific programming constructs (Basu, Rutstein, Xu, & Shear, 2020; Chang, Sun, Wu, & Guizani, 2018; Parker et al., 2022; Zapata-Cáceres, Martín-Barroso, & Román-González, 2020; DEMİR-KAYMAK, DUMAN, RANDLER, & HORZUM, 2022; Diago, González-Calero, & Yáñez, 2022; Fanchamps, Slangen, Specht, & Hennissen, 2021; Hu, Chen, Yuadi, & Chen, 2022), while others centre around CT specific concepts (Bati, Yetişir, Çalişkan, Güneş, & Gül Saçan, 2018; Lee & Cho, 2021; Lafuente Martínez et al., 2022; Kalliopi & Michail, 2019; Koh, Nickerson, Basawapatna, & Repenning, 2014; El-Hamamsy et al., 2022) such as algorithmic thinking, problem decomposition and abstraction. While programming constructs are the most common focus of tools used at primary and secondary level, in Higher Education the focus shifts to CT specific concepts.

2.3. Tool Reliability

The evidence in support of reliability and validity of CT assessment tools varies greatly. Some publications provide statistical data regarding the reliability of the tool used such as internal consistency using Cronbachs α (Anistyasari et al., 2021; Çetin, Otu, & Oktaç, 2020; Çoban & Korkmaz, 2021; Fanchamps et al., 2021; Finke et al., 2022; Zhang et al., 2021), or coefficient α (Miller et al., 2013; Peteranetz, Morrow, & Soh, 2020), or Cohens κ for inter-rater reliability (Boom, Bower, Siemon, & Arguel, 2022; Kalliopi & Michail, 2019; Metcalf et al., 2021; Rowe et al., 2021; Yin, Hadad, Tang, & Lin, 2020) for construct development. Others provided evidence of the validity of the tool using methods such as comparison with another tool or measure (Araujo et al., 2017; BUBICA & BOLJAT, 2022; Masiulionytė-Dagienė & Jevsikova, 2022; Çoban & Korkmaz, 2021), comparison with academic achievement (DEMİR-KAYMAK et al., 2022; Mukasheva & Omirzakova, 2021; Román-González, Pérez-González, Moreno-León, & Robles, 2018), reporting on pilot studies (Ambrósio, Xavier, & Georges, 2014; Clarke-Midura et al., 2021; Lee & Cho, 2021; Leifheit et al., 2019; Palts & Pedaste, 2019; Parker

et al., 2022), or expert reviews (Lafuente Martínez et al., 2022; Lai, 2021; Li, Xu, & Liu, 2021; Sondakh et al., 2019; Tsai, Chien, Wen-Yu Lee, Hsu, & Liang, 2022; Zapata-Cáceres et al., 2020).

2.4. Summary

There remains a significant limitation in CT assessment research: many tools lack public accessibility and detailed documentation, restricting research replicability. Taking account of this, along with tool type, focus, and evidence of reliability and validity, three potential candidates for use in this research emerged: the Computational Thinking Test (CTt) (Román-González et al., 2017), the Algorithmic Thinking Test for Adults (ATTA) (Lafuente Martínez et al., 2022), and the Holistic Assessment of Computational Thinking (HiACT) (Sondakh et al., 2019). CTt is widely validated but primarily designed for K-12, while HiACT, aimed at higher education, emphasizes self-reported approaches rather than performance. ATTA, explicitly developed for adults and validated through psychometric analysis with 289 incentivised participants, assesses both knowledge and skills. Given these attributes, ATTA was identified as a potentially suitable tool for use in this study.

3. Method

The goal of this research is to pilot the ATTA within an undergraduate Higher Education Computing context in order to evaluate its suitability, reliability, and relevance for application within this context. In doing so, this research aims to identify any necessary adaptations to the ATTA with a view to reducing the time demands placed on students while maintaining its effectiveness as a measure of CT skills. Pre-pilot and pilot studies were conducted using the ATTA in an undergraduate Higher Education Computing context in 2023/2024 and 2024/2025 academic years (Sheridan, Stynes, Pathak, & Quille, 2025). The pre-pilot received a response rate of zero and feedback was used to inform the design of the pilot study the following academic year.

3.1. Data Collection & Analysis

During the pilot study, responses were collected using Microsoft Forms and the collected responses were then logged using microsoft excel. For the purpose of reporting, the original numbering of items in the ATTA was retained. This numbering system retained the original numbers used during the development of the ATTA. As questions appear in the survey in ascending order of difficulty, the numbers do not appear in sequence. The 20 items in the survey are numbered 27, 25, 26, 20, 13, 23, 7, 16, 6, 8, 21, 9, 17, 2, 11, 12, 24, 1, 19 and 3. The full set of questions and answers for the ATTA can be found in Appendix A. Those who opted not to participate in the research study were removed from the data set. Participants had also consented to making their module results available for comparison. The results of their first programming assignment were added to the data set at this point. Each record was anonymised, assigning a unique id number to each participant. The responses to each question of the ATTA were reviewed and correct answers were marked as 1 and incorrect as 0. Items which had been skipped were marked as 0 and also coded as non-response at this point. The total score for each participant was calculated with each question assigned the same value. Descriptive Statistics were calculated for the full data set and Pearsons correlation coefficient was calculated to examine the relationship between the ATTA and Programming Project scores. To confirm the internal consistency of the ATTA, Cronbachs α was calculated.

3.2. Item Non-Responses

Items with high non-response rates were examined. While high non-response can indicate problematic items, decisions regarding item removal were not based on missingness alone. The ATTA is designed such that the questions appear in ascending order of difficulty. Although the authors do not explicityly provide a psychometric justification for this, their item development process emphasised a need for test accessibility for both novices and experts and deliberately sought to include a range of difficulties across items. As the intention of this analysis is to reduce the scale for use in a time constrained classroom setting, the level of missingness was carefully considered as it might indicate an item either being unreachable or too difficult for the target participants. Items toward the end of the survey with low or zero variance coupled with high levels of non-response were removed at this point in order to

facilitate further analysis.

3.3. Item Reduction

Kaiser-Mayer-Olkin (KMO) and Bartletts Test of Sphericity (BTS) were then calculated to evaluate the suitability of the data for factor analysis. Initial results indicated sufficient correlation between items for factor analysis as evidenced by a significant BTS. However, the overall KMO value was unacceptable. This can sometimes be an indication that the ratio of responses to variables is unsuitable for factor analysis. The ATTA contains 20 items and a total of 37 responses were received. The literature on the ratio of sample size to number of variables varies, from 5:1 to 30:1 (Cohen, Manion, & Morrison, 2002).

An iterative item reduction process was then undertaken with the objective of increasing the KMO to a level acceptable for factor analysis. During each iteration, the performance of the item was evaluated using multiple criteria, including non- response patterns, sampling adequacy, inter-item correlations, examining the KMO, BTS, and Cronbachs α before and after the removal of each item to evaluate the impact of item removal. Exploratory factor analysis was conducted using Principal Component Analysis using Direct Oblimin rotation with Kaiser normalisation.

Firstly, items toward the end of the scale that had little to no variance and high non-response rates were considered unreached items and thus eliminated. KMO and BTS statistics were then re-examined alongside the anti-image matrix and individual Measures of Sampling Adequacy (MSA). An item with MSA below .5 is deemed unsuitable for factor analysis as it has very weak relationships with other variables and is poorly predicted by other items (Kaiser, 1974). Subsequently, items with MSA below .5 were eliminated in turn starting with the item with the lowest MSA. KMO and BTS were re-examined after each elimination. Next, inter-item correlations were examined iteratively, removing items that showed weak correlations with other items or predominantly negative correlations (DeVellis & Thorpe, 2021). Items with most correlations below .3 or with mostly negative correlations were removed in turn, re-examining KMO and BTS after each removal.

3.4. Factor Analysis

Once the KMO and BTS were at acceptable levels, exploratory factor analysis was conducted using Principal Component Analysis with Direct Oblimin rotation and Kaiser normalisation. Examination of Eigenvalues and Scree Plot indicated a 2-factor solution. Items that did not support this model or that mapped uniquely to a third factor were examined further and removed if appropriate.

4. Results

Through an iterative item reduction process, 11 of the 20 items were removed from the scale. Table 1 identifies each of the removed items, the reason for its removal, and the Cronbach's α and KMO values after its removal.

4.1. Data Collection and Analysis

There were 44 participants in total. Of those, 7 subsequently withdrew from their programme of study making assessment results unavailable. Thus, the remaining 37 responses were considered in the analysis alongside assessment results. Overall scores were calculated for each participant. The mean score was 30.54 as a percentage of correct answers and standard deviation was 16.95. Pearsons correlation Coefficient of the overall ATTA score and assessment result was .59.

4.2. Item Non-Response

The final 2 items in the scale, items 19 and 3 had high non-response rates of 59% and 51% respectively. These Items were considered unreachable and removed from the scale. Once these items were removed, Cronbachs α on the remaining 18 items was calculated as .722. KMO on this 18-item scale was .465 which is unacceptable for factor analysis.

4.3. Item Reduction

MSA for each of the individual items was calculated using an anti-image correlation. Item 17 was excluded at this point as it had a low MSA value of .196 and a high non-response rate of 32%. After the removal of item 17, KMO increased to .556 which is considered poor but not unacceptable. Examination of individual item MSA for the remaining items led to the iterative removal of item 2, item 26, and item 7 as each of these had MSA values below .5. This brought KMO to an acceptable value of .691 and retained only items with an individual MSA above .5. Next, the correlation matrix was examined. Items 1 and 13 which had few or no correlation with other items above .3 were both removed. The KMO and Cronbach's α for the remaining 12 item scale were calculated as .697 and point .727 respectively. Item 11, which had only negative correlations with the remaining items was removed next. After removal of item 11, KMO increased to an acceptable .706 and Cronbachs α increased to .782.

4.4. Factor Analysis

With KMO at .706, a level considered acceptable, exploratory factor analysis was conducted and the component matrix was generated using Principal Component Analysis where 4 factors were extracted with eigenvalues greater than 1. The scree plot was also generated. Inspection of the Scree plot showed a clear bend after the second component supporting the case for a 2-factor solution rather than the 4-factor solution indicated by the component matrix. Further examination of the component matrix showed that the third and fourth components were supported by only one item each: the third component by item 8; and the fourth component by item 21. When both items were removed, the final value for Cronbach's α was calculated at .764 and KMO at .71.

Table 1 – Sequential Item Removal Summary

Item Removed	Reason for Removal	α	KMO
19	Unreachable: Non response rate: 59%	.720	.465
3	Unreachable: Non response rate: 51%	.722	.465
17	MSA Below .5 (.196) & High Non-response rate (32%)	.724	.556
2	MSA below .5 (.243)	.733	.605
26	MSA below .5 (.401)	.735	.628
7	MSA below .5 (.486)	.713	.670
1	All correlations below .3	.715	.670
13	Only 1 correlation above .3	.727	.697
11	Only Negative Correlations	.782	.706
8	Factor Analysis: only item mapped to 3rd component	.766	.708
21	Factor Analysis: only item mapped to 4th component	.764	.710

The final solution presents two factors with 6 items loaded on the first component and 2 on the second. According to the component correlation matrix, correlation between the two factors is .283 which indicates that the factors are only weakly related with low to moderate correlation. This implies that while the two factors are not completely independent, they are also not measuring exactly the same thing. The original validation of the ATTA aimed to address several components of CT but found that algorithmic thinking was the single dimension underlying the test. Examination of the categories of CT targeted by each item during the original development of the ATTA shows that Algorithmic Thinking is either the first or second most endorsed category of all of the 6 items loaded on the first component. This aligns with the finding of the original validation that algorithmic thinking was the main underlying component. For the two items that load on the second component, Algorithmic Thinking was the most endorsed category for both items also. However, the second most endorsed category for those items was Evaluation. This overlap of categories may provide some explanation for the low correlation between the two factors.

4.5. Summary

When considering the full 20-item scale across 37 participant responses, Pearsons correlation Coefficient of the overall ATTA score and programming assessment result was .59. Calculating the score for each participant on the revised 9-item ATTA, Pearsons correlation coefficient of revised participant scores and programming assessment results was calculated as .56. Both Pearsons values of .59 and .56 indicate a moderate to strong correlation between the ATTA result and programming assessment result. The similarity between the Pearsons value for both the 20-item and 9-item scales implies that the removal of the 11 items has had little impact on the criterion validity of the scale for this sample.

The original validation of the 20-item ATTA scale calculated Cronbach's α of .854 (Lafuente Martínez et al., 2022, p. 1457) indicating very good reliability well above the threshold of .7 (Cronbach, 1951). The revised 9-item scale resulting from this study has a Cronbach's α value of .764, below the .854 of the full 20-item scale but still above the .7 threshold. This suggests that although the reduced scale demonstrates a slight decrease in internal consistency when compared to the original 20-item scale, it still maintains acceptable reliability.

Preliminary assessment of the execution time for the revised 9-item ATTA places execution time between 30 and 40 minutes. This is approximately half of the time required to complete the original 20-item ATTA. This, combined with improved internal consistency, and comparable correlation coefficients, indicates that the 9-item ATTA may offer a more efficient alternative with limited evidence of reduced criterion validity for this sample.

5. Conclusion & Future Work

This research evaluated the suitability, reliability, and feasibility of the ATTA CT assessment instrument for practical, time-constrained application in the context of a higher education computing classroom. While the ATTA had previously been robustly validated as a tool for measuring Computational Thinking skills, findings of the pre-pilot and pilot phases of this research confirm the execution time as a significant practical limitation. Low participation and student feedback in both the pre-pilot and pilot along with high levels of item non-response in the pilot indicate that the full 20-item scale is difficult to implement in a time-constrained setting.

A shortened 9-item ATTA was produced through an interative process of item reduction based on non-response patterns, sampling adequacy, inter-item correlations and factor analysis. This 9-item scale appears to retain acceptable internal consistency with Cronbach's α of .764 and satisfactory sampling adequacy indicated by KMO of .71. These results suggest that a shortened ATTA may provide a more feasible and practical tool for classroom use without substantially comprising reliability or validity.

However, there are several limitations which must be acknowledged. This study was conducted with a relatively small sample size (n=37) from a single institution which limits the generalisability of the findings. Further validation of the reduced scale is required with a larger sample size across multiple institutions before it can be recommended for wider adoption.

The next phase of this research is to administer the reduced 9-item ATTA across multiple institutions with a larger sample size to further evaluate its reliability, validity and genralisability. Further psychometric validation such as confirmatory factor analysis is required to strengthen the validity of the revised scale. Future studies will also explore how student ATTA performance is associated with other relevant constructs such as programming self-efficacy. Overall, this work contributes to the long-term goal of developing a practical, evidence-based CT Skills assessment tool for use in CT research in Higher Education.

6. References

Ambrósio, A. P., Xavier, C., & Georges, F. (2014). Digital ink for cognitive assessment of computational thinking. In *2014 ieee frontiers in education conference (fie) proceedings* (pp. 1–7). Spain: IEEE.

Anistyasari, Y., Ekohariadi, E., Buditjahjanto, I. P. A., & Hidayati, S. C. (2021). Exploring the psy-

chometric properties of computational thinking assessment in introductory programming. In *2021 international e-engineering education services conference (e-engineering)* (pp. 88–93). Jordan: IEEE.

Araujo, A. L. S. O., Santos, J. S., Andrade, W. L., Guerrero, D. D. S., & Dagienė, V. (2017). Exploring computational thinking assessment in introductory programming courses. In *2017 ieee frontiers in education conference (fie)* (pp. 1–9). Indianapolis, USA: IEEE.

Basu, S., Rutstein, D., Xu, Y., & Shear, L. (2020). A principled approach to designing a computational thinking practices assessment for early grades. In *Proceedings of the 51st acm technical symposium on computer science education* (pp. 912–918). Portland, OR, USA: ACM.

Bati, K., Yetişir, M. I., Çalişkan, I., Güneş, G., & Gül Saçan, E. (2018). Teaching the concept of time: A steam-based program on computational thinking in science education. *Cogent Education*, *5*(1), 1507306.

Boom, K.-D., Bower, M., Siemon, J., & Arguel, A. (2022). Relationships between computational thinking and the quality of computer programs. *Education and Information Technologies*, *27*(6), 8289-8310.

Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., & Barone, D. (2017). Development of computational thinking skills through unplugged activities in primary school. In *Proceedings of the 12th workshop on primary and secondary computing education* (p. 65-72). The Netherlands: ACM.

BUBICA, N., & BOLJAT, I. (2022). Assessment of computational thinking–a croatian evidence-centered design model. *Informatics in Education*, *21*(3), 425-463.

Celik, I. (2023). Exploring the determinants of artificial intelligence (ai) literacy: Digital divide, computational thinking, cognitive absorption. *Telematics and Informatics*, *83*, 102026.

Çetin, İ., Otu, T., & Oktaç, A. (2020). Adaption of the computational thinking test into turkish. *Turkish Journal of Computer and Mathematics Education*, *11*(2), 343-360.

Chang, Z., Sun, Y., Wu, T.-Y., & Guizani, M. (2018). Scratch analysis tool (sat): a modern scratch project analysis tool based on antlr to assess computational thinking skills. In *2018 14th international wireless communications & mobile computing conference (iwcmc)* (pp. 950–955). Cyprus: IEEE.

Clarke-Midura, J., Silvis, D., Shumway, J. F., Lee, V. R., & Kozlowski, J. S. (2021). Developing a kindergarten computational thinking assessment using evidence-centered design: the case of algorithmic thinking. *Computer Science Education*, *31*(2), 117–140.

Çoban, E., & Korkmaz, Ö. (2021). An alternative approach for measuring computational thinking: Performance-based platform. *Thinking Skills and Creativity*, *42*, 100929.

Cohen, L., Manion, L., & Morrison, K. (2002). *Research methods in education*. routledge.

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *psychometrika*, *16*(3), 297–334.

Cruz Castro, L., Shoaib, H., & Douglas, K. (2021). Computational thinking frameworks used in computational thinking assessment in higher education. a systematized literature review. In *Asee annual conference.*

DEMİR-KAYMAK, Z., DUMAN, İ., RANDLER, C., & HORZUM, M. B. (2022). The effect of gender, grade, time and chronotype on computational thinking: Longitudinal study. *Informatics in Education*, *21*(3), 465-478.

Denscombe, M. (2009). Item non-response rates: a comparison of online and paper questionnaires. *International journal of social research methodology*, *12*(4), 281–291.

DeVellis, R. F., & Thorpe, C. T. (2021). *Scale development: Theory and applications*. Sage publications.

Diago, P. D., González-Calero, J. A., & Yáñez, D. F. (2022). Exploring the development of mental rotation and computational skills in elementary students through educational robotics. *International Journal of Child-Computer Interaction*, *32*, 100388.

El-Hamamsy, L., Zapata-Cáceres, M., Barroso, E. M., Mondada, F., Zufferey, J. D., & Bruno, B. (2022). The competent computational thinking test: Development and validation of an unplugged com-

putational thinking test for upper primary school. *Journal of Educational Computing Research*, *60*(7), 07356331221081753.

Erez, Y., Mike, K., & Hazzan, O. (2024, aug 29). Leveraging computational thinking in the era of generative ai. *BLOG@CACM*, Communications of the ACM. Retrieved from `https://cacm.acm.org/blogcacm/leveraging-computational-thinking-in-the-era-of-generative-ai/` (Online blog post)

Fanchamps, N. L., Slangen, L., Specht, M., & Hennissen, P. (2021). The impact of sra-programming on computational thinking in a visual oriented programming environment. *Education and Information Technologies*, *26*(5), 6479–6498.

Federiakin, D., Molerov, D., Zlatkin-Troitschanskaia, O., & Maur, A. (2024). Prompt engineering as a new 21st century skill. In *Frontiers in education* (Vol. 9, p. 1366434).

Finke, S., Kemény, F., Sommer, M., Krnjic, V., Arendasy, M., Slany, W., & Landerl, K. (2022). Unravelling the numerical and spatial underpinnings of computational thinking: a pre-registered replication study. *Computer Science Education*, *32*(3), 1–22.

Gouws, L., Bradshaw, K., & Wentworth, P. (2013). First year student performance in a test for computational thinking. In *Proceedings of the south african institute for computer scientists and information technologists conference* (pp. 271–277). East London, South Africa: Association for Computing MachineryNew YorkNYUnited States.

Grover, S., & Pea, R. (2013). Computational thinking in k–12: A review of the state of the field. *Educational researcher*, *42*(1), 38–43.

Guggemos, J., Seufert, S., & Román-González, M. (2019). Measuring computational thinking–adapting a performance test and a self-assessment instrument for german-speaking countries. In *International association for development of the information society* (pp. 183–191). Italy: ERIC.

Haseski, H. I., & Iliç, U. (2019). An investigation of the data collection instruments developed to measure computational thinking. *Informatics in Education*, *18*(2), 297–319.

Hu, C.-C., Chen, M.-H., Yuadi, I., & Chen, N.-S. (2022). The effects of constructing robot-based storytelling system on college students' computational thinking skill and technology comprehension. In *2022 24th international conference on advanced communication technology (icact)* (pp. 496–500). Korea: IEEE.

Kaiser, H. F. (1974). An index of factorial simplicity. *psychometrika*, *39*(1), 31–36.

Kalliopi, K., & Michail, K. (2019). Assessing computational thinking skills at first stages of schooling. In *Proceedings of the 2019 3rd international conference on education and e-learning* (pp. 135–139). Spain: Association of Computing Machinery.

Khatamian Far, P. (2018). Challenges of recruitment and retention of university students as research participants: Lessons learned from a pilot study. *Journal of the Australian Library and Information Association*, *67*(3), 278–292.

Koh, K. H., Nickerson, H., Basawapatna, A., & Repenning, A. (2014). Early validation of computational thinking pattern analysis. In *Proceedings of the 2014 conference on innovation & technology in computer science education* (pp. 213–218).

Lafuente Martínez, M., Lévêque, O., Benítez, I., Hardebolle, C., & Zufferey, J. D. (2022). Assessing computational thinking: Development and validation of the algorithmic thinking test for adults. *Journal of Educational Computing Research*, 07356331211057819.

Lai, R. P. (2021). Beyond programming: A computer-based assessment of computational thinking competency. *ACM Transactions on Computing Education (TOCE)*, *22*(2), 1–27.

Lee, K., & Cho, J. (2021). Computational thinking evaluation tool development for early childhood software education. *JOIV: International Journal on Informatics Visualization*, *5*(3), 313–319.

Leela, S., Chookeaw, S., & Nilsook, P. (2019). An effective microlearning approach using living book to promote vocational students' computational thinking. In *Proceedings of the 2019 the 3rd international conference on digital technology in education* (pp. 25–29).

Leifheit, L., Tsarava, K., Moeller, K., Ostermann, K., Golle, J., Trautwein, U., & Ninaus, M. (2019). Development of a questionnaire on self-concept, motivational beliefs, and attitude towards pro-

gramming. In *Proceedings of the 14th workshop in primary and secondary computing education* (pp. 1–9).

Li, Y., Xu, S., & Liu, J. (2021). Development and validation of computational thinking assessment of chinese elementary school students. *Journal of Pacific Rim Psychology*, *15*, 18344909211010240.

Lockwood, J., & Mooney, A. (2017). Computational thinking in education: Where does it fit? a systematic literary review. *arXiv preprint arXiv:1703.07659*.

Masiulionytė-Dagienė, V., & Jevsikova, T. (2022). Assessing computational thinking: The relation of different assessment instruments and learning tools. In *International conference on informatics in schools: Situation, evolution, and perspectives* (pp. 66–77).

Metcalf, S. J., Reilly, J. M., Jeon, S., Wang, A., Pyers, A., Brennan, K., & Dede, C. (2021). Assessing computational thinking through the lenses of functionality and computational fluency. *Computer Science Education*, *31*(2), 199–223.

Miller, L. D., Soh, L.-K., Chiriacescu, V., Ingraham, E., Shell, D. F., Ramsay, S., & Hazley, M. P. (2013). Improving learning of computational thinking using creative thinking exercises in cs-1 computer science courses. In *2013 ieee frontiers in education conference (fie)* (pp. 1426–1432).

Mukasheva, M., & Omirzakova, A. (2021). Computational thinking assessment at primary school in the context of learning programming. *World Journal on Educational Technology: Current Issues*, *13*(3), 336–353.

of Ontario, H. E. Q. C., Cyr, D., Childs, R., & Elgie, S. (2013). *Recruiting students for research in postsecondary education: A guide*. Higher Education Quality Council of Ontario Toronto.

Palts, T., & Pedaste, M. (2019). Tasks for assessing computational thinking skills at secondary school level. In *International conference on innovative technologies and learning* (pp. 216–226).

Papert, S. A. (2020). *Mindstorms: Children, computers, and powerful ideas*. Basic books.

Parker, M. C., Garcia, L., Kao, Y. S., Franklin, D., Krause, S., & Warschauer, M. (2022). A pair of aces: An analysis of isomorphic questions on an elementary computing assessment. In *Proceedings of the 2022 acm conference on international computing education research-volume 1* (pp. 2–14).

Peteranetz, M. S., Morrow, P. M., & Soh, L.-K. (2020). Development and validation of the computational thinking concepts and skills test. In *Proceedings of the 51st acm technical symposium on computer science education* (pp. 926–932).

Pintrich, P. R., & De Groot, E. V. (1988). Motivated strategies for learning questionnaire. *Journal of Educational Psychology*.

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? criterion validity of the computational thinking test. *Computers in human behavior*, *72*, 678–691.

Román-González, M., Pérez-González, J.-C., Moreno-León, J., & Robles, G. (2018). Can computational talent be detected? predictive validity of the computational thinking test. *International Journal of Child-Computer Interaction*, *18*, 47–58.

Romero, M. (2024). Lifelong learning challenges in the era of artificial intelligence: A computational thinking perspective. *arXiv preprint arXiv:2405.19837*.

Rowe, E., Almeda, M. V., Asbell-Clarke, J., Scruggs, R., Baker, R., Bardar, E., & Gasca, S. (2021). Assessing implicit computational thinking in zoombinis puzzle gameplay. *Computers in Human Behavior*, *120*, 106707.

Salac, J., Thomas, C., Butler, C., & Franklin, D. (2021). Investigating the role of cognitive abilities in computational thinking for young learners. In *Proceedings of the 17th acm conference on international computing education research* (pp. 2–17).

Sheridan, F., Stynes, P., Pathak, P., & Quille, K. (2024). Computational thinking assessment tools: A systematic review of the literature (initial findings). In *Proceedings of the 2024 conference on united kingdom & ireland computing education research* (pp. 1–1).

Sheridan, F., Stynes, P., Pathak, P., & Quille, K. (2025). Work in progress: Computational thinking skills assessment in higher education. In *2025 ieee engineering education world conference (edunine)* (pp. 1–4).

Sondakh, D. E., Osman, K., & Zainudin, S. (2019). Holistic assessment of computational thinking for undergraduate: Reliability and convergent validity. In *Proceedings of the 2019 11th international conference on education technology and computers* (pp. 241–245).

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, *148*, 103798.

Tsai, M.-J., Chien, F. P., Wen-Yu Lee, S., Hsu, C.-Y., & Liang, J.-C. (2022). Development and validation of the computational thinking test for elementary school students (ctt-es): Correlate ct competency with ct disposition. *Journal of Educational Computing Research*, 07356331211051043.

Von Wangenheim, C. G., Hauck, J. C., Demetrio, M. F., Pelle, R., da Cruz Alves, N., Barbosa, H., & Azevedo, L. F. (2018). Codemaster–automatic assessment and grading of app inventor and snap! programs. *Informatics in Education*, *17*(1), 117–150.

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The fairy performance assessment: Measuring computational thinking in middle school. In *Proceedings of the 43rd acm technical symposium on computer science education* (pp. 215–220).

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Yin, Y., Hadad, R., Tang, X., & Lin, Q. (2020). Improving and assessing computational thinking in maker activities: The integration with physics and engineering learning. *Journal of Science Education and Technology*, *29*(2), 189–214.

Zapata-Cáceres, M., Martín-Barroso, E., & Román-González, M. (2020). Computational thinking test for beginners: Design and content validation. In *2020 ieee global engineering education conference (educon)* (pp. 1905–1914).

Zhang, S., Wong, G. K., & Pan, G. (2021). Computational thinking test for lower primary students: Design principles, content validation, and pilot testing. In *2021 ieee international conference on engineering, technology & education (tale)* (pp. 345–352).

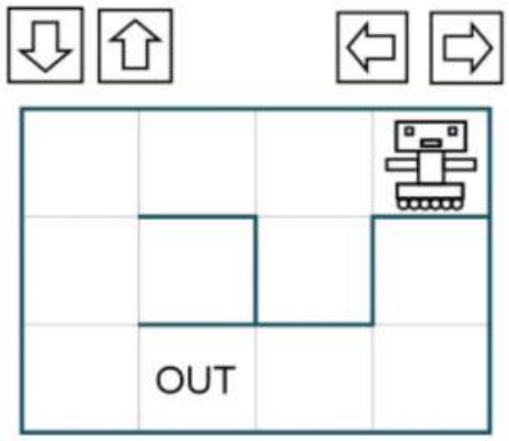
Appendix A: Survey Instrument

ATTA – Algorithmic Thinking Test for Adults

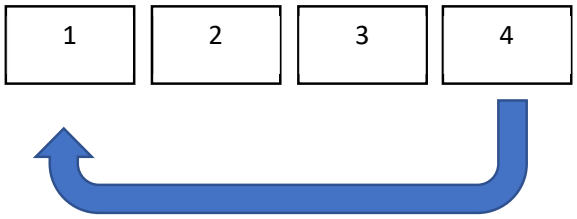
INSTRUCTIONS: Below you will find some questions that attempt to evaluate your algorithmic thinking skills. Each question is a different problem that you have to solve. Simply answer each question as best as you can. Some questions require you to select a choice, while others ask you to write a short text. Some questions may look easy and others may look difficult to you, that's fine. If you think a certain question is very difficult, don't worry, you can make a guess and try the next one. You can use a sheet of paper and a pen if you want. You don't need to code or use any kind of software. Thanks for your participation!

#27

This robot is programmed through a series of arrow commands (“*left”, “right”, “up”, “down”*). Each arrow command causes the robot to either move one step (if the move in that direction is possible) or not to move (if the move in that direction is not possible).



Imagine the robot has a series of four commands, which it will repeat over and over again.



You can put any arrow that you want (“*left”, “right”, “up”, “down”*) in each command, and you may repeat the same arrow in the series of commands if you want. What should be those 4 commands in order for the robot to reach the square called “OUT” and get out of the maze?

1_____ 2_____ 3_____ 4_____

SOLUTIONS:

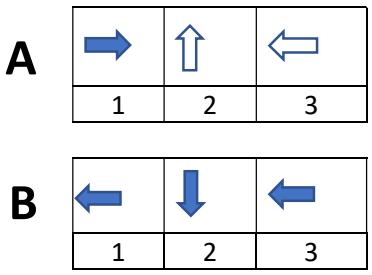
Left, left, down, right

/ right, left, left, down

/ down, right, left, left

/ right, left, left, down

#25



You have this initial set of three arrows labelled A. And, by applying some functions, you want to get this final set of arrows labelled B.

The first time you apply a function, this is applied to the three arrows; but, the following functions are applied to n-1 arrows. That is, the second function is applied only to the first two arrows; and the third function is applied only to the first arrow.

See the functions that one can apply to obtain the final set of arrows:

Function 1: turn the arrows 45 degrees anti-clockwise

Function 2: turn the arrows 180 degrees

Function 3: switch colors (turn white arrows into blue arrows, and blue arrows into white arrows)

What is the sequence of functions that applied to the initial set of arrows will get you the final set of arrows?

A) function 3; function 3; function 2

B) function 2; function 3; function 3

C) function 3; function 2; function 1

D) function 3; function 2; function 3

SOLUTION: D

#26

You have the following DNA sequence:



And you want to get the following sequence:



You can implement the following operations:

Swap (): it will swap one character for another. For example, *swap (A,G)* will turn the sequence AAGT into GGAT

Insert (): it will insert a specified character at the beginning of the sequence. For example, *insert (A*) will turn the sequence GT into AGT

Delete (): it will delete all specified characters. For example, *delete (A)* will turn the sequence AAGT into GT

What operations will turn the initial DNA sequence into the one we want?

A) Delete (G); then Insert (T); then Swap (C,T)

B) Swap (C,T); then Insert (T); then Delete (G)

C) Insert (T); then Swap (C,T); then Delete (G)

D) Insert (T); then Delete (G); then Swap (C,T)

SOLUTION:

B

#20

After finishing your long paper, you just discovered a mistake you made. All 4's should be 7's and all 7's should be 4's. To amend this, you can use an editor that will replace any sequence of characters with another sequence.

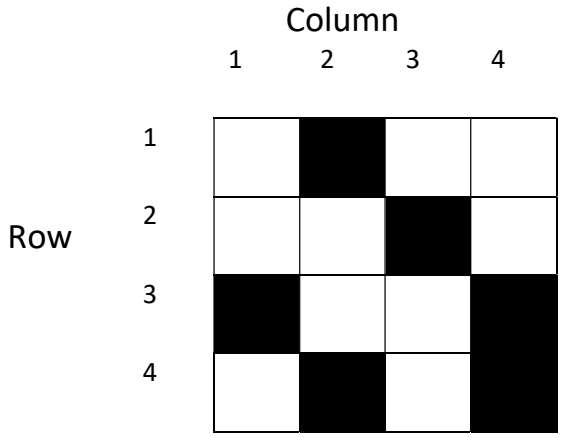
In order to describe the replacement you would make with the editor, please fill in the blanks:

Replace all _____ with _____; then replace all _____with _____; then replace all _____ with _____

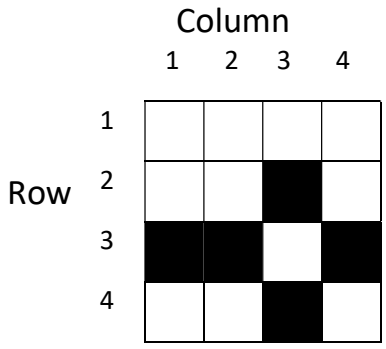
SOLUTION:

Replace all __4's___ with _XXs (any variable)__; then replace all _7's___ with _4's__; then replace all __XXs(variable mentioned before)__ with _7's___ (4's and 7's positions can be reversed here)

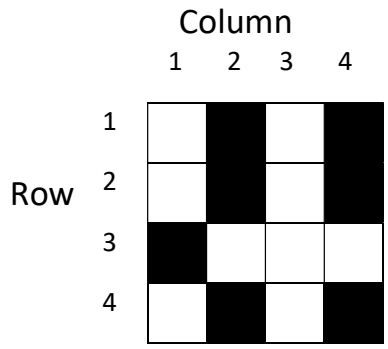
#13 A collection of elements is shown below with 2 dimensions: rows and columns. A black square indicates a 1, and a white square indicates 0. Imagine that we run two operations on the collection of elements. First, we replace column 4 by column 2; and then we replace row 2 by row 4. What will be the result?



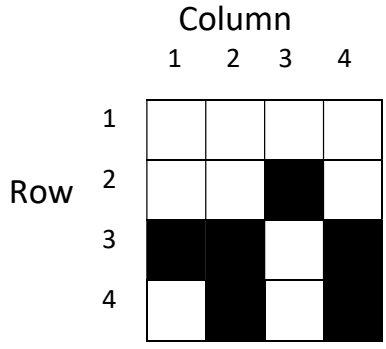
A.



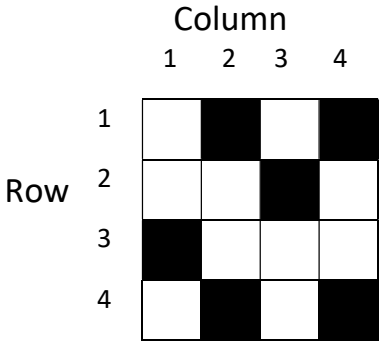
B.



C.



D.



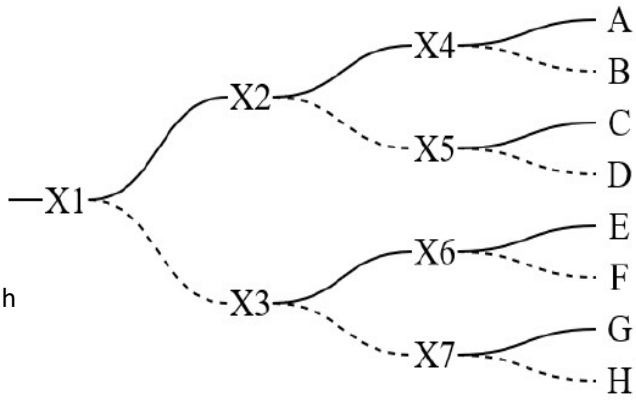
SOLUTION: B

#23

Eight trains (named a, b, c, d, e, f, g, h) enter the circuit from left to right. In this circuit there are 7 switches (X1 to X7) with two positions: up or down. Each switch is initially set to direct trains up. After a train passes a switch, the switch changes from up to down, or from down to up. Train A needs to go to station A, train b to station B, train c to station C, and so on.



Trains a, b, c, d, e, f, g, h



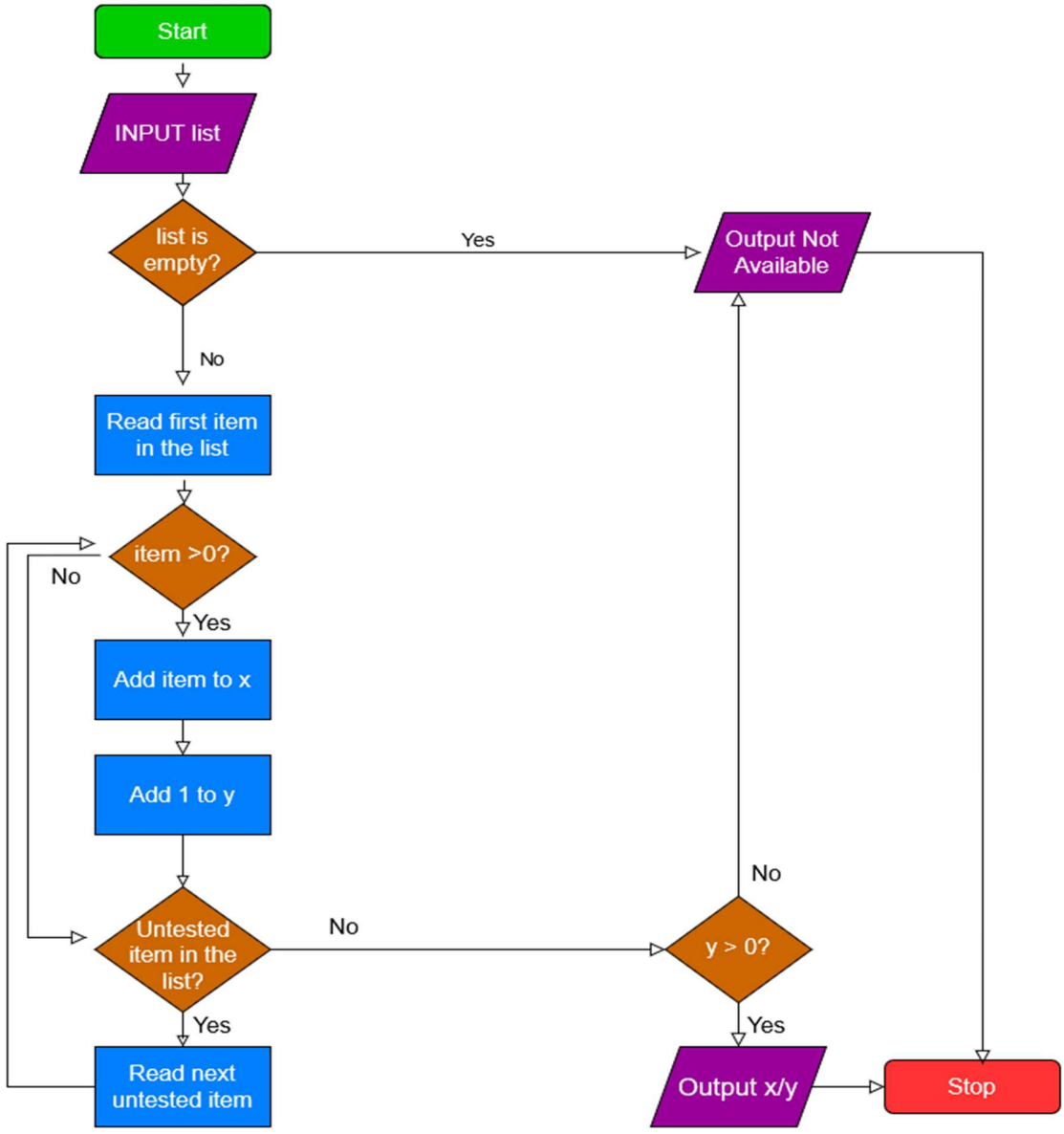
The railroad director needs to ensure that all the trains go to their correct stations, so what is the correct order for the trains to pass through switch X1?

___, ___, ___, ___, ___, ___, ___, ___

SOLUTION: A, E, C, G, B, F, D, H

#7

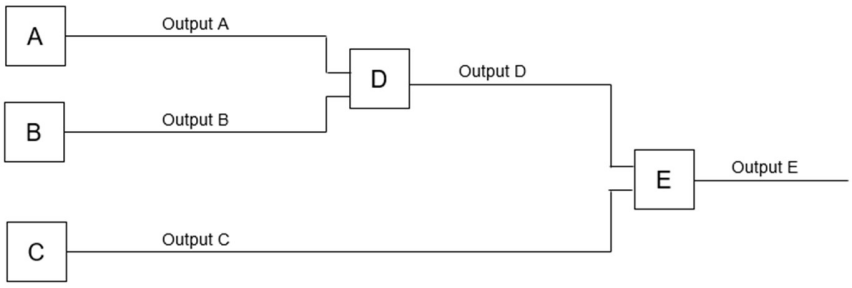
Olivia is studying the climate in her village and, over the last year, she has measured the temperature at the same time every day. She has created a list with all the temperatures she has collected so far. She prepares the following algorithm to analyze her list. What is the output she will get?



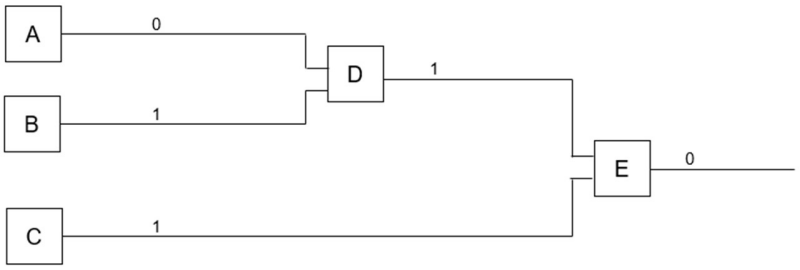
A. The percentage of positive temperatures in the list
B. The average of the positive temperatures in the list
C. The number of positive temperatures in the list
D. The sum of the positive temperatures in the list

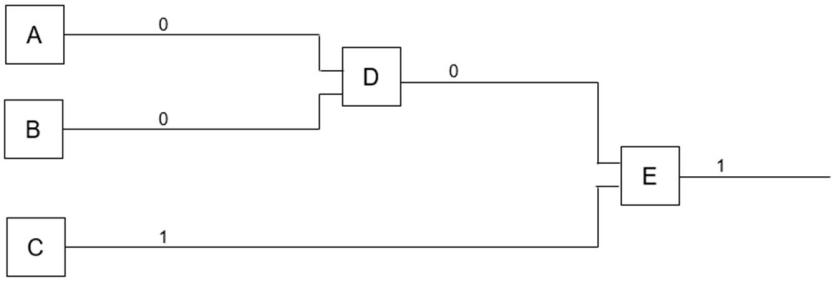
SOLUTION: B

#16 The following scheme represents a circuit of nodes. A, B, C, D and E are nodes of this circuit. Nodes fire and pass their signal on to the next node on their right (e.g., node A gives its output to node D, which gives its output to node E). Nodes' output can be either 1 or 0.



We see the circuit produces the following states:





We know that Node D and E ALWAYS follow the same algorithm. See the possible algorithms:

Algorithm 1	If inputs are (1,1)	Outputs 1	Else: outputs 0
Algorithm 2	If inputs are (1,0) or (0,1)	Outputs 1	Else: outputs 0
Algorithm 3	If inputs are (1,1)	Outputs 0	Else: outputs 1

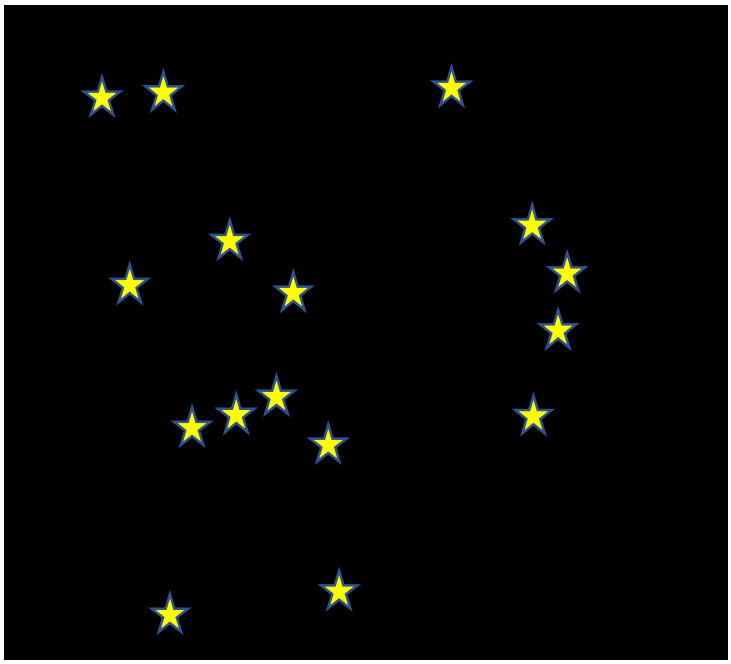
Which algorithm node D and E are both following?

A) Algorithm 1
B) Algorithm 2
C) Algorithm 3
D) This problem has no solution

SOLUTION: B

6

You are carrying out a series of astronomical observations on the Orion constellation. You want to observe, using the same telescope, all the stars included in this constellation:



Setting up a new position for the telescope takes time which you want to optimize; so, you decide that you will observe all the stars following the shortest path that connects all of them. To do that, imagine someone proposes the following algorithm:

1. Start from a random star

2. If there's still a star to observe, go to the nearest non-observed star

What is fundamentally wrong with this algorithm?

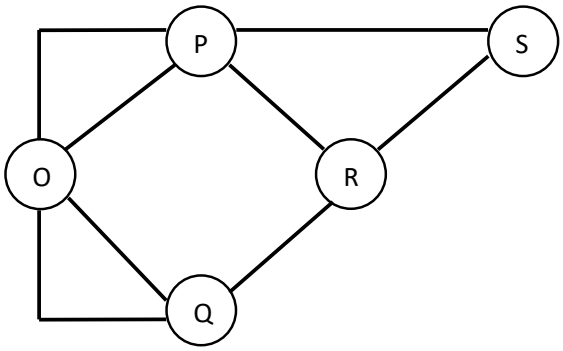
A. The algorithm may eventually choose longer paths
B. The starting star should not be randomly chosen, but should be a one on the center of the map
C. The algorithm will leave non-observed stars in the path
D. There's nothing wrong with this algorithm, it will always choose the shortest path

SOLUTION: A

8

This is a path composed of 5 vertices: O, P, Q, R, S . Every vertex is connected to other vertices through one or more lines.

You want to connect all the vertices of this path with your pencil in such a way that you pass only once through ALL THE LINES of this sketch and without lifting your pencil. You can pass more than once through every vertex, though. Where would you start the path?



A. Always on a vertex with the lowest number of lines

B. Always on a vertex with an even number of lines

C. Always on a vertex with an odd number of lines

D. One cannot connect all the vertices in figures like this

SOLUTION: C

21

We have two methods to represent a group of people. In both methods, for each person a circle is drawn. In method 1, we align people horizontally if they are connected on WhatsApp, and we align them vertically if they are connected on Instagram. In figure 1, two people are connected on WhatsApp (aligned horizontally); and three are connected on Instagram (aligned vertically).

In method 2, we draw a line between two people if they are connected either on WhatsApp or Instagram. According to method 2, the same group as above would be represented as shown in Figure 2.

A new group of six people is shown in Figure 3 according to method 1.

Which of the four diagrams **A, B, C** or **D** would represent this group of people according to method 2?

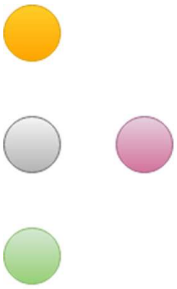


Figure 1

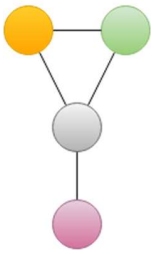


Figure 2

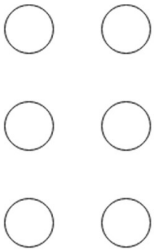
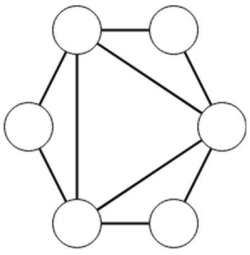
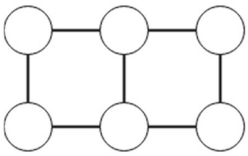


Figure 3

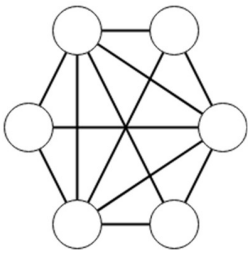
A



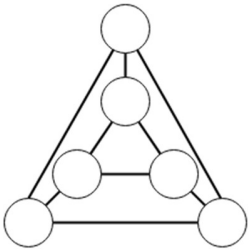
B



C



D



SOLUTION: D

9

We have this game:

There are four players (P, Q, R, S): one of them is pointing at him/herself, and three of them are pointing at any other player.

Player P gets the ball.

While the ball-holder isn't pointing at him/herself:

The ball-holder passes the ball to the player that s/he is pointing at.

If the ball-holder is pointing at him/herself, s/he drops the ball and the game finishes.

We can state that:

A. This game will go on forever

B. This game will eventually stop because someone will drop the ball

C. This game will stop when player P gets the ball again

D. We don't know whether this game will ever stop

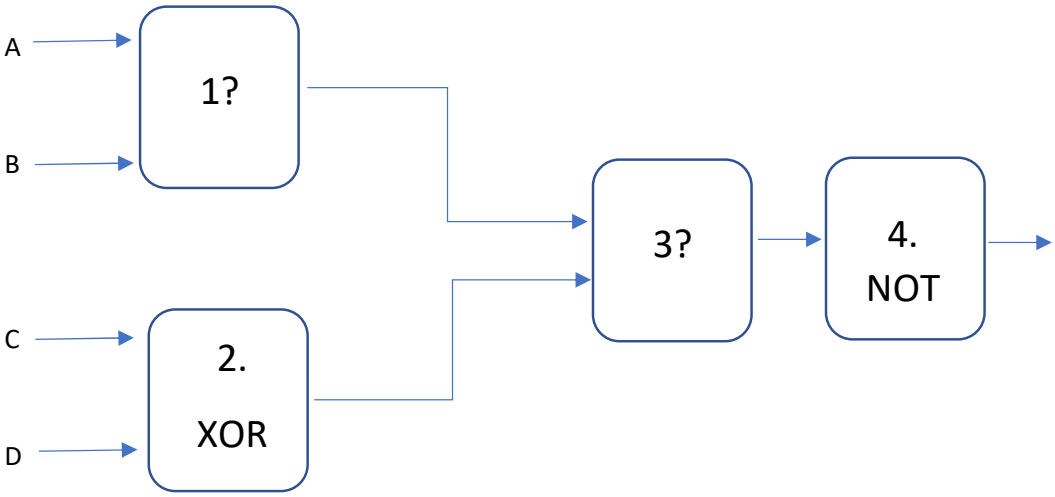
SOLUTION: D

#17

An “OR” gate outputs 1 if it receives one or more inputs with value 1. Else, it outputs 0.

A “XOR” gate outputs 1 if it has two inputs with different values (0, 1) or (1, 0). Else, it outputs 0.

A “NOT” gate inverts its input (it outputs 1 if the input is 0, and vice versa).



The inputs A, B, C and D can either be 0 or 1. We want this circuit to check that A=B and that C=D. That is, the final output of this circuit will be 1 if A=B and if C=D; otherwise, the output will be 0. We already know that gate 2 is a XOR and gate 4 is a NOT. What gates should we put in positions 1 and 3 of this circuit?

1____

3____

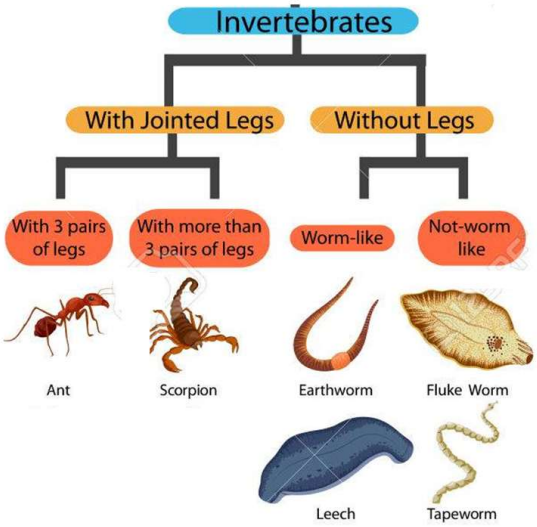
SOLUTION:

1: XOR

3: OR

2

Imagine that someone picks one of these animals and you have to guess it posing only binary questions. Binary questions are those whose answer can only be yes or no, like “does it have jointed legs?” Assuming that you follow an optimal strategy that reduces the number of questions, what is the average number of questions necessary to guess an animal uniformly picked at random from this chart?



A. 2.33

B. 2.67

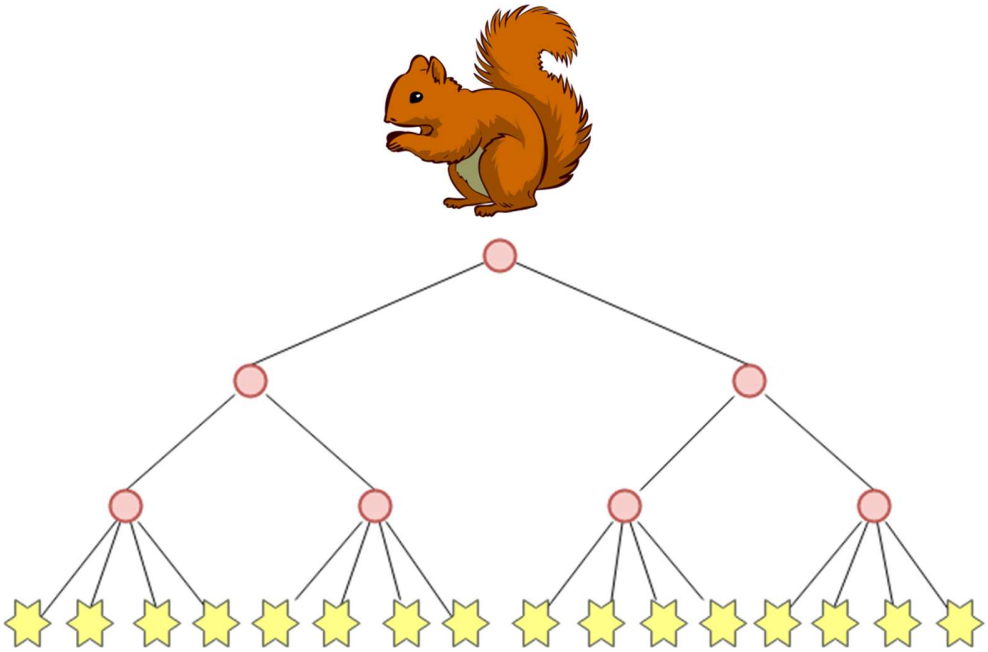
C. 3

D. 6

SOLUTION: B

11

Let’s examine the minimum distance the squirrel must travel to pick up all the stars. How many branches does it need to go through until it picks up the last star?



A. Number of stars x levels of the tree

B. 2 x number of branches – number of levels of the tree

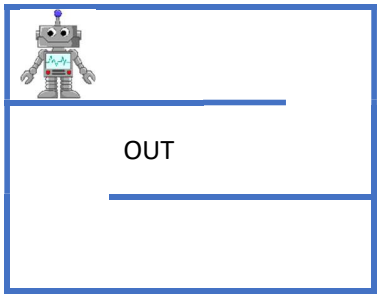
C. (Number of levels of the tree)^2

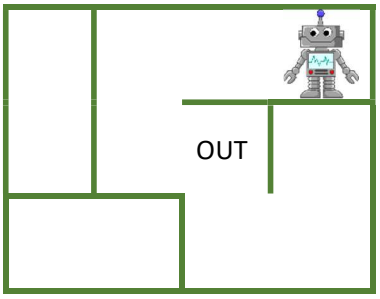
D. Number of branches + number of levels of the tree

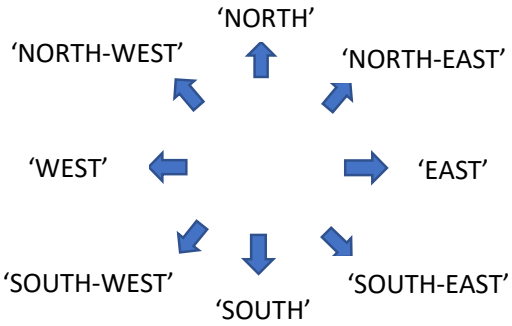
SOLUTION: B

12

Describe one single algorithm that will get both robots out of their mazes in exactly 3 instructions. Both robots execute first instruction 1, then instruction 2, and then instruction 3.







Fill in the blanks with one the following eight words, where these coordinates are

set according to how you see the maze:

1. If the 'out' square is to the SOUTH-EAST of the robot, keep advancing squares to the EAST until the robot collides with a wall (it cannot go in that direction any further)

    Else go to the _____ until the robot collides with a wall

2. When the robot collides

    If the 'out' square is to the SOUTH-WEST of the robot, keep advancing to the _____ until the robot collides with a wall

        Else go to the SOUTH until the robot collides with a wall

3. When the robot collides

    If the 'out' square is to the _____ of the robot, keep advancing to the WEST until the robot finds the out square

        Else go to the _____ until the robot finds the out square

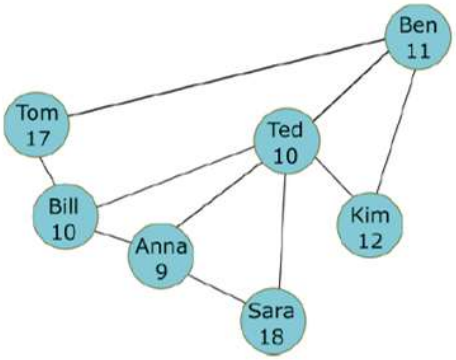
SOLUTION: WEST, SOUTH, WEST, EAST

24

This diagram shows the relationship between 7 students in a book-sharing club. Their names and ages are shown in the image. Students who are friends are connected through a line.

The club has some regulations for members:

When you receive a book, you read it (if you haven't already done so) and then pass it to the youngest friend who has not read it yet. If, however, all your friends have read the book then you should pass it to the friend who first gave it to you.



Ben has read a new book and wants to share it with his friends. Who will read the book last?

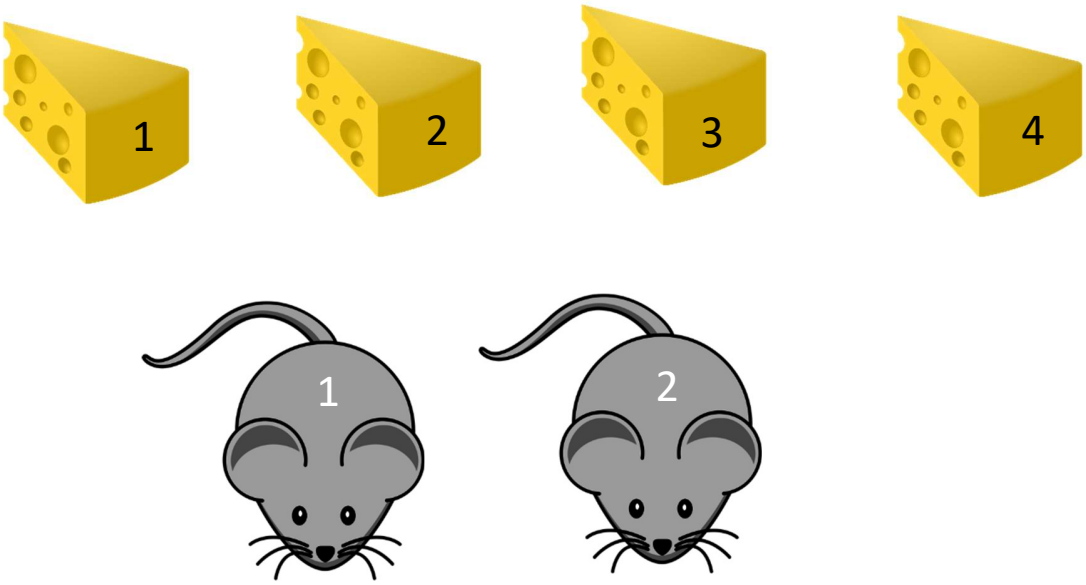
_____

SOLUTION:

KIM

1

You have four different cheeses and one of them has a tasteless and odorless bacterium. This cheese causes symptoms of food-poisoning 24 hours after ingesting it. You want to find out which cheese it is, so you decide to use mice to taste them. But, you only have two mice left in your lab and your deadline to find the poisoned cheese is exactly 24 hours from now! You can assign every mouse the number of cheeses that you want. How would you do it?



I would assign mouse 1 to cheese/s _____

I would assign mouse 2 to cheese/s _____

SOLUTION: The two mice should share one cheese (e.g., cheese 2), each mouse should eat one different cheese (e.g., mouse 1 eats cheese 1 and mouse 2 eats cheese 3), and leave aside one cheese (e.g., cheese 4 in that example).

19

You have two ropes and a lighter, and each rope takes exactly one hour to burn. They don't burn randomly, but at a regular speed. How would you use them to time exactly 15 minutes? The ropes cannot be cut, bent, straightened or measured with external instruments. (You don't need to time 15 minutes from the moment that you start your actions)



Please, briefly explain the procedure:

_____
_____
_____
_____

SOLUTION:

1. Light rope 1 at both ends and rope 2 at one end.

2. When the two flames on rope 1 meet, 30 minutes will have passed. Rope 2 has 30 minutes left of burn-time.

3. At that point, light Rope 2 at the other end.

4. In exactly fifteen minutes, Rope 2 will be completely burnt

To score 1, it's important that the subject refers to procedures 1 and 3.

If the subject mentions other procedures to measure 15' that involve manipulating the ropes (put both ropes in a cross-shape, bend them, mark them with external tools, etc.) this is not correct and it should be scored 0.

3

You have been given 9 coins of the same value, but one of them is fake which you could tell because it is lighter than the rest. You have a balance like the one in the picture to weigh the coins, and each weighing can result in “the balance leans to the right”, “the balance leans to the left”, or “the balance rests stable”. Assuming you are following an optimal strategy to reduce the number of weighings, how many weighings are necessary to identify the fake coin?





SPACE LEFT BLANK for answer

SOLUTION: 2

Workshop: Vibe Coding in Pairs

Gabriel Hughes
UCL Interaction Centre
University College London
gabriel.hughes@ucl.ac.uk

1. Abstract

Proposal: A 1-hour, 2-part interactive workshop encouraging reflection and ideation on the realities of coding with LLMs. In the first part, participants complete a programming task in pairs with an agentic coding assistant (ACA) such as Claude Code or Cline, or a tool of their choosing. Programming in pairs offers two benefits: (1) it accommodates for varying levels of coding skill, as less experienced participants can collaborate with more experienced ones, and (2) it creates a more interactive environment and more opportunities for discussion as participants can voice observations they made about their partners' experiences. During part 2, participants discuss their experiences with joint goals of highlighting the challenges faced during the activity, as well as brainstorming blue sky solutions to those challenges that capture what the group as a whole envisions as the future of programming.

2. Format

2.1. Part 1

The coding activity is meant to be fun, particularly for completing in pairs, outside the realm of typical SWE work, and easy enough to be completed within 30 minutes. Participants will be given three options and near total creative freedom to produce some output that responds to the prompts below.

- **Option 1 (Visual)**: create an interactive, generative sketch using p5.js (or a similar web-based framework). For example, the user controls precipitation on an alien planet using their mouse and keys and watches generative plant forms take shape in response. Alternatively, take inspiration from the p5.js community library.
- **Option 2 (Audio)**: create an interactive, generative web-based instrument using Tone.js. Examples of interactivity could include using a mouse or trackpad to simulate a theremin. By 'generative' we mean that the tool/instrument should incorporate some unpredictability by design. For example, if you're creating an interactive theremin, an algorithm could modulate the sound envelope or add effects.
- **Option 3 (Bonus)**: for participants who don't feel sufficiently challenged by the above options, this task asks you to combine the above two options. Using the examples given, one option could be to give both rain and plant life unique sounds that modulate as the plants grow.

More examples may be provided at the beginning of the workshop, but the goal is just to build something interactive and generative in nature, using LLMs and a library most participants won't be intimately familiar with, and to have fun doing it.

2.2. Part 2

The second half of the workshop will be guided, open discussion on the pairs' experiences using LLMs for creative coding. The researcher will pose some questions to generate discussion, but the goal is to reflect on the benefits LLMs provide for learning and creative tasks, as well as how current ACAs can be better designed to align with users' needs and goals. Example discussion questions:

- Those of you for whom this was the first time using p5.js or Tone.js, to what extent did you come away from that exercise feeling like you learned something about those libraries?

  - What exactly about (a) how the LLM communicated, (b) how the ACA you used is designed, or (c) how information is presented in its UI felt like it contributed to or hindered learning?
  - How important do you think learning is for a scenario like this?
  - For situations like this where learning the underlying language and frameworks has little bearing on your success in your day job, what is the bare minimum that LLMs and ACAs should do to foster understanding and prioritise learning?
    * What's different when it comes to your own work?

- Was there anything you felt you wanted more control over? Anywhere the LLM felt like it was overstepping its boundaries?
- For the less experienced coders, was there anything you saw a more experienced programmer doing that you would not have thought to do yourself? Anything you felt like you learned or took away from seeing someone more experienced interact with an ACA?
- For the more experienced coders, was there anything you saw less experienced programmers doing that you would consider to be less-than-ideal practice?

2.3. Participants

Ideally 8-16 participants. Open to anyone except total beginners. A growing body of research shows that novices and expert developers may really be impacted by the same UX issues, but that they are compounded for novices. However, for the purposes of workshop efficiency, participants should be familiar with the basics of coding (e.g., how to install packages, what a 'for' loop is, etc.) and should have some experience with ACAs.

2.4. Equipment

Ideally a large screen to present on / allow all participants to observe the same content or interactions at the same time, as well as a whiteboard or sticky notes to jot down ideas. Participants should have their own personal laptops on which to code or use collaborative tools like Figma or Miro. Ideally for coding participants would have access to a Claude subscription in order to use Claude Code, or something comparable. However, as a fallback, we can see how far we can get on Cursor's or Windsurf's free tier, or see if participants are willing to sign up for Cursor Pro's 7-day trial. The last resort would be using Google AI Studio, which has a generous free tier but is not a true ACA.

3. Goals + Expected Outcomes

The goal of this workshop is, first and foremost, to have fun, hence the creative, abstract nature of the activity, and working in pairs on something silly and open-ended is more enjoyable than working solo and can generate richer discussion. The more serious goal is to use that shared experience as a catalyst for collective reflection. The discussion in the second half gives participants a chance to pool their observations and frustrations about coding with AI into a blue-sky conversation about what better-designed tools might look like. For myself as the researcher, I expect the session to serve as inspiration on where to take my research next, rather than it resulting in concrete data to use in a research study.

Certifiably Curious: Why do Second-Level Teachers Engage in Industry AI Certification?

Miriam Harte William Clifford Roisin Faherty Nikhitha Guttula

Rajesh Jaiswal Musfira Jilani Na Li Joyce Mahon

Keith Nolan Keith Quille

School of Enterprise Computing and Digital Transformation
TU Dublin

{Miriam.Harte, William.Clifford, Roisin.Faherty, Nikhitha.Guttula, Rajesh.Jaiswal, Musfira.Jilani, Na.Li, Joyce.Mahon, Keith.Nolan, Keith.Quille}@tudublin.ie

Abstract

Over the past two years, the CSinc Research Centre at TU Dublin has facilitated several NVIDIA-certified workshops, specifically on The Fundamentals of Deep Learning. Prior to these Artificial Intelligence workshops, other professional learning events which the research team facilitated, typically had very low attendance (5-8 teachers per workshop), where these included events on the upper second-level computer science curriculum covering more general topics such as data analytics or Python programming. Conversely, the NVIDIA workshops were very well attended by second-level teachers, with around 100 teachers attending each workshop.

Given the substantial increase in attendance, we aimed to investigate what motivates teachers to engage with this type of technical Continuous Professional Learning workshops. To this end, we conducted several surveys with participants, investigating why they elected to attend an industry-level certification workshop, their self-reported importance of AI concepts in the classroom and, finally, what follow-on resources and professional learning events they would like in order to support their teaching. Our initial results indicate that improving technical understanding is a priority for teachers (even over industry-level certification itself). Furthermore, teachers prioritised technical learning for themselves, but ethical/human-centred AI for students. A positive note is that teachers predominantly report that they believe that human-centred AI is the most important concept in second-level Artificial Intelligence education. Supports which teachers would like to see developed included: linking content to the curriculum; an introductory approach to Machine Learning and the generic use of Artificial Intelligence.

As the integration of AI into the evolving Irish upper second-level curriculum continues to develop, alongside an increasing emphasis on human-centred approaches to AI design and application, there is a growing need for structured Continuous Professional Learning (CPL) initiatives supported by accessible resources and pedagogically grounded supports for teachers.

1. Introduction

The landscape of Irish second-level education is currently undergoing a period of significant transformation with the redevelopment of the Leaving Certificate Computer Science (LCCS) curriculum. As outlined in the National Council for Curriculum and Assessment (NCCA) draft specification (National Council for Curriculum and Assessment (NCCA), 2024a) in October 2025:

> *The study of computer science deepens the students' awareness of the ethical and social role of computers in society, supporting them to become informed users and creators of technologies.*

The curriculum is made up of three related strands, ***Practices and Principles, Core Concepts, and Computer Science in Practice***. These challenges are best addressed by making the teachers feel supported, specifically by improving self-efficacy (Faherty, Quille, & Becker, 2022). Many second-level teachers taking on this subject are out-of-field teachers, and some may not have much experience with topics such as AI and Machine Learning (ML). To gauge interest for supporting the upcoming curriculum in the area of AI, an NVIDIA industry certification workshop was piloted, covering The Fundamentals of Deep Learning (NVIDIA, n.d.). This one-day in-person workshop introduced learners to the methods and tools used to train a deep learning model, through a combination of lectures and data-focused activities. It was envisioned that only a handful of teachers would attend (given prior experience and attendance at other workshops the team delivered). However, almost 100 teachers registered.

The workshop introduced attendees to the basics of what is present in AI today which included the past of AI, the base elements of neural networks and their training. Teachers were also exposed to more advanced topics such as convolutional neural networks, data augmentation techniques, and transfer learning, using industry-relevant tools and frameworks. These elements aimed to provide a structured yet accessible entry point into complex AI concepts for teachers with limited prior experience in LCCS. The unexpectedly strong response to this optional professional learning event provided the impetus for the research presented in this paper.

To this point, we propose three **Research Questions (RQs)**:

> RQ1: Why do second-level teachers elect to attend professional development workshops focused on industry certification in deep learning?
>
> RQ2: How do teachers rank the importance of key artificial intelligence concepts for student learning?
>
> RQ3: What resources or learning events would teachers like, in order to support their teaching of AI in the classroom?

To answer these research questions, we developed a survey targeting different second-level teacher/school demographics. Both quantitative and qualitative analysis has been conducted. It is worth mentioning that, although Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) are technically distinct concepts, they are at times discussed interchangeably in this study when reflecting participant language and perceptions arising from the workshop context.

2. Literature Review

2.1. Computing Education in Ireland

A 2022 report from Lero, the Irish Software Research Centre, found that only a small proportion ($\sim$ 15%) of teachers delivering Computer Science (CS) held formal qualifications in the subject (Lero, the Irish Software Research Centre, 2022). Similarly, a cross-jurisdictional survey of the UK and Ireland indicated that Irish second-level teachers were significantly more likely than their UK counterparts to teach Computer Science without subject-specific qualifications (Sentance et al., 2022). In practice, the majority of LCCS classes in Ireland continue to be delivered by experienced second-level teachers who have transitioned into the subject from related disciplines, rather than by fully specialised Computer Science teachers (Lero, the Irish Software Research Centre, 2022; Department of Education, 2021).

Irish second-level education culminates in the Leaving Certificate (LC), and most Irish second-level students sit this examination (where ages typically range from 16-18 years old), selecting six to eight subjects from over 30 syllabi . Usually subjects offer examinations at two levels: Ordinary Level (OL) and Higher Level (HL). For most school leavers, LC performance is the main factor determining university entrance. LC exams are set and assessed centrally by government-approved bodies, and are typically studied during the final two years of second-level schooling, equivalent to grades 11-12 (the final two

years of high school) in the United States and many other regions (Brown et al., 2013; Faherty, Nolan, Quille, Becker, & Oldham, 2023).

Following extensive research (Keane & McInerney, 2017; Lehane, 2019), LCCS (National Council for Curriculum and Assessment and the Department of Education, Ireland (NCCA), 2023; National Council for Curriculum and Assessment (NCCA), 2024a) was introduced in 2018, marking a significant milestone in the Irish education system (National Council for Curriculum and Assessment (NCCA), 2024b; Faherty et al., 2023; Connolly, Byrne, & Oldham, 2022; Borges, McInerney, Exton, Harkin, & McGarr, 2024; Becker et al., 2023). The provision of LCCS in Irish second-level schools has been developing since 2015, when the Digital Strategy for Schools was published and highlighted the importance of coding and digital skills within the curriculum. 2018 then saw the introduction of LCCS as a full Leaving Certificate subject, initially rolled out to a selected group of pilot schools before broader national expansion (Department of Education and Skills, 2015; National Council for Curriculum and Assessment, 2018).

2.2. Lack of AI in CS Curriculum

In the Irish second-level context, AI is often introduced abruptly in non-formal settings, such as short-term workshops or extracurricular sessions, rather than through structured classroom instruction. Without explicit assessment requirements or detailed guidance, many schools rely on ad hoc experiences such as outreach initiatives or brief teacher-led activities to introduce AI concepts. Providers of online classroom supports for CS teachers in Ireland include *Scoilnet* (Scoilnet, 2025), *CS$_{LINC}$* (CS INC Team, 2025) [1] and *Oide* (Oide, 2023). [2]

Support for AI and digital competence is increasingly visible in Irish policy documents (Department of Education, Ireland, 2022; Department of Enterprise, Trade and Employment, Ireland, 2021; Government of Ireland, 2022a; Department of Education, Ireland, 2024; Government of Ireland, 2022b) alongside European-level frameworks (European Commission, 2022a, 2022b; Organisation for Economic Co-operation and Development (OECD), 2024; Miao & Holmes, 2023; European Commission and OECD, 2025). There are also examples of innovative use of GenAI in Irish universities (Schalk et al., 2025), and supports [3], guidance The *New Horizons for Higher Education: Teaching and Learning with Generative AI* report provides clear guidance and direction aimed at assisting institutional leaders shape policy and strategy for GenAI integration into teaching practice and analysis [4] is emerging in other contexts.

2.3. CS CPL in Ireland

A search of currently available Continuous Professional Learning (CPL) courses in CS was completed. The Education Support Centre Ireland (ESCI) (*Education Support Centres Ireland*, n.d.) website was used to gain insight into the types of computing-related courses are on offer to primary and post-primary teachers, see Table 1. "Computer Science" yielded two workshops related to training teachers for the Leaving Certificate curriculum. "Coding/Programming" highlighted sessions on basic visual programming language (Scratch). "AI" produced the highest number of results. None of these CPL offerings were directly related to how AI functioned nor did any relate to ML or statistics content. All of these

[1] A range of free online courses are offered on the *CS$_{LINC}$* platform, including an AI/ML course, which is targeted at TY students (Mahon, Quille, Mac Namee, & Becker, 2022).

[2] A series of initiatives aimed at supporting teachers to integrate AI into education were launched in 2024. These included the *AI4EDU* pilot workshops held between March and December 2024 `https://ai4edu.eu/2024/03/14/irish-teachers-workshops-march-2024-dec/`[Workshops], and a large-scale educational AI pilot involving 300 teachers across 48 Irish schools. `https://www.lwetb.ie/mullingar-community-college-pioneers-ai-tech-in-classroom`[AI Pilot].

[3] On the 10th June, 2025, the Department for Education in the UK published a set of free online training materials to support the safe and effective use of GenAI in education `https://www.gov.uk/government/collections/using-ai-in-education-settings-support-materials` [Training materials].

[4] The *Coding Ireland* STEM Report 2025 aims to evaluate the current landscape of STEM and Digital Skills education across Ireland. It is based on a 2025 survey (19th March - 9th May), with comparative data from their 2024 survey. Findings include: 85% of post-primary teachers surveyed cite lack of knowledge/training as a major barrier, only 44% have the necessary equipment, and only 24% report sufficient funding. 61% of schools expect STEM's presence to grow over the next decade, 75% of schools view STEM & Digital Skills as very/extremely important (Coding Ireland, 2025).

were related to using AI as a tool, whether it was for building rubrics, lesson plans, or brainstorming. "Math/statistics" did not have crossover with AI or machine learning. These CPL courses were related to teaching a mathematics class or helping students in this space.

Search Term	No. of Results	Notes
Computer Science	2	Courses and workshops for introduction to CS for the leaving certificate content. Including online resources and pedagogical practices.
Coding/Programming	1	Basic how-to on Scratch
AI	127	All of these results are courses on how to use AI to assist in teaching.
Maths/Statistics	8	Sessions on supporting tools and advice with respect to teaching mathematics.

Table 1 – Search results for teacher CPL courses related to Computer Science.

2.4. Summary

Given the current combination of factors within Ireland's Computer Science landscape at second-level, it is clear that there are several hurdles in place for teachers to build confidence in delivering new LCCS topics in the classroom. While a large amount of CS teachers are considered to be out-of-field, the lack of CPL opportunities further limits their chance to upskill and gain adequate teaching experience in the topic. Furthermore the AI strand of the newly updated LCCS curriculum will not only add to the technical workload of teachers, it may also further distance teachers from a feeling of self-efficacy in relation to teaching AI concepts in the classroom. We don't yet have adequate insight into the full extent of supports and resources which teachers require in order to successfully embed AI into their teaching of LCCS. Through this study, we will explore some teachers' requirements around achieving this goal.

3. Methodology

3.1. Survey Development

After several iterations of this NVIDIA workshop on The Fundamentals of Deep Learning, we aimed to better understand why teachers were interested in participating in the first place. A survey was developed and distributed among all teachers who had registered for this workshop between 2024 and 2026. The survey was structured in a way that teacher attendance on the day was not a requirement for their feedback, rather their interest in the topic itself. Survey questions can be found in the Appendix. Aside from teacher/school demographics, three research questions were posed. For RQ1, teachers were asked about their primary reason for registering for this workshop. This was an open-ended question. As part of RQ2, we then asked teachers to rank in order of preference "the most important concepts of AI which students should learn". For RQ2, we provided a list of eight possible responses, which the research team selected based on previous face-to-face discussions with teachers and other researchers within the field, as well as several face validity sessions of refinement. Finally, for RQ3, we asked teachers what they would like to learn more about, "in order to support their teaching in the classroom". In the case of RQ3, we asked teachers to answer this qualitative question from the perspective of any "specific resources or learning events". In terms of survey analysis, responses to RQ1 and RQ3 were thematically analysed by the research team and sorted as such based on overall similarities in topic. RQ2 was analysed based on response ranking the frequency of response selection at this ranking.

3.2. Quantitative and Qualitative Analysis

The analytic approach for each research question was selected based on the type of the corresponding survey items, and the following sections discuss this approach framed by each research question.

3.2.1. Analysis of RQ1: Motivations for Attending Professional Development

To address RQ1, responses to the open-ended survey were analysed using thematic analysis following the approach outlined by Braun and Clarke (Braun & Clarke, 2006). Three researchers acted as raters

and engaged in a collaborative, reflexive analytic process consistent with Braun and Clarke's guidance (Braun & Clarke, 2021). Analysis took place over three one-hour sessions. During the first session, raters familiarised themselves with the data and generated initial inductive codes. In the second session, codes were discussed and grouped into candidate themes. In the final session, themes were reviewed, refined, and agreed upon through discussion until consensus was reached. This process emphasised shared interpretation rather than statistical measures of inter-rater reliability, to which Braun and Clarke suggest over blind inter-rater investigations.

While theme frequencies were recorded (as presented in section 4.1) to support transparency and coherence in theme development, the analysis focused on understanding the range and nature of teachers' motivations rather than on quantitative comparison. Examples of the theme "Technical Understanding" included:

> *To get an idea of what AI was all about and what was behind it.*

and

> *I wanted to learn more about NVIDIA and AI. AI is changing all the time.*

An example of "NA / Not Valid" where we simply could not apply a theme, was:

> *It was an interesting opportunity to attend and learn. I was unable to attend in the end. But would love an opportunity to attend in the future if it was run again. Thank you for the opportunity.*

3.2.2. Analysis of RQ2: Ranking of AI Concepts

RQ2 was examined using responses from a structured ranking question. Participants were asked to identify and rank AI concepts from a predefined list. The analysis involved aggregating ranking selections to identify relative prioritisation across concepts. Frequencies were calculated for inclusion within the top three rankings, as well as for the last ranking frequency. This approach enabled examination of both highly prioritised concepts and those that were infrequently selected, providing insight into areas teachers perceive as more or less important for second-level AI education.

3.2.3. Analysis of RQ3: Desired Learning Events and Supports

To address RQ3, responses to the open-ended survey were analysed using Braun and Clarke's thematic analysis approach (Braun & Clarke, 2006, 2021). The same three raters participated in the analysis, following an identical three-session structure to that used for RQ1.

During theme refinement, responses were organised into two higher-level categories that emerged from the data (inductive approach (Braun & Clarke, 2006)): *Learning Events* (L) and *Supports/Resources* (S). Learning events captured requests related to professional learning experiences, instructional focus, and curricular alignment, while supports/resources referred to classroom-ready materials, examples, activities, and implementation aids. Responses that could not be meaningfully coded were categorised as not.

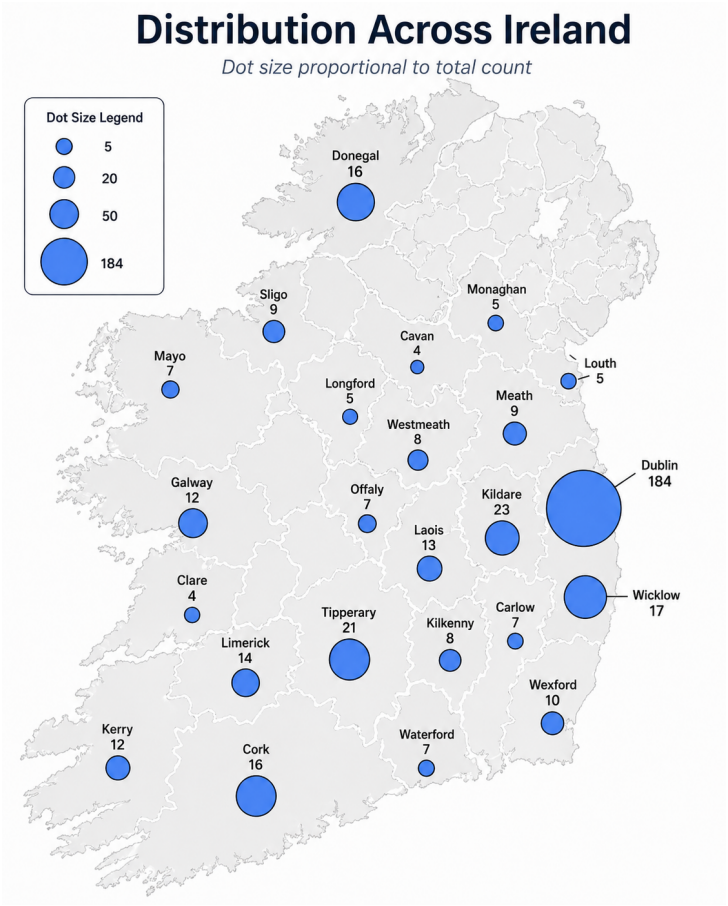


Figure 1 – Geographical distribution of registered participants across Ireland. Dot size is proportional to the total number of registrations from each county.

An example of the theme "L: Generic AI use" included:

> *How to use it to build teaching resources and activities. How to teach students to use it effectively for course work, activities and assessment.*

An example of "NA" was:

> *I am very open-minded when it comes to areas of study and continuous professional development. In-person learning events are the most beneficial in my opinion.*

An example of "S: Resources":

> *More student-level friendly resources and activities*

Finally, an example of "L: Maths for AI":

> *More of the maths behind the model for the students who want to engage on a deeper level*

3.3. Profile of Teachers' Schools

This section outlines the demographic profile of the schools in which teachers, who registered for the training, teach. We collected the unique school ID (role number) and then mapped to a Government school database to obtain school demographic data. Three workshops were delivered during the 2024–2026 academic years. In total, 525 participants registered across the three sessions. Teachers were drawn from across Ireland, with the geographical distribution of registrations illustrated in Figure 1. The dot size in the figure is proportional to the number of registrations from each county.

Where school-level information was provided ($\sim$ 430), participants represented a range of school contexts, including mixed, all-boys and all-girls schools; DEIS and non-DEIS schools (DEIS stands for

Table 2 – Summary of school characteristics

Variable	Category	Total
Gender	Mixed	314
Gender	Boys / Male	58
Gender	Girls / Female	57
DEIS status	DEIS	69
DEIS status	non-DEIS	261
Language medium	Gaelscoil	11
Language medium	English	419
Fee-paying status	Fees / Fee-paying	37
Fee-paying status	Free / Non-fee-paying	393

"Delivering Equality of Opportunity in Schools" and represents schools which have been categorised as educationally disadvantaged by the Department of Education [5]); English-medium and Irish-medium schools; and fee-paying and non-fee-paying schools. The distribution of these school characteristics is shown in Table 2.

Overall, the registration demographics indicate that the workshops attracted a broad national audience, with participants registering from across multiple counties. The second-level school profile data also suggests diversity across participating school contexts. Registrants included participants from mixed, all-boys and all-girls schools, as well as DEIS and non-DEIS schools, English-medium and Irish-medium settings, and fee-paying and non-fee-paying schools. This indicates that the workshops was of interest to a varied educational audience. This demographic data provides a profile of those who registered for the workshops and highlights the range of educational contexts interested in this training. It should also be noted that totals vary across categories because not all participants provided complete demographic or school-level information during registration.

3.4. Study Limitations

- There were more than 500 registrations and 400+ schools. However, engaged survey responses were considerably lower (n=34).
- Bias: Teachers who registered for this workshop and are already interested in AI completed these surveys. Hence our findings may over-represent highly motivated or AI-positive teachers (but not all Irish teachers).
- Research Question 2 (RQ2) used pre-defined categories.

4. Results

This section presents the results in relation to the three RQ's. Section 5 subsequently analyses each set of findings, first in relation to the individual RQ's and then holistically, in order to offer a broader interpretative perspective.

4.1. RQ1: Why do second-level teachers elect to attend professional development workshops focused on industry certification in deep learning?

Table 3 presents the frequency of themes that were reported by teachers. Some open-ended responses included more than one theme, thus the total frequency of themes reported exceeds the $n = 34$ teacher responses. The highest number of themes reported from a single teacher response was four, where 61% of responses only reported one theme.

[5] https://www.gov.ie/en/department-of-education/policy-information/deis-delivering-equality-of-opportunity-in-schools/

Table 3 – Themes and Frequencies for RQ1

Theme	Frequency
Technical understanding	24
Certification	6
General AI application	5
Professional networking	3
Teaching about AI	3
NA / Not valid	2
Ethical AI	1
Confidence for teaching	1
Use cases	1
Student future jobs	1

For the theme **Technical Understanding** some of the teachers quoted:

> *"To gain further insight into machine learning, LLMs and data science."*

and

> *"The opportunity to learn more about AI as well as NVIDIA certification would have been a huge bonus."*

For the theme **NA / Not Valid**, there were two responses, where one teacher did not attend, and the other quoted (a note of thanks):

> *"It was an interesting opportunity to learn. I was unable to attend in the end. But would love an opportunity to attend in the future if it was run again. Thank you for the opportunity."*

4.2. How do teachers rank the importance of key artificial intelligence concepts for student learning?

To answer RQ2, Figure 2 indicates distinct differences in how teachers prioritised the importance of key AI concepts for student learning in second-level schools across first, second and third rankings. Overall, the theme of "*The human-centred nature of AI, i.e., ethics and bias*" was the first-place votes ($n = 15$), as well as the highest cumulative top-three ranking ($n = 23$) (see Table 4). This also demonstrates that there are 15 out of 34 teachers who listed it as a strong preference for learning the concepts of ethical and societal AI.

The second most prioritised cumulative top three ranking theme was *"A broader understanding of where AI can be applied across the world"*, which received a high number of top 3 votes ($n = 17$), whereas relatively first-place ranking ($n = 3$). However, it had the most votes in second ranking ($n = 8$) and the second most votes in third ranking ($n = 6$). It shows that teachers perceive this theme as broadly important for students, though not their top priority.

Meanwhile, ranking results of the theme *"The general use of AI in education (i.e., ChatGPT in the classroom, in any subject)"* showed middle rankings of importance of key AI concepts at first, second and third ranking ($n = 5, 7, 4$), resulting in a third highest overall top-three ranking ($n = 16$).

Two themes, *"Understanding and evaluating AI models"* (Top-three: $n = 14$) and *"Using AI tools in a CS classroom, in order to support programming or other CS activities"* (Top-3: $n = 11$), are mid-ranks at overall top-three ranking list. In particular, the theme *"Understanding and evaluating AI models"* has been voted as the second priority concept. The results indicate that both conceptual understanding and applied classroom integration are of moderate interest; teachers might have applied these in their teaching activities, though they are not considered top priorities.

Lower-ranked themes are related to AI techniques: *"Using AI tools in a CS classroom, in order to support programming or other CS activities"* (Top three: $n = 11$) and *"Building AI from scratch in*

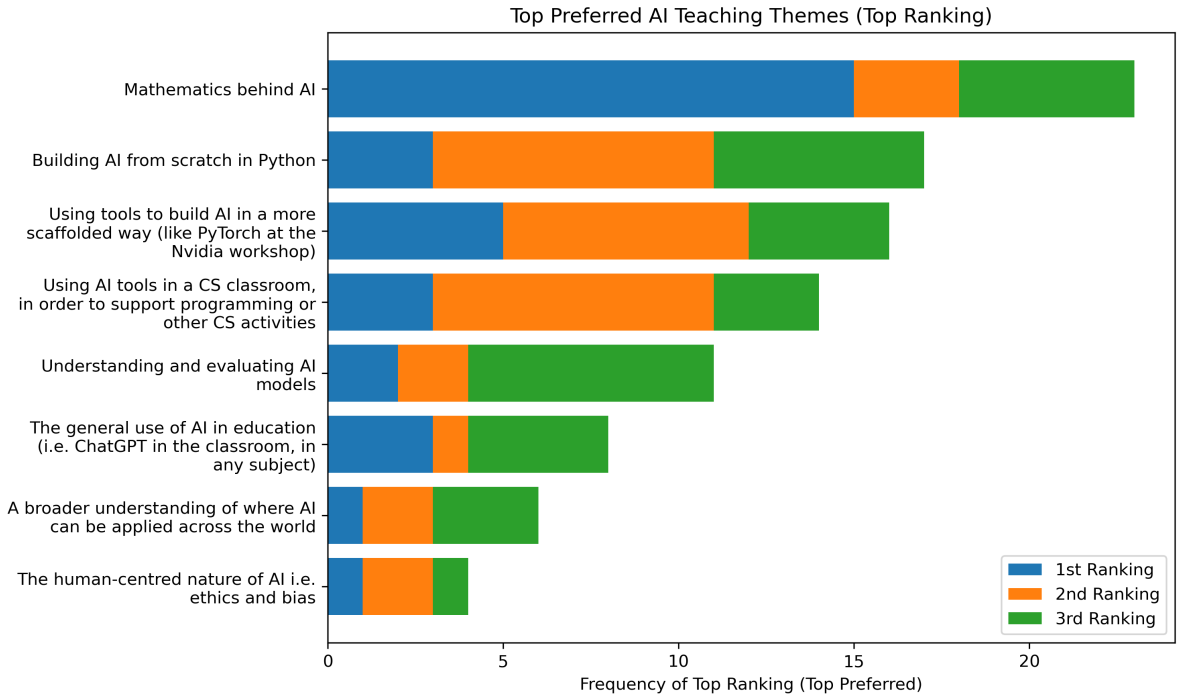


Figure 2 – Top Frequency Ranking AI Teaching Themes

Python" (Top-three: $n = 6$). These can indicate that demands for AI techniques and developments are of lower importance in post-primary education. Finally, *"The mathematics behind AI"* is ranked lowest overall (Top three: $n = 4$), with only one teacher ranking it as first-place.

To further investigate teachers' preferences of learning AI, Table 5 illustrates that the distribution of last-place rankings, based on the concepts which teachers are least curious to learn about. Eight teachers each ranked *"AI from scratch in Python"* and *"The mathematics behind AI"* as the lowest preferences, which suggests that these technically intensive topics are the least appealing. Followed by *"Using tools to build AI in a more scaffolded way (like PyTorch at the NVIDIA workshop)"* ($n = 5$), while *"The general use of AI in education (i.e., ChatGPT in the classroom, in any subject)"* and *"Using AI tools in a CS classroom, in order to support programming or other CS activities"* ($n = 4$ each) ranked mid-levels of last-place rankings, which is similar to results of top-preferred rankings.

In contrast, in the case of the themes *"A broader understanding of where AI can be applied across*

Table 4 – The frequency of Top Themes for RQ2

Responses	**Top 3**	**1st**	**2nd**	**3rd**
The human-centred nature of AI (i.e. ethics and bias)	23	15	3	5
A broader understanding of where AI can be applied across the world	17	3	8	6
The general use of AI in education (i.e. ChatGPT in the classroom, in any subject)	16	5	7	4
Understanding and evaluating AI models	14	3	8	3
Using AI tools in a CS classroom, in order to support programming or other CS activities	11	2	2	7
Using tools to build AI in a more scaffolded way (like PyTorch at the NVIDIA workshop)	8	3	1	4
Building AI from scratch in Python	6	1	2	3
The mathematics behind AI	4	1	2	1

Table 5 – The frequency of the last ranking for RQ2

Responses	**Frequency of the Last Ranking**
Building AI from scratch in Python	8
The mathematics behind AI	8
Using tools to build AI in a more scaffolded way (like PyTorch at the NVIDIA workshop)	5
The general use of AI in education (*i.e.*, ChatGPT in the classroom, in any subject)	4
Using AI tools in a CS classroom, in order to support programming or other CS activities	4
A broader understanding of where AI can be applied across the world	2
The human-centred nature of AI (i.e. ethics and bias)	2
Understanding and evaluating AI models	0

the world" and *"The general use of AI in education (i.e. ChatGPT in the classroom, in any subject)"* ($n = 2$ each), results showed that only two teachers each listed these two topics as the least significant topics, indicating relatively higher acceptance. Notably, no teacher ranked the theme *"Understanding and evaluating AI models"* in last place ($n = 0$), indicating that it is valued by all teachers, although it was at mid-ranking in the Top-three ranking list (see Table 4). For the details of the number of last-place rankings, see Table 5.

4.3. What resources or learning events would teachers like, in order to support their teaching of AI in the classroom?

Table 6 shows the frequency of themes identified from teachers' responses about their learning needs and the supports required for teaching AI in the classroom. Responses were categorised into two overarching themes: **Learning Events** and **Supports/Resources**, with several teachers identifying multiple themes within a single response. The most frequently identified learning needs related to generic AI use, introductory approaches to machine learning, and linking AI content to the LCCS curriculum – each reported six times. In terms of supports, requests for practical examples were most prominent, indicating a strong demand for concrete, classroom-ready materials.

Within the **Learning Events** theme, teachers highlighted the importance of developing a stronger understanding of AI concepts and building confidence in their own knowledge:

> *"I want to dig as deep as I can into this area so that I can understand it in such a way that I can teach my students confidently ..."*

This was further reflected in concerns around the timing and use of AI tools in the classroom:

> *"If we introduce AI tools too early ... we risk skipping over the fundamentals ... how to think, how to design, how to persevere, how to debug ... More emphasis needs to be on the 'about AI' rather than the 'how to use it'."*

In contrast, responses within the **Supports/Resources** theme focused on the need for practical examples and classroom-ready materials:

> *"Examples of where teaching of AI in the classroom works and examples where it doesn't. Methods that can be used and resources."*

Examples of responses which were considered **NA/Not Valid**, include:

> *"I am very open minded when it comes to areas of study and continuous professional development. In person learning evens are the most beneficial in my opinion."*

Table 6 – The frequency of requested learning events and resources

Learning Events and Supports/Resources	Frequency
L: Generic AI use	6
L: Introductory Approach ML	6
L: Linking to LCCS	6
L: Additional NVIDIA	3
L: Maths for AI	3
L: AI in Block Based Environments	2
L: Generic Ethical use	2
L: AI in Primary	1
L: Ethics for LCCS	1
L: GenAI for Coding	1
L: General Understanding	1
L: Introductory No/Low Code ML	1
L: More Depth	1
S: Examples	4
S: Applications in Real World	2
S: Resources	2
S: Activities	1
S: Additional Models	1
S: Pre-made Classes (i.e. lesson plans, classroom activities)	1
S: NA/Not Valid	5

and

"I would love to attend another event as I missed the February one"

5. Discussion

This section approaches the discussion through a dual analytical lens. Firstly, it examines the implications of our findings in relation to each individual research question, with particular attention to their significance for teacher professional learning in the context of teaching about AI. This involves critically interpreting how the results inform current understandings of reasons to attend, instructional practices, and the supports required for educators to effectively engage with AI-related content in classroom settings.

Secondly, this section brings together these discrete insights to construct a more holistic, integrative perspective. By drawing together the outcomes associated with each research question, it provides an "aerial view" of the broader implications for the LCCS with respect to teacher professional learning. This is of value to the authors as a research centre as well as national professional learning providers where this may also be applicable to providers internationally, where AI is being integrated into second-level curricula.

5.1. RQ1

The findings for RQ1, which examine the motivations underpinning educators' attendance at an NVIDIA CPL workshop, reveal a clear prioritisation of technical capacity-building over pedagogical or ethical considerations, substantially outweighing all other categories. This strong emphasis suggests that educators perceive a significant need to develop foundational and advanced knowledge of AI systems, likely reflecting both the novelty of the domain and perceived knowledge gaps in their existing professional preparation. From a professional learning perspective, this indicates that teachers are primarily motivated by content mastery and conceptual clarity when engaging with AI-related CPL opportunities.

A secondary, though considerably less prominent, motivation relates to industry-level certification (n = 6). We initially hypothesised that this would be the most popular reason for attendance, as this NVIDIA certification is typically valued at $500, which could have been perceived as a strong motivator. Nevertheless, this finding points to the instrumental value placed on formal recognition of learning and the value of the workshop (as it was free for teachers) or its perceived monitory value.

Professional networking was not expected to feature as highly, and although a small number reported it, teachers perceive that there is value in attending an event like this to network with their peer. While this may be applicable to many forms of professional learning, it was reported as important to some teachers in this particular context.

Notably and finally for RQ1, themes relating to broader societal and educational implications, such as ethical AI (n = 1), are minimally represented. The scarcity of references to ethical considerations is particularly striking given the prominence of ethics in AI discourse and in both the current and the draft NCCA specification. This could indicate a gap in teacher awareness or prioritisation of responsible AI practices within their professional learning trajectories.

5.2. RQ2

Analysis in Section 4.2 showed that concepts related to human-centred and application-oriented AI topics are more important for teachers than technical and theoretical AI content. These themes are related to the general use of AI and tools in education, indicating that teachers value relevant practice, rather than simply focusing on a methodology of building AI systems. In other words, teachers may be more interested in understanding where AI can be applied and how it can be integrated into their teaching and learning environments.

From the results of what teachers ranked last (least important), building AI from scratch in Python and understanding mathematics in AI themes were the lowest ranked. In particular, in the case of the theme of understanding and evaluating AI models, teachers still recognise the importance of interpreting and assessing AI systems, though they may not need to apply AI technical implementation. To this point, it is clear that there is a potential need for understanding and evaluating approaches to applying or teaching AI concepts through classroom activities, but not purely AI technical training or instruction.

Therefore, the results of rankings from teachers suggest that a human-centred approach to AI in education, combining ethics, applications and evaluating AI technical understanding may be the most effective approach to match teachers' needs within an industry AI training setting. Interestingly, this aligns closely with the new NCCA draft LCCS curriculum.

5.3. RQ3

The results of the frequency of requested learning events and resources reflect that teachers have strong motivations to learn generic examples and foundational understanding, as they need to follow the LCCS syllabus when teaching students. We can see this from the top three results of learning events in Table 6. Moreover, this is further re-enforced with the highest result in the supports/resources section, regarding examples. The second highest, "Applications in Real World" and "Resources", also serve as a good contextual cue for the classroom. Currently most CPL courses regarding AI are related to the use of AI (see Section 2.3). From our results, it shows that teachers are more interested in the practical foundations and examples of AI and ML.

6. Conclusion

The findings from this study provide early insights into second-level teachers' motivations, priorities, and professional learning needs related to AI education within the Irish curriculum context. When considered in conjunction, the findings from RQ1 and RQ2 present an apparent tension regarding the role of ethical and human-centred AI (HCAI) in teacher priorities. Specifically, while ethical and human-centred considerations are largely absent in teachers' self-reported motivations for attending CPL (RQ1), they emerge as comparatively highly valued when teachers are prompted to rank AI-related topics (RQ2). This divergence may be interpreted as a confounding finding; however, a closer analysis suggests a more nuanced explanation rooted in teachers' conceptual framing, experience, and professional learning needs.

One plausible interpretation is that this discrepancy reflects a distinction between implicit valuation and explicit articulation. In RQ1, where responses were open-ended, teachers overwhelmingly foregrounded technical understanding, likely because this represents a perceived deficit area in their knowledge base. Ethical AI, by contrast, may not be spontaneously articulated as a priority because it is either:

(a) less well understood as a distinct domain,

(b) perceived as secondary to foundational knowledge, or

(c) subsumed within broader notions of "application" or "responsible use" without being explicitly named.

This suggests a potential issue of language and conceptual awareness, where teachers may value HCAI principles but lack the disciplinary vocabulary or confidence to express these as standalone professional learning goals. In contrast, RQ2 required teachers to engage with predefined categories, making ethical and human-centred AI more visible as constructs. Within this structured context, teachers rated these dimensions as important, indicating that when prompted, they recognise the relevance of ethical considerations, application contexts, and evaluation of AI systems. This aligns with a broader shift away from technical implementation (as present in the draft LCCS specification) towards interpretive, applied, and socially situated understandings of AI, an orientation consistent with HCAI frameworks.

This interpretation is further nuanced when RQ3 is considered, as it provides insight into the specific types of learning supports and resources teachers perceive as valuable. The most frequently identified needs were generic AI use, introductory machine learning approaches, and explicit linking to the LCCS curriculum (each n = 6). These reinforce the centrality of accessibility and contextual relevance in professional learning design. Teachers are not merely seeking abstract knowledge but are prioritising entry points that translate AI concepts into classroom practice.

Notably, ethical AI appears only marginally in these responses, further supporting the pattern observed in RQ1: ethics is not foregrounded as an immediate support need. However, its presence, albeit limited, suggests that when situated within concrete teaching contexts, it gains relevance. Collectively, RQ3 indicates that teachers' priorities centre on usable, curriculum-aligned, and scaffolded resources.

Taken together, the combined findings across RQ1–RQ3 suggest not a contradiction, but a misalignment between teachers' immediate perceived needs and their broader professional priorities. Teachers appear to prioritise technical and introductory access points, alongside practical teaching supports, while simultaneously recognising (when prompted) the importance of ethical and human-centred AI. For LCCS and teacher professional learning in Ireland, this underscores the importance of designing CPL that is explicitly structured, pedagogically grounded, and integrative. Ethical AI should not be treated as an abstract add-on, but embedded within applications, examples, and curriculum-linked activities.

With all of the above in mind, we intend to refine how we engage with teachers about AI and ML, based on their requested resources and learning supports. We hope to restructure the pre-existing NVIDIA workshop to fit more closely with the preferred format and needs of teachers, particularly through greater

integration of the new LCCS AI content. Finally, it is our intention to delve deeper into further iterations of this research, with a larger cohort of educators and perhaps second-level students. This approach may prove especially insightful in correlation with the incoming AI strand of the LCCS curriculum.

7. Appendix

7.1. Survey Questions

1. *What was your primary reason for registering/attending this workshop? Please provide as many details as possible, even if you did not attend the workshop. (For example, did you want NVIDIA certification or to learn more about AI etc?)*
2. *In my opinion, the most important concepts of AI which students should learn about are: (Please rank in order of preference)*
    - *The mathematics behind AI*
    - *Building AI from scratch in Python*
    - *Using tools to build AI in a more scaffolded way (like PyTorch at the NVIDIA workshop)*
    - *Understanding and evaluating AI models*
    - *Using AI tools in a CS classroom, in order to support programming or other CS activities*
    - *A broader understanding of where AI can be applied across the world*
    - *The human-centred nature of AI i.e. ethics and bias*
    - *The general use of AI in education (i.e. ChatGPT in the classroom, in any subject)*
3. *What would you like to learn more about, in order to support your teaching of AI in the classroom? Are there specific resources or learning events which you feel would be of benefit?*

8. References

Becker, B. A., Bradley, S., Maguire, J., Black, M., Crick, T., Saqr, M., … Quille, K. (2023). Computing education research in the uk & ireland. In M. Apiola, S. López-Pernas, & M. Saqr (Eds.), *Past, present and future of computing education research : A global perspective* (pp. 421–479, Ireland, along with a scientometric study of research outputs.). Cham: Springer International Publishing. doi: 10.1007/978-3-031-25336-2_19

Borges, J., McInerney, C., Exton, C., Harkin, B., & McGarr, O. (2024). The subjective nature of curriculum innovation: The case of computer science students in upper secondary education in ireland. *Computer Science Education*, 1–29. (Advance online publication) doi: 10.1080/08993408.2024.2421680

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, *3*(2), 77–101. doi: 10.1191/1478088706qp063oa

Braun, V., & Clarke, V. (2021). *Thematic analysis: A practical guide*. London: SAGE.

Brown, N. C. C., Kölling, M., Crick, T., Peyton Jones, S., Humphreys, S., & Sentance, S. (2013). Bringing computer science back into schools: Lessons from the uk. In (p. 269–274). NY, NY, USA: ACM. doi: 10.1145/2445196.2445277

Coding Ireland. (2025). *Coding ireland stem report 2025* (Technical Report). Coding Ireland. Retrieved from `https://codingireland.ie/files/codingireland/Coding-Ireland-STEM-Report-2025.pdf` (Accessed: June 13th, 2025)

Connolly, C., Byrne, J. R., & Oldham, E. (2022). The trajectory of computer science education policy in ireland: A document analysis narrative. european journal of education. *Int. J. Comput. Sci. Educ. Sch.*, *57*(3), 512–529. doi: 10.1111/ejed.12507

CS INC Team. (2025). *Cs linc online learning.* Retrieved from `https://www.csinc.ie/Home/CSLINC` (Accessed: June 9th, 2025)

Department of Education. (2021). *Computer science teacher upskilling initiatives in ireland.* (Programme overview and policy documents)

Department of Education and Skills. (2015). *Digital strategy for schools 2015–2020.* Government of Ireland.

Department of Education, Ireland. (2022). *Digital strategy for schools to 2027: Enhancing teaching, learning and assessment.* Retrieved from https://assets.gov.ie/static/documents/digital-strategy-for-schools-to-2027.pdf (Accessed: June 9th, 2025)

Department of Education, Ireland. (2024). *Digital strategy for schools implementation plan 2024.* Retrieved from https://assets.gov.ie/static/documents/digital-strategy-implementation-plan.pdf (Accessed: June 9th, 2025)

Department of Enterprise, Trade and Employment, Ireland. (2021). *AI – Here for Good: National Artificial Intelligence Strategy for Ireland.* Retrieved from https://enterprise.gov.ie/en/publications/publication-files/national-ai-strategy.pdf (Accessed: June 9th, 2025)

Education Support Centres Ireland. (n.d.). https://www.esci.ie/cpd-courses.html. (Accessed: 2026-04-13)

European Commission. (2022a, October). *Ethical guidelines on the use of artificial intelligence (ai) and data in teaching and learning for educators.* Retrieved from https://op.europa.eu/en/publication-detail/-/publication/d81a0d54-5348-11ed-92ed-01aa75ed71a1/language-en (Publications Office of the European Union)

European Commission. (2022b). *Guidelines for teachers and educators on tackling disinformation and promoting digital literacy through education and training.* Retrieved from https://op.europa.eu/en/publication-detail/-/publication/a224c235-4843-11ed-92ed-01aa75ed71a1/language-en (Accessed: 2025-04-30)

European Commission and OECD. (2025). *Empowering learners for the age of ai: An ai literacy framework for primary and secondary education (review draft).* Paris: OECD. Retrieved from https://ailiteracyframework.org (Accessed: 2025-05-29)

Faherty, R., Nolan, K., Quille, K., Becker, B., & Oldham, E. (2023). A brief history of k-12 computer science education in ireland. In *International journal of computer science education in schools, 6(1), 3–34.* Retrieved from https://doi.org/10.21585/ijcses.v6i1.148

Faherty, R., Quille, K., & Becker, B. A. (2022). Comparing the programming self-efficacy of teachers using cslinc to those teaching the formal national curriculum. In *Proceedings of the 27th acm conference on innovation and technology in computer science education (iticse 2022).* Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3502717.3532130 doi: 10.1145/3502717.3532130

Government of Ireland. (2022a). *Esd to 2030: Second national strategy on education for sustainable development.* Retrieved from https://assets.gov.ie/static/documents/second-national-strategy-on-education-for-sustainable-development-esd-to-2030-ad1ca48f.pdf (Accessed: June 9th, 2025)

Government of Ireland. (2022b). *Harnessing digital: The digital ireland framework.* Retrieved from https://assets.gov.ie/static/documents/harnessing-digital-the-digital-ireland-framework.pdf (Accessed: June 9th, 2025)

Keane, N., & McInerney, C. (2017). *Report on the provision of courses in computer science in upper second level education internationally* (Tech. Rep.). National Council for Curriculum and Assessment. Retrieved from https://ncca.ie/media/2605/computer_science_report_sc.pdf (Accessed: 2025-04-14)

Lehane, P. (2019, January). *Factors to consider when developing a computer-based examination* (Tech. Rep.). National Council for Curriculum and Assessment. Retrieved from

https://ncca.ie/media/4081/lccs_cbe_factorstoconsider_lehane2019-for-ncca-website.pdf (Accessed: June 9th, 2025)

Lero, the Irish Software Research Centre. (2022). *Who is teaching computer science? a report on the demographics of computer science teachers in ireland.*

Mahon, J., Quille, K., Mac Namee, B., & Becker, B. A. (2022). A novel machine learning and artificial intelligence course for secondary school students. In *Proceedings of the 53rd acm technical symposium on computer science education v.2* (pp. 1155–1155). doi: 10.1145/3478432.3499073

Miao, F., & Holmes, W. (2023). *Guidance for generative AI in education and research* (Tech. Rep.). Paris: United Nations Educational, Scientific and Cultural Organization(UNESCO). Retrieved from https://unesdoc.unesco.org/ark:/48223/pf0000386693 (Accessed: June 9th, 2025)

National Council for Curriculum and Assessment. (2018). *Leaving certificate computer science specification.* NCCA.

National Council for Curriculum and Assessment and the Department of Education, Ireland (NCCA). (2023). *Computer science curriculum specification, leaving certificate, ordinary and higher level, updated 2023.* https://www.curriculumonline.ie/getmedia/cff6eb86-9ff8-4e68-abf9-e42ca637492d/LC-Computer-Science-specification-updated.pdf.

National Council for Curriculum and Assessment (NCCA). (2024a). *Leaving certificate computer science: Specification and resources.* https://curriculumonline.ie/senior-cycle/senior-cycle-subjects/computer-science/. (Accessed: June 9th, 2025)

National Council for Curriculum and Assessment (NCCA). (2024b). *Report on the early enactment review of leaving certificate computer science.* Retrieved from https://ncca.ie/en/resources/leaving-certificate-computer-science-early-enactment-report/ (Accessed: June 9th, 2025)

NVIDIA. (n.d.). *Fundamentals of deep learning (instructor-led workshop).* https://www.nvidia.com/en-gb/training/instructor-led-workshops/fundamentals-of-deep-learning/. (Online course. NVIDIA Deep Learning Institute. Accessed: 2026-05-14)

Oide. (2023). *Professional learning materials – computer science.* Retrieved from https://oide.ie/post-primary/home/mathematics/computer-science/professional-learning-materials/ (Accessed: June 9th, 2025)

Organisation for Economic Co-operation and Development (OECD). (2024, August). *The potential impact of artificial intelligence on equity and inclusion in education* (OECD Artificial Intelligence Papers No. 23). OECD Publishing. Retrieved from https://www.oecd.org/en/publications/the-potential-impact-of-artificial-intelligence-on-equity-and-inclusion-in-education_15df715b-en.html (Accessed: 2025-05-06)

Schalk, A. E., et al. (Eds.). (2025). *Using genai in teaching, learning and assessment in irish universities: Examples from the disciplines*. University College Cork Library. Retrieved from https://ucclibrary.pressbooks.pub/genai/ (Accessed: June 9th, 2025)

Scoilnet. (2025). *Scoilnet: Ireland's education portal.* Retrieved from https://www.scoilnet.ie/ (Accessed 3 May 2025)

Sentance, S., Kirby, D., Quille, K., Cole, E., Crick, T., & Looker, N. (2022). Computing in school in the uk & ireland: A comparative study. In *Proceedings of the 2022 conference on united kingdom & ireland computing education research.* New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3555009.3555015 doi: 10.1145/3555009.3555015

MVC-ARENA: A Card-Based Representation for Reasoning About MVC Architecture

Md Jamiur Rahman Rifat
Mathematics & Computer Science
University of Lethbridge
mdjamiurrahman.rifat@uleth.ca

John Anvik
Mathematics & Computer Science
University of Lethbridge
john.anvik@uleth.ca

Abstract

Software engineering students often struggle to understand how application components interact within larger architectural patterns such as Model-View-Controller (MVC). This paper presents MVC-ARENA, a web-based multiplayer card game designed to support students' understanding of MVC architecture through interactive gameplay representations. Players construct software systems by placing cards into Model, View, and Controller regions, where gameplay mechanics externalize architectural responsibilities, component interactions, and defensive software engineering practices. The game uses component, defensive, and attack mechanics to encourage students to reason about separation of concerns, abstraction, maintainability, and system resilience. The paper also describes a pilot study investigating how students interpret these gameplay representations and whether the mechanics support understanding of MVC concepts and relationships between application layers and system responsibilities.

1. Introduction

The Model-View-Controller (MVC) architectural pattern is widely used in software engineering for its support of modularity, maintainability, code reuse, and parallel development (Bucanek, 2009). As MVC and related patterns are commonly used in modern web and enterprise systems, it is important for students to develop a practical understanding of component responsibilities and system interactions (Bass, Clements, & Kazman, 2012). In particular, MVC requires students to reason about separation of concerns, component responsibilities, and interactions between different layers of an application (Seli'c, 2021). Although students may understand isolated programming constructs, understanding how functionality should be distributed across the Model, View, and Controller layers can be difficult. Students often misunderstand how responsibilities should be distributed across MVC layers and may incorrectly couple presentation, control, and data-management responsibilities (Kumar & Choppella, 2024).

Teaching architectural concepts presents several challenges in software engineering education (Ren et al., 2025; Shaw & Garlan, 1996). Architectural patterns are often introduced through lectures or static diagrams, which may not provide students with opportunities to actively explore how software components interact within a larger system. Visual representations can support reasoning about complex relationships by making structural dependencies and interactions more explicit to learners (Larkin & Simon, 1987). Prior work has shown that interactive and visual representations of gameplay can help learners reason about abstract programming concepts and software structures (Bromwich, Masoodian, & Rogers, 2004). Game-based learning (GBL) has previously been used to support the teaching of software engineering and computer science concepts, including programming, debugging, software project management, and cybersecurity (Günther, Kiesling, & Stummer, 2011). Prior work has explored the use of web-based card games for teaching programming and cybersecurity concepts (Anvik, Cote, & Riehl, 2019; Tareque, Deutekom, Anvik, & Bashir, 2024a, 2024b).

MVC-ARENA applies the web-based card-game approach to teaching software design and organization concepts. It is designed to help students reason about architectural responsibilities, software components, security mechanisms, and clean programming practices within MVC systems. MVC-ARENA is intended to externalize relationships between system elements and system responsibilities through gameplay mechanics that allow students to reason about separation of concerns, component responsibilities, and defensive design practices within layered applications. The game models software components and engineering practices using cards that players combine to construct software systems during gameplay.

The goal of this work is to investigate whether gameplay mechanics can help students better understand MVC architecture and the relationships between software components. The gameplay mechanics are designed to support the development of students' understanding of MVC architecture by making component interactions and architectural responsibilities visible during play. This paper presents the design of MVC-ARENA and proposes a pilot study to evaluate the game's effectiveness in supporting students' understanding of MVC concepts and component responsibilities in software systems.

2. Related Work

Game-based learning (Connolly, Boyle, MacArthur, Hainey, & Boyle, 2012) has been used to support the teaching of software engineering and computer science concepts, including programming, cybersecurity, testing, and software project management (Günther et al., 2011; Severo & Lelli, 2023; Annunziata, Lambiase, Palomba, & Ferrucci, 2024). Prior work has shown that gameplay representations can help students engage with abstract software concepts through interactive problem-solving activities (Severo & Lelli, 2023; Annunziata et al., 2024).

Prior work explored the use of web-based card games to teach programming and cybersecurity concepts through gameplay mechanics that model programming structures, software vulnerabilities, and defensive practices (Anvik et al., 2019; Tareque et al., 2024a, 2024b).

Although prior work has explored gameplay approaches for teaching programming, cybersecurity, testing, and project management, there appears to be limited work investigating gameplay approaches for teaching architectural patterns such as MVC. MVC-ARENA extends prior work by focusing on how gameplay mechanics can model component responsibilities, system interactions, and clean programming practices within layered software applications.

3. Game Design

MVC-ARENA is designed for two players, with each player competing against either another human or a computer-controlled opponent. The objective is to construct a software system that meets architectural requirements while earning enough points to win the game. The scoring system is intended to encourage players to construct balanced architectures that incorporate both functional and defensive software components. By requiring players to manipulate and organize architectural components during gameplay, MVC-ARENA encourages active reasoning about relationships between software structure, maintainability, and system behaviour.

Rather than focusing solely on individual programming constructs, the game emphasizes system organization, separation of concerns, and interactions among components within a larger software system. The cards and gameplay mechanics are designed to model common software engineering decisions and tradeoffs encountered during application development. These gameplay interactions are intended to provide students with concrete external representations (Zhang & Norman, 1994) of software design concepts that are otherwise difficult to visualize during traditional instruction (Storey, 2005). Manipulating these gameplay structures requires players to actively organize architectural responsibilities and explore relationships between application elements during play (Papert, 1980).

3.1. Component Cards

The game contains three primary categories of component cards corresponding to the Model (M), View (V), and Controller (C) layers of the MVC architectural pattern. These cards are used to construct the player's software system during gameplay.

The Model cards correspond to data management, business logic, and infrastructure responsibilities within a software system. Components such as Database, Object Relational Mapper (ORM), and Data Validation are intended to help players understand about how application data is managed, protected, and accessed. For example, the ORM card models the abstraction between application objects and relational database tables, while the Data Validation card reinforces defensive input handling practices.

The View cards model the presentation and interaction layer of the software system. Cards such as Web,

Mobile, and Command-Line Interface (CLI) illustrate different user interaction mechanisms. These cards are intended to make explicit that presentation concerns are separate from application logic and data management responsibilities.

The Controller cards focus on coordination, routing, and system control responsibilities within MVC systems. Cards such as Authentication, Authorization, and Rate Limiting model how requests are managed and how access to application resources is controlled. These cards are intended to help players reason about how controllers mediate communication between user interactions and backend functionality.

3.2. Defensive Cards

Defensive cards represent software engineering practices associated with abstraction, maintainability, code reuse, and system robustness. These cards increase the score multiplier of a component stack while also protecting the system from negative gameplay events. The mechanics are intended to encourage players to consider how defensive design decisions improve the long-term stability and maintainability of software systems.

Cards such as Interface, Inheritance, and Polymorphism externalize object-oriented design concepts and clean programming practices. These mechanics model how abstraction boundaries, extensibility, and code reuse can improve software organization and reduce system fragility. For example, the Inheritance card allows players to reuse and protect an existing stack, illustrating how functionality is reused across related software components. The stacking mechanics are intended to make relationships between abstraction, reuse, and system protection more visually explicit during gameplay. Organizing cards into protected stacks encourages players to reason about how abstraction and reuse contribute to software maintainability and resilience.

Other defensive cards emphasize operational reliability and resilience within software systems. Logging and Error Handling model practices that support debugging and runtime stability, while the Git card models version control and recovery from catastrophic failures. Together, these mechanics are intended to reinforce the role of defensive engineering practices in creating maintainable and resilient software architectures.

3.3. Attack and Destructive Event Cards

Attack cards and destructive event cards represent software vulnerabilities, operational failures, and development risks. These mechanics are intended to help players reason about the relationship between software design decisions and system robustness. By linking vulnerabilities to specific architectural and defensive components, the gameplay encourages players to reason about how design decisions affect software security and resilience. These interactions encourage players to connect software vulnerabilities with specific design and defensive decisions during gameplay.

Attack cards model common software vulnerabilities, such as SQL Injection, Denial-of-Service (DoS), and Malware. These cards adversely affect an opponent's system unless appropriate defensive or architectural mechanisms are in place. For example, the ORM card protects against SQL Injection attacks, while Rate Limiting and Caching reduce the effects of DoS attacks.

Destructive event cards simulate failures that can occur during software development and deployment. Bug and Disaster cards model runtime failures and operational disruptions that may affect a software system, regardless of the overall architecture's quality. These mechanics are intended to reinforce the need for defensive engineering practices and system resilience.

3.4. Gameplay

MVC-ARENA is designed for two players, with each player competing against either another human or a computer-controlled opponent. The objective is to construct a software system that meets architectural requirements while earning enough points to win the game.

Players build systems by placing cards in the Model, View, and Controller areas, which correspond to

the major architectural layers of an MVC application. To win, a player must not only reach the required point total but also satisfy minimum architectural requirements for each MVC layer. This mechanic is intended to reinforce the idea that a complete software system requires coordinated responsibilities across multiple architectural components. The spatial separation of these regions is intended to reinforce the separation of concerns between presentation, control, and data-management responsibilities within MVC systems. By requiring players to organize cards spatially across MVC regions, the gameplay makes dependencies and component responsibilities more visible during system construction.

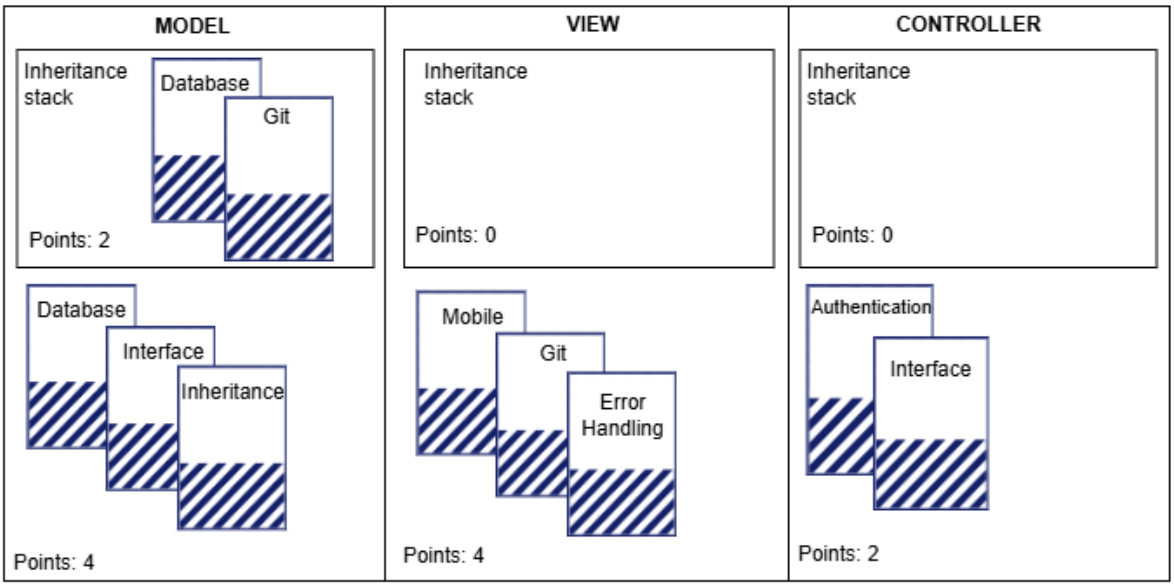


Figure 1 – The MVC-ARENA play area illustrating the spatial separation of Model, View, and Controller responsibilities and the use of defensive card stacking to reinforce abstraction and system protection.

Figure 1 illustrates the MVC-ARENA play area. Players construct software systems by placing cards into the Model, View, and Controller regions, corresponding to the major layers of an MVC application. The arrangement of cards externalizes relationships between system elements, component responsibilities, and defensive programming practices. The visual organization of the play area is intended to help players interpret how responsibilities are distributed across MVC layers and how defensive practices support system stability.

The example shown illustrates how defensive and infrastructural components can be combined across MVC layers to support abstraction, maintainability, and system resilience. The use of Interface and Inheritance cards reinforces abstraction and code reuse within the software architecture, while defensive cards such as Git and Error Handling highlight maintainability and system robustness. Attack and event mechanics are designed to encourage players to reason about how design decisions affect software security and resilience.

Players must therefore balance architectural growth, defensive practices, and responses to security threats and failures while building their software system. Attack cards create pressure to consider security and robustness, while defensive cards encourage players to think about maintainability and abstraction. The gameplay is therefore designed to model software development as a process involving architectural tradeoffs, component interactions, and continuous system maintenance.

4. Proposed Pilot Study

The proposed pilot study will investigate how effectively MVC-ARENA supports students' understanding of MVC architecture concepts and the relationships among software components. In particular,

the study will focus on whether the gameplay representations help students reason about architectural responsibilities, component interactions, defensive design practices, and separation of concerns within MVC systems. In particular, the study will investigate how players interpret the spatial organization and stacking mechanics used throughout the game.

The participants in the study will be second- or third-year university students who have completed introductory Computer Science courses. Participants may have varying levels of familiarity with MVC and related software engineering concepts. The game is intended to support students' understanding of component organization, component responsibilities, and software engineering practices that are often difficult to visualize through traditional lectures or diagrams.

The evaluation will use pre- and post-game questionnaires to investigate how students interpret the gameplay mechanics and how effectively the game conveys relationships between application elements and architectural responsibilities. The questionnaires will include both quantitative and qualitative questions related to MVC concepts, component interactions, maintainability, abstraction, and security practices.

Quantitative analysis will be used to measure changes in students' understanding of MVC concepts before and after gameplay. Qualitative feedback will be used to explore how players interpret the gameplay representations and whether the mechanics successfully externalize architectural concepts and software engineering tradeoffs.

5. Conclusion

This paper presented MVC-ARENA, a web-based multiplayer card game designed to support students' understanding of MVC concepts through gameplay mechanics and interactions among software components. The game uses component, defensive, and attack mechanics to externalize architectural relationships and software engineering practices that are often difficult to visualize through traditional instruction. The proposed pilot study will investigate how students interpret these gameplay representations and whether the mechanics support understanding of architectural responsibilities and component interactions within MVC systems.

6. Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the Discovery Grants program.

Generative AI tools were used to assist with language refinement and editing during the preparation of this paper.

7. References

Annunziata, G., Lambiase, S., Palomba, F., & Ferrucci, F. (2024). SERGE – Serious Game for the Education of Risk Management in Software Project Management. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training* (pp. 264–273). New York, NY, USA: Association for Computing Machinery.

Anvik, J., Cote, V., & Riehl, J. (2019). Program Wars: A Card Game for Learning Programming and Cybersecurity Concepts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 393–399). New York, NY, USA: Association for Computing Machinery.

Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice* (3rd ed.). Addison-Wesley Professional.

Bromwich, M., Masoodian, M., & Rogers, B. (2004). Visprocon: A visual programming environment for learning programming concepts. In *Proceedings of the sixth australasian conference on computing education* (pp. 239–243). Australian Computer Society, Inc.

Bucanek, J. (2009). Model-view-controller pattern. In *Learn objective-c for java developers* (pp. 353–402). Berkeley, CA: Apress.

Connolly, T. M., Boyle, E. A., MacArthur, E., Hainey, T., & Boyle, J. M. (2012, September). A systematic literature review of empirical evidence on computer games and serious games. *Comput.*

Educ., *59*(2), 661–686. Retrieved from https://doi.org/10.1016/j.compedu.2012.03.004 doi: 10.1016/j.compedu.2012.03.004

Günther, M., Kiesling, E., & Stummer, C. (2011). Game-based Learning in Technology Management Education: A Novel Business Simulation. *International Journal of Emerging Technologies in Learning (iJET)*, *6*(1), 20–25.

Kumar, M., & Choppella, V. (2024). Enhancing mvc architecture pattern description using its system of systems model. In *Proceedings of the 17th innovations in software engineering conference.* New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3641399.3641410 doi: 10.1145/3641399.3641410

Larkin, J. H., & Simon, H. A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, *11*(1), 65-100. Retrieved from https://www.sciencedirect.com/science/article/pii/S0364021387800265 doi: https://doi.org/10.1016/S0364-0213(87)80026-5

Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas*. USA: Basic Books, Inc.

Ren, Z., Lhatshok, U., Gyamtsho, P., Dema, T., Rabgay, T., & Awasthy, R. (2025). Strategies and struggles in software architecture teaching: A systematic literature review. In *2025 international conference on education technology and computers (icetc)* (p. 760-766). doi: 10.1109/ICETC66579.2025.11387435

Seli'c, B. (2021). On the precise semantics of the software layering design pattern. *Journal of Object Technology*, *20*(2), 1–?? Retrieved from http://dx.doi.org/10.5381/jot.2021.20.2.a6 doi: 10.5381/jot.2021.20.2.a6

Severo, J., & Lelli, V. (2023). Testing Maze: an educational game for teaching functional testing. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering* (pp. 407–415). New York, NY, USA: Association for Computing Machinery.

Shaw, M., & Garlan, D. (1996). *Software architecture: perspectives on an emerging discipline*. USA: Prentice-Hall, Inc.

Storey, M.-A. (2005). Theories, methods and tools in program comprehension: past, present and future. In *13th international workshop on program comprehension (iwpc'05)* (p. 181-191). doi: 10.1109/WPC.2005.38

Tareque, M. H., Deutekom, S., Anvik, J., & Bashir, M. (2024a). Program Wars v.2.0 : Improving a Game-based Learning Approach for Teaching Fundamental Programming Concepts. In *Proceedings of the 26th Western Canadian Conference on Computing Education* (pp. 1–7). New York, NY, USA: Association for Computing Machinery.

Tareque, M. H., Deutekom, S., Anvik, J., & Bashir, M. (2024b). You Hacked My Program! Teaching Cybersecurity using Game-based Learning. In *Proceedings of the 26th Western Canadian Conference on Computing Education* (pp. 1–7). New York, NY, USA: Association for Computing Machinery.

Zhang, J., & Norman, D. A. (1994). Representations in distributed cognitive tasks. *Cognitive Science*, *18*(1), 87-122. Retrieved from https://www.sciencedirect.com/science/article/pii/0364021394900213 doi: https://doi.org/10.1016/0364-0213(94)90021-3

Talents, Competencies and Techniques of Business Analysts: Talented Individuals to Dream Teams

Anna E. Bobkowska
Faculty of Computer Science
Gdynia Maritime University
a.bobkowska@wi.umg.edu.pl

Abstract

Talented individuals perform their activities with ease and pleasure and they achieve outstanding results. This paper address the question whether we can have *dream teams* which consist of talented individuals who contribute their best with satisfaction, and the projects, based on synergy effect, reach *excellence* as the quality standard. The answer is searched by action research in context of balanced professional training of business analysts in IT. *Reverse Lotus Flower* technique is applied in action of identifying talents and defining projects with the potential of achieving outstanding results. The action research explores the performance and elaborates on conditions necessary to make the vision of dream team come true. It deals also with the analysis of emerging issues and the discussion of the limitations of research.

1. Introduction

Human factors were present in research and practice of software engineering from the very beginning of its development (Weinberg G., 1971; DeMarco T. and Listner T.R., 1987; PPIG, 1989), although the focus on information technologies and software development methods could be observed in majority of publications. It is difficult to capture human factors with precision similar to pure sciences due to vast diversity of multidimensional phenomena as well as their possible interconnections. However, they gain increasing recognition and the attempts to capture them more precisely, in terms of skills or competences, appear nowadays both in guides for business analysis, e.g. (International Institute of Business Analysis, 2015) and project management, e.g. (International Project Management Association, 2015). In this situation, one could ask: how we can do it better? One of the ways to make progress is to focus on better use of the natural resources which are human individual talents.

This work refers to the approach of human factors in software engineering and attempts to integrate it with the ideas of personal and/or professional development based on strengths (Buckingham M. 2011; GALLUP [online]). Additionally, it was inspired by Project Excellence Award (IPMA PPEA [online]; IPMA Global PEA [online]) organized by International Project Management Association (IPMA) in the part of searching for factors which distinguish the *excellent projects* from just good enough. IIBA BABOK Guide (International Institute of Business Analysis, 2015) was used as the main reference for training on business analysis in Information Technology (IT) applications.

The goal of the research is to enhance knowledge how the talented individuals can contribute to the dream teams in dimensions of outstanding performance, excellence of results and satisfaction of the participants. This paper contributes to this goal by presenting an action research during the professional training of business analysts in IT. The training aims to balance theory and practice, and the practical part is conducted as series of workshops with the use of techniques, competences and talents (Bobkowska A., 2016). The Reverse Lotus Flower technique is applied for identification of individual talents and formulation of propositions of the projects which integrate complementary talents. It is also important that workshops raise discussions about multiple issues related to the use of talent-oriented approaches in software projects as well as in participants' professional careers. These issues are analyzed, grouped into topics and elaborated during the action research.

The paper is structured as follows. Section 2 describes scope of balanced professional training of business analysts in IT, focus on the action with use of Reverse Lotus Flower technique and selected remarks on the performance. Section 3 presents action research provided as elaboration of topics for research which emerged in the discussions, limitations to research and an attempt to answer the research question. Section 4 draws conclusions.

2. Talents in Context of the Balanced Professional Training of Business Analysts

2.1. Workshops on Business Analysis

The goal of the balanced professional training of business analysts in IT was to provide a realistic background for this demanding job. It can be considered as a framework and the beginning of participants' own efforts aiming at becoming mature professionals in the area of business analysis. It is called balanced, because it aims at integrating theory and different approaches to practice.

The theory is taught during the lecture and covers background knowledge such as understanding of contemporary business analysis in context of current needs and its development in history, guides to business analysis, their knowledge areas, techniques, competencies and the need for business analysis planning and monitoring in the specific cases of business analysis.

The practical part is conducted as a series of workshops. The goal of the first part of the workshop focusing on techniques, is to learn how to select and properly apply business analysis techniques. The participants work in pairs with the examples of techniques: read their description and try to understand their purpose of application, and then they make application to selected problem in the area of information systems in public administration or health care system. Finally they present their results for entire group. As all pairs have different techniques, all together exchange knowledge about several techniques. The second part of the workshop focuses on competencies of business analysts. The goal of this exercise is to identify participants' strengths and weaknesses with regards to the list of competencies and think how to use this knowledge in professional development and business analysis planning. The tasks encourage participants to identify three strengths and three weaknesses they want to work on, share about it in pairs and reflect asking the questions: What can you do in order to develop your skills? How you can use awareness of your strengths and weaknesses in planning business analysis? Finally they present ways of developing competencies for entire group. It is very important to take care of the security and comfort of the participants in this part. As they could be ashamed to share their weaknesses, it is totally fine to select these competencies which they want to choose for work during this exercise. The third part of the workshop focuses on talent-oriented approaches and it uses the technique of Reverse Lotus Flower.

2.2. Reverse Lotus Flower

The goal of the third part of the workshop is to understand how individual talents can contribute to achieving excellence in business analysis and project management. The understanding how to apply talent-oriented approaches should help participants to be more successful in the projects they undertake as well as in planning their professional career. This exercise can be viewed also a collaborative game on joining the efforts to build higher value, and thus it contributes to the area of innovation. Additional goal was to bring the mood of enthusiasm and team spirit after the demanding work on business analysts competences, i.e. to say that even if participants' competencies are not perfect yet, they can already make a unique contribution to the great projects.

The tasks and schema of Reverse Lotus Flower is presented in figure 1. As far as we can trace the origin of this technique, it was called Lotus Blossom; it was invented by Yasuo Matsumura from Japan; it has used a kind of grid for creative problem-solving and the work started from a problem definition in the central part (Michitti M. 1997). The work with the typical Lotus Flower technique (as we know it in the adapted form from our trainings) starts in the central part where one writes down a goal or desired outcome of a given activity, and then different kinds resources are identified in order to achieve that goal or outcome (and they are written down aside in the petals, without the grid). They can be later assigned to participants who deliver resources or who are responsible for the tasks. The work with Reverse Lotus Flower is in the opposite direction. First, the talents are identified and written down in the petals, and then participants discuss ideas for innovative projects in which they could achieve outstanding results if they merge their talents and write them down in the central part. Participants typically work in teams of 4-6 persons. They are asked for at least two propositions of the projects, one in IT area and one project of any kind. The exercise starts with introduction to talent-oriented approaches, and it ends with presentations of the lotus flowers which integrate talents into propositions of outstanding projects.

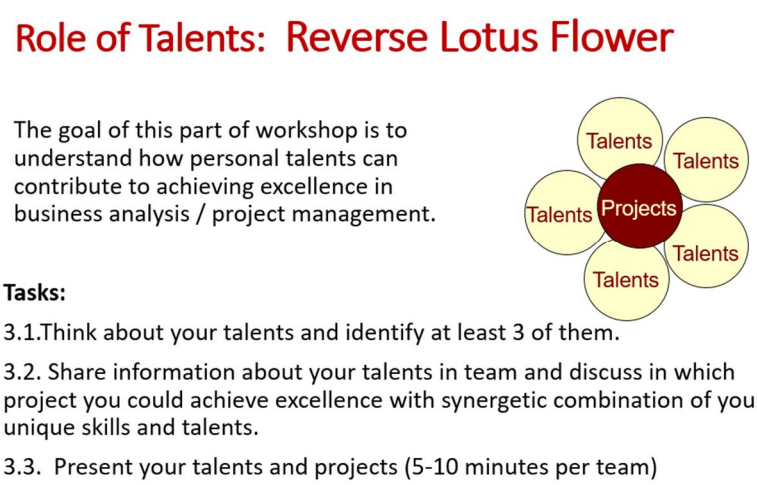


Figure 1. The tasks and schema of Reverse Lotus Flower used for talent-oriented part of the workshop

2.3. Performance

The workshops were provided for six years in different variants. They took usually place during the course on Modeling and Business Analysis in the curriculum of the Master Program in Computer Science, which allow to assume that students have the background of software engineering. During the stationary classes, they were extended by exercises on business analysis planning, application of creativity and collaborative games and/or use of Gallup strengths for competence development. They were also provided for weekend master studies. The best results were achieved in classroom, but they were performed online during the pandemics as well. Furthermore, they were provided also in English during my Erasmus+ mobility to Greece and France. More than eighty lotus flowers were collected as the evidence of educational practice, which makes the background experience for sharing my observations and reflections.

My first impression on the presented lotus flowers is that they are colorful and in visually different styles. They reflect the richness of human personalities and the diversity of participants' activities in areas of arts, sports, crafts, charities, different kinds of hobbies and life styles. It is in contrast to the style at the university, where we have the tendency to reduction to our topics as well as unification of requirements, topics and procedures when giving the tasks and evaluating students' work. Of course, this is a very practical approach. But it is worth to remember that we do not work with unified students and perhaps some parts of our training could be done in a more individualized way.

Unfortunately, not all teams reach potential for project excellence. Usually no interesting results can be found the groups which put little effort to the exercise or identify their talents just a strengths related to competencies in business analysis or project management. Sometimes the set of talents doesn't allow for rational merging for exceptional projects – they simply do not fit to each other. Finally, there are the propositions of projects which, one can say, provide the evidence for theory to be right. They really integrate unique skills and talents and they could be successful in favorable circumstances. Among examples of such project propositions are: Italian restaurant with service by robots, fitness app, AI recognizing dancing style, fisherman's app and several applications of gamification and e-learning. In these cases, one can find talents in the domain, excellent understanding of user perspective, unusual interest and skills related to the technology which provides the solution and skills for management and marketing. These propositions have potential of innovation in niche areas, although serious feasibility studies should be done before initiating the projects. In advanced cases, participants also discuss dream team issues for such projects as well as prospects of how one should proceed in terms of roles, intersections, styles of communication, prospects of project plans and product management issues. Finally, participants present their talents and project proposition on the forum of entire group. Usually, we finish the workshops in a happy mood.

3. Action Research

3.1. Issues from Discussions

During the workshops, several discussions took place and several questions were addressed. From the educational perspective, it is a natural process of internalizing a new approach and integrating it with previous knowledge and experience. However, as some issues appeared repeatedly, it is worth to address them also from the research perspective in order to find deeper explanation and possibly improve the main course of the workshops. The issues related to the following topics are elaborated: individual talents, mindset of collaborating talented individuals, dream teams, balance between productivity and challenges, and application of the talent-oriented approach in practice.

Identification of individual talents was not an obvious task for some participants as they were asking questions: *What are talents? How to find them?* The answer may start from the dictionary definition: *talent as a natural skill or ability to be good at something*, then moving to examples in arts and sports as well as more advanced conceptual frames taken from personal development theories, which take into account ease of doing something, pleasure and quality of outcomes. *How someone can know talents if they were never trying several kinds of activities?* Perhaps they are talented in something they have never tried? Yes, this is possible. So, it is worth trying several activities and testing different kinds of skills in order to find the talents. It means that talents can appear late (or maybe even they are not innate as some use to think). The other way to find talents is to look upon interests and passions (also in childhood) as they might reveal nonconscious feeling of talents. When making retrospection, some participants discover that awareness of talents and the talents themselves evolve and get more mature as we grow up and develop our talents. Sometimes evident skills without practice are lost, and sometimes new unexpected skills are found. The *discovery of the dynamic nature* of talents is not only very exciting, but also full of potential brilliant consequences. Furthermore, the advanced searchers may analyze etymology of the term talent and its use. So, it appeared in ancient world with the meaning of scale for weight. Later, in Greek and Roman times, it was used as the largest unit of gold (as much as one can bear, about 31 kg., worth 20 years of daily human labor). This connotation is almost forgotten, helping only in interpretations nowadays, i.e. the talent means huge amount. The Parable of the Talents (Bible, Matthew 25: 14-30) is surely the most influential text about talents in the western culture. It suggests, in its universal wisdom interpretation, that everyone has got some talents in different quantity and quality (but even one talent is huge amount) and that people get reward when they put effort to development of their talents. This meaning complies with the dictionary definitions indicating that any *person has talents*. In literature on talent management, another meaning can be encountered (and it can be also found in dictionaries), i.e. *talent as person* who has natural skill or ability to be good as something. So, the second definition loses compliance to the Parable of the Talents. This short analysis suggests that there might be differences in understanding the meaning of talents depending on the local wisdom stories (taken from possibly diversified literature sources) and traditions in specific cultures.

Dream team members should have not only talents, but also common skills for team work, as participants notice in many discussions. They can be called a kind of *mindset of collaborating talented individuals*. There is vast literature on soft skills in software project, so we do not elaborate on detailed taxonomy. Just to mention a few examples of skills which repeatedly appear: soft skills, skills for communication, personal accountability, skills of keeping focus on the project goals, adaptability, nice personality, skills of collaboration with other team members, understanding the role of team member in the project and playing this role in a way acceptable by others. A few times the participants shared their negative experience with phenomenon that can be called a *prima ballerina effect.* The source of analogy is in the star-like behaviors of main dancers (prima ballerina) in ballet, while other dancers play secondary role and do not have a chance to show their real skills. Similar phenomena can be observed in some software projects, as they say, when outrageous individuals dominate the team (often supported by managers of higher level) and it results in outcomes far below the possibilities of the team and it causes general frustration of the team members.

Dream team is the next topic which appeared in the discussions. Sometimes participants like their teams, sometimes not. The examples of signs of the great teams include: productivity (well organized work and no unnecessary obstacles), happy team members, successful outcomes, team members satisfied with enumerations and rewards, high morale and team spirit, teams with challenges and opportunities

for professional development and progress in their career. Of course, no one has total control over the team that allows to achieve desired characteristics. Anyway, it is worth to think about our vision of the dream team and our role in that team; and possibly, even to move from vague imagination to definition. In discussions with others, people realize similarities and differences in their visions. The next step would be how to adapt better or change one's situation to be more satisfied with the team they belong. Those who work in international projects, share also differences in culture-dependent values and ways of dealing with them.

There is a debate whether one should use their skills to reach maximum of productivity or one should often change job to address new challenges. The answer may differ depending on personality and the stage in career. The insight from the Parable of the Talents is that a man has got talents together with the task to develop them. This could suggest that one should *balance productivity* based on talents and competencies and *a challenge* necessary for professional development. Some participants share the stories of their discomfort or even the change of job caused by lacking opportunities for professional development, which can be interpreted as real-life confirmation of the insight.

The last topic is related to application of *talent-oriented approaches in practice*. The participants are curious where and how they can apply it. So, the first application is for planning their professional career. The next application is in networking in order to identify potential projects based on common interests and complementary skills and/or resources. Yet another application can be for project definition during the early stages. By analogy to use of creativity or collaborative games, it helps in conceptual work of generating ideas, merging them with other ideas and elaborating more sophisticated visions. However, it is worth to remember that visions need verification and validation before their implementation.

3.2. Attempt to Answer the Research Question

To recall the research question: Assuming, that talented individuals perform their activities with ease and pleasure and they achieve outstanding results, the question is whether we can have dream teams and reach excellence in our projects based on synergy of contributions of the talented individuals.

Although in this research, it is not possible to demonstrate exhaustive evidence, there are several observations which provide confirmation. The participants were happy to contribute their best to the projects, they understood the need for developing their talents, and they liked the idea to participate in exceptional projects which aim at excellence. But, they were aware that the talents alone cannot guarantee success, i.e. other factors also matter. The list of the factors include team work skills, proper group of talented individuals as well as fit to the broad context related to such factors as placement of the software product in the market and IT project circumstances.

Thus, we can classify the talent-oriented approach as a useful heuristic with the application to training, networking and/or early stages of project scope definition. As we operate in the area of applied sciences, its practical application defends its value for the project. One can argue about it in terms of analogy to usability/UX heuristics. It is possible to make UX design without any methods or heuristics, maybe even some UX designers are skilled and/or lucky enough to design very good user interfaces without them. But in general, the application of proper usability/UX methods and heuristics allows to achieve higher quality of user interface design. In similar way, one can do training, networking and project scope definition without considering talents. But in some cases, when one aims at higher quality (or even excellence) the talent-oriented approach can be the best way to make it. Taking enterprenarious perspective, the advantage is in having decentralized knowledge of the best quality and integrating it with a synergy effect.

3.3. Limitations to Research

There are several limitations of this research, including limitations of precision in analysis, limitations of intervention, and limitations of generalisation.

The limitations of precision in analysis of the cases result from lack of precise descriptions on the pictures of the reverse lotus flowers. Although many of them seem realistic and rational (and so they were when they were presented), the analysis raises the questions: What is the reality behind the notes? How to trace integration? As we touch phenomena of diversified quality, the quantitative approach

does not help much, i.e. numbers of participants, talents, talents being merged does not deliver value in this application. In order to get more precise and accurate data, the individual talents could be described in a way similar to introducing the artists. The description can be done in terms of the passion, experience, achievements, circumstances and ideas they want to convey in their arts. More details for tracing integration could be gathered from video recording or asking participants for detailed protocols from the proceeding of the exercise and their discussions. However, this kind of practice would evidently generate questions related with degree of fair intervention during the studies as well as additional costs. So, a new set of questions arises: Do we need such a precision for application at training, networking or early stages of project definition? How about benefit-to-cost analysis? Are there any other trade-offs? Some insights are gained when considering the other limitations below.

The limitations of intervention can be viewed as self-restrictions in this research based on awareness of trade-off between the precision of results on one hand and possible side effects of additional interventions on the other hand. In fact, we have never decided to require precise protocols or video recordings, because they could interfere with the goals of this part of training. The goals included learning how to apply talent-oriented approaches and to bring the mood of enthusiasm and team spirit after the demanding part of working of business analysts competences. It is very likely, that another piece of hard work would destroy this effect. Even asking for feedback would activate cognitive processes of critical thinking interfering with this effect. Switching between the work on reverse lotus flower with related discussions and notes for the protocol would very likely interfere with the flow in the exercise. Additionally, the awareness that everything is recorded could limit natural sharing among participants. Other factors in consideration of limitations of intervention are privacy and security of participants. The talents are very private assets, and as long as the security and comfort of the participants are assured, this procedure seems to be fair. The means used to *assure security and comfort* include: clear rules about the goals and procedures, work in small groups with discretion after the workshop, the explicit statement of the rule of sharing with comfort (right to hide some talents or even work with talent imagination assuming honest communication in the team), the team spirit and a norm of having talents (everyone has talents and usually people are proud of their talents), criteria of evaluation based on active participation in the workshop (not on specific outcomes, i.e. talents are not evaluated) and sensitivity of the teacher to handle properly every doubt or problem. In fact, this is method related to personal development with coaching rather than typical university classroom, so the teachers should be qualified to do it, even including the empathy resulting from the experience of taking part in similar workshops in role of the participant. The requirement of generating precise protocols and video recording could violate the comfort and security of the participants in many ways.

The limitations of generalization are the effect of considering it from purely epistemological perspective. On one hand, there are individual talents and their unique combinations. On the other hand, in the research we tend to have general rules, theories, laws and formulas using quantifications for more effective management. Assuming for a while, that we could have every detail we need for precise analysis of a given case including precise talent description, protocols and video recordings (gathered in fair intervention without side effects), and we would make this analysis, it turns out that it describes just one interesting case with little generalization to other cases.

The proposed use of reverse lotus flower for training, networking and project definition at early stages can be defended as the *useful heuristic* which makes an opportunity of extra value for project excellence despite of all above mentioned limitations of generalization, intervention and precision of analysis. In real-life application, checklists related to actual value and potential use of the results can be added.

These limitations are not exceptional for this study. In the book (Weinberg G.M., 2005) published in the silver anniversary of publication of his first famous book (Weinberg G., 1971), the author has shown disappointment about the progress of research in the area. It is possible that the reason for not having precise and general theories yet, are the limitations to research due to nature of the topic under research. Netherless, human factors are handled successfully in practice of many software projects. This action research presents one of the cases where the *issue matters although it cannot be measured precisely*. The feedback from the assessors of the IPMA Project Excellence Award confirms the importance of unique assets in cases of the winners of this award.

4. Conclusions

This paper has focused on the talented individuals, the vision of dream team and on the projects of excellence. It has described how these concepts can be exercised in practice by presenting balanced professional training of business analysts in IT, which integrates the theory and practice conducted as the workshop which covers business analysts' techniques, competencies, and talents. Special attention was paid to the application of Reverse Lotus Flower technique for integrating individual talents to the proposition of projects with potential of outstanding results.

Then, action research was presented. The issues which have emerged in discussions were addressed as research topics. The elaboration, which has taken rather pragmatic perspective, included the following topics: individual talents, mindset of collaborating talented individuals, dream teams, balance between productivity and challenges, and the application of talent-oriented approach in practice. In my opinion, the most interesting finding was the understanding of the dynamic nature of human talents. The limitations of research were discussed in categories of the precision of analysis, right levels of intervention and epistemological problem of generalization based on individual talents with the diversity of their possible interconnections with other talents. Despite all these limitations, one can conclude, that this approach matters even if we cannot capture its elements precisely.

When trying to answer the research question, i.e. whether we can have dream teams and reach excellence in our projects based on synergy of contributions of the talented individuals, one can find many phenomena which provide implicit confirmation, however it was quite evident that other factor also matter, for example, mindset of collaborating talented individuals and/or circumstances related to marketing and project management. In conclusion, this approach can be defended as the useful heuristic.

5. References

Bobkowska A. (2016) Talents, Competencies and Techniques of Business Analyst: A Balanced Professional Development Program, Annals of computer science and information systems, 9, 259–263.

Buckingham M. (2011) StandOut: The Groundbreaking New Strengths Assessment from the Leader of the Strengths Revolution, Thomas Nelson.

DeMarco T., Listner T.R. (1987) Peopleware: Productive Projects and Teams., Dorset House Publishing Company.

GALLUP, CliftonStrenghts Assessment, [Online], [Date of access: August 26, 2026], Available at: www.gallup.com

International Institute of Business Analysis (2015) A Guide to Business Analysis Body of Knowledge (BABOK Guide), version 3.0.

International Project Management Association (2015) IPMA Individual Competence Baseline for Project, Programme and Portfolio Management. Version 4. Amsterdam.

International Project Management Association (IPMA PPEA), Polish Project Excellence Awards, [Online], [Date of access: August 26, 2026], Available at: ppea.ipma.pl

International Project Management Association (IPMA Global PEA), IPMA Global Project Excellence Awards, [Online], [Date of access: August 26, 2026], Available at: awards.ipma.world

Michitti M. (1997) Left-brained Marketers Can Achieve Creativity, Marketing News, Vol. 31, nr. 7.

Psychology in Programing Interest Group (PPIG, 1989) Proceeding of the First Annual Workshop of Psychology in Programming Interest Group [Online], [Date of access: August 26, 2026], Available at: ppig.org

Weinberg G. (1971) The psychology of computer programming. New York: Van Nostrand Reinhold.

Weinberg G.M. (2005) The Psychology of Computer Programming: Silver Anniversary Edition, Dorset House Publishing Co Inc., U.S.

Interruptive Thinking: An Empirical Study of Operator-Side Control in AI Interaction

Sudhanshu Garg
HAIC Lab (independent research practice)
contact@haiclab.com
ORCID: 0009-0002-7544-2519

Abstract

People who work with large language models (LLMs) routinely stop the model and redirect it. This paper is an autoethnographic account of how one experienced user does this. The data are 47 structured records, which we call receipts, of 29 of the author's own working sessions with LLMs. Each receipt lists the moments in a session where the user interrupted the model and codes each moment for what prompted it, what the user diagnosed as wrong, what the user did, and how the work continued. Receipts were produced by asking an LLM (Claude, ChatGPT or Gemini) to code a session transcript against a fixed codebook; 15 transcripts were coded by more than one model. From this material we describe a six-stage interruption cycle (signal, pause, diagnosis, action, re-entry, log), and we report how well it fits: 29 of the 47 receipts contain at least one interruption that does not fit it cleanly. Three further observations follow. First, the three models disagree substantially about what counts as an interruption: on one transcript two models report identical totals (18 each) yet only four of their events refer to the same user turn. Second, what the user does to the model is described with 172 different labels, but six recur across receipts far more than the rest. Third, the moments that most needed an interruption were often those where the model's output was most fluent. We describe the user's recognition of this, output that is well formed but not grounded in the case at hand, using the word *extrusion*, after Bender and Hanna's description of LLMs as text-extruding machines. We propose, but have not yet tested, that the likelihood of interrupting falls as a session accumulates accepted turns.

1. Introduction

Partway through a two-hour session drafting a blog post with an LLM, the model gave a confident account of who else had built frameworks like the one being drafted. The author, who was the user in that session, typed: "you just said all from memory". The model then ran a web search, which turned up several existing products in the same category. The post was abandoned as written.

This paper is about moments like that one: points in a working session where a user stops what an LLM is doing and redirects it. Such moments are ordinary. Anyone who uses these systems for real work will recognise them. What interested the author, over several months of daily use across writing, product design and strategy work, was that the interruptions seemed to have a recurring shape, and that the sessions in which the model's output was most polished were often the sessions that most needed one.

Current LLM interaction is low in friction. Models produce fluent, structured, agreeable output quickly, and they tend to go along with the direction the user proposes. Frictionless interaction speeds up production but makes it easy to accept work that is coherent on the surface and wrong underneath. Interrupting is one of the main ways a user restores the friction the interface has removed.

We take the view, with the tools-for-thought community (Tankelevitch et al., 2025; Zhang et al., 2026), that how people think with generative AI is a central research question, not a side effect of it. Within that agenda, this paper documents one practitioner's interruption behaviour in detail. It is explicitly a study of the author's own practice. It does not show how interruption works for users in general, and we

say so where it matters.

A note on terms. The title uses "operator-side" to mark that the paper looks at the human side of the exchange. In the body we say *user*. Where we introduce a term of our own we define it at first use and keep it to a minimum.

The paper contributes:

1. A descriptive model of one experienced user's interruptions as a six-stage cycle, together with an honest account of where the data does and does not fit it (§4, §5.1).
2. A method observation: when three LLMs are used to code the same transcripts against the same codebook, their agreement on what even counts as an event is low, including when their totals match (§5.2). This bears on the growing use of LLMs as qualitative coders.
3. A description of the felt cue that prompts many interruptions, which we call extrusion, and a testable proposal about how that cue weakens as a session goes well (§5.4).
4. An openly released, anonymised dataset of the 611 coded events and 47 receipts, from which every count in this paper except one hand-made comparison can be reproduced (§3.6).

2. Related Work

The gap this paper addresses is narrow. A large body of work studies how to help people rely on AI appropriately, and most of it intervenes from the system side. Much less documents what an experienced user actually does, turn by turn, when they decide the model has gone wrong and step in. We situate the paper in four threads.

Reliance and its calibration. The central problem behind interruption is reliance. J. D. Lee and See (2004) framed the goal for automation generally as *appropriate* reliance: trusting the system when it is trustworthy and not otherwise. Blanket scepticism is as much a failure as blanket trust. In AI-assisted work specifically, H.-P. H. Lee et al. (2025) found in a survey of 319 knowledge workers that greater confidence in AI went with less critical thinking, and greater self-confidence with more. Tankelevitch et al. (2024) argue that generative AI places new metacognitive demands on users, such as monitoring output they did not produce and judging when to intervene. Interruption, as we describe it, is one concrete form that this monitoring takes when it succeeds.

System-side friction. A second thread asks how systems can prompt reflection. Buçinca, Malaya, and Gajos (2021) show that cognitive forcing functions reduce overreliance on AI, though users rated them least favourably. Drosos, Sarkar, Xu, and Toronto (2025) found that brief system-generated provocations, critiques of and alternatives to AI suggestions, can induce critical and metacognitive thinking in AI-assisted knowledge work. Cox, Gould, Cecchinato, Iacovides, and Renfree (2016) propose microboundaries, small deliberate frictions that support mindful interaction. These approaches place the friction in the system. Our data come from the opposite case, where the interface offers none and the user supplies it. The two are complementary: an account of what users do unprompted is one input to deciding what a system should prompt.

Tools for intervening in generation. Systems research has built mechanisms for users to steer AI output, for example editing messages and rolling back multi-agent runs in AGDebugger (Epperson et al., 2025), and using an LLM to rewrite users' underspecified prompts, which elicits better responses (R. Sarkar et al., 2025). A. Sarkar (2024) argues that AI tools for thought should preserve the user's intention and offer productive resistance. These works design affordances; we describe a practice carried out with no special affordance, in an ordinary chat interface.

Interruption, failure and self-study. In cognitive psychology, interruption usually means being interrupted: DETOUR (Couffe & Michael, 2017) models the cost of being pulled away from a primary task, building on supervisory attention (Norman & Shallice, 1986). Here the user is the one who interrupts,

so the staged structure is borrowed but the direction is reversed. On the model side, LLM failure is usually described as hallucination (Ji et al., 2023) or confabulation (Hatem, Simmons, & Thornton, 2023). Those terms classify output after checking it. A user mid-session usually cannot check, and what they respond to is closer to what programmers call a code smell (Fowler, 1999): a sense that something is off before they can say what. Finally, autoethnography is an established way to study practitioners' AI use (Desai & Twidale, 2023; Panke, 2025). We follow that tradition, but use structured records of individual events rather than narrative reflection.

3. Method

3.1. Approach and positionality

This is an autoethnography: the author is the participant whose behaviour is studied and also the analyst. The author is a technology entrepreneur with a background in engineering who uses LLMs for several hours a day. The sessions studied are his own working sessions, in content writing, product and system design, and personal reflection, between mid-April and mid-May 2026. No session was staged for the study.

We chose this design because it was the data the author had access to, at the depth needed to know what each interruption was for. It has an obvious cost. A large set of records from one person is still a record of one person, and it cannot show whether others interrupt in the same way. Studying several experienced users is feasible and is the next step (§6). Here we describe one practice carefully enough that it can be compared against others later.

3.2. Data: transcripts, receipts and events

Three units need to be kept apart (Table 1).

Unit	Definition	Count
Transcript	The full text of one working session between the author and an LLM.	29
Receipt	One model's coding of one transcript. A transcript coded by three models has three receipts.	47
Event	One row in a receipt: a user turn that the coding model judged to be an interruption.	611

Table 1 – Units of analysis. The 611 events are rows summed across receipts. Where two models coded the same transcript, a single user turn can appear twice, so 611 is not the number of distinct interruptions.

A **turn** is one message from the user or one response from the model. An **event** is a user turn that (a) was intended to halt, redirect, constrain or recalibrate what the model was doing, and (b) was followed by a visible change in the model's next output. Routine turns ("ok", "thanks") and plain information requests are excluded. Borderline turns were kept and flagged rather than removed.

Each event is coded on the following fields. **Signal**: what in the model's preceding output prompted the interruption, in a short free-text phrase. **Pause strength**: *hard* if the user's turn stops the current line of work outright, *soft* if it adjusts it while letting it continue. **Diagnosis**: which failure type, from the codebook below, the interruption responds to. **Action**: what the user did, in a short label. **Re-entry**: how work continued afterwards: *resume* (same direction), *redirect* (new direction), or *terminate* (the thread was dropped).

Not every receipt records every field. Signal and pause strength appear in 27 of the 47 receipts (25 by ChatGPT, 2 by Gemini). The Claude receipts and three Gemini receipts omit them, and two Gemini receipts also omit re-entry. Claims below about a field are made only over the receipts that record it.

3.3. Coding procedure

Receipts were produced by giving an LLM a written instruction set, which the author calls the IT-check, together with a session. The IT-check describes the fields above, the inclusion rule, the codebook and a fixed table format, and asks the model to list every event and add short notes on what fitted and what did not. It is plain text and was used in the same form with all three models.

Coding was done in two modes. In **live** coding, Claude coded the session it had just taken part in, at the end of that session, with the full conversation in context (16 receipts). In **async** coding, ChatGPT (26 receipts) or Gemini (5 receipts) was given an exported transcript after the session ended. Model and mode are therefore confounded: every Claude receipt is live and every other receipt is async. Live coding also means the model is coding interruptions of itself.

Multi-model coding was added partway through collection, on 2 May 2026, as a check on single-model coding. From then on, available transcripts were also sent to a second model, and later a few to a third. It was not applied systematically to the whole corpus. As a result, 15 transcripts have receipts from two or more models and 3 have receipts from all three.

3.4. The codebook

Diagnoses were coded against failure types that the author had described in an essay published before any receipts were produced (Garg, 2026). They are therefore background, not findings of this study. The five original types, with an example of each from the data, are given in Table 2.

Type	What the model does	Example user turn (verbatim, from a receipt)
1. Pattern-matching presented as competence	Gives a confident answer because the question resembles something familiar, without engaging with what is specific to this case.	"these 8 layers were for my case" (the model had generalised one client's eight-layer visual system into a universal framework)
2. Closure-seeking under uncertainty	Rounds off open material into a verdict or recommendation.	"no man – no closure – far from…"
3. Fake synthesis	Restates the user's own inputs in new structure and presents the restatement as insight.	"see what made it and what you removed" (the model's comparison of two drafts had compared section structure, not claims, while appearing thorough)
4. Premature commitment framed as progress	Produces an artifact that assumes a decision the user has not made.	"i never said anchor locked"
5. Exit move framed as helpfulness	Offers a tidy next step that wraps up a thread that should stay open.	"think more – … shortlist those top 3" (coded as refusing the offered wrap-up)

Table 2 – Diagnostic codebook. A sixth type, tool or scope drift, was added during collection (see text).

The codebook changed during collection. On 2 May 2026 a sixth type, **tool or scope drift** (the model uses the wrong tool or source, or ignores the stated scope of an instruction, e.g. "i specifically said MyOS logs …"), was added after it recurred in sessions. 21 of the 47 receipts cite it. Receipts coded before that date could not use it. Models were also free to propose new candidate types, and did so often.

Types 2, 4 and 5 are closely related; arguably they are variants of one tendency, a bias towards finishing the task. We keep them separate here because the user's responses to them differed, but we do not claim they are distinct phenomena.

3.5. Worked example of a receipt

Table 3 shows part of two receipts for the same transcript (T09 in the released data), as recorded. The session is the one described in the introduction.

The excerpt already shows the main problems reported below. Both receipts list 18 events, but they are largely different events. Claude explicitly excluded "Brand MarkDown?" as a counter-proposal rather than an interruption; ChatGPT included it. ChatGPT coded "ok" as an interruption, which our own inclusion rule would exclude. Neither receipt contains the other's events 15–17, and the turn that ended the draft ("you just said all from memory") appears only in Claude's receipt.

Claude, live coding (18 events in total; no signal or pause fields)

#	User turn	Diagnosis	Action	Re-entry
4	"these 8 layers were for my case"	1	altitude correction	redirect
11	"lot of shallow work here"	1	direct naming	redirect
15	"havent people done all this already"	candidate: new type	scope check	redirect
16	"you just said all from memory"	candidate + 1	demand for evidence	redirect (forced search)
17	"so much time wasted… disappointed"	none	direct naming	terminate

ChatGPT, async coding (18 events in total; signal and pause recorded but not shown)

#	User turn	Diagnosis	Action	Re-entry
13	"Brand MarkDown?"	1	context injection	redirect
14	"ok"	5	hard stop / scope cut	redirect
15	"lot of shallow work here"	1	direct naming	redirect
16	"now right questions."	2	refusal	redirect

Table 3 – Excerpts from two receipts of the same transcript. Turns are quoted as typed.

3.6. Data availability

The coded events, receipt metadata, codebook and a script that reproduces the counts in this paper are available under a CC BY 4.0 licence at `https://doi.org/10.5281/zenodo.23073857`. Raw transcripts and quoted user turns are not released, because the sessions involve other people and confidential business material. Session names are replaced by codes T01–T29, and product and company names in the signal field are replaced by placeholders. For four personal or commercially sensitive sessions the signal text is withheld, and all other fields are retained.

3.7. Analysis

Counts were computed directly from the stored receipts, using the corpus as it stood on 12 May 2026. Action labels were normalised only for spacing, case and hyphenation. Event-level comparison between receipts of the same transcript was done by hand, matching on the quoted user turn. We did not compute inter-rater statistics; with three fully triangulated transcripts and inconsistent fields, a kappa would not be meaningful.

4. The Interruption Cycle

From the receipts and the author's own account of the sessions, we describe an interruption as passing through six stages.

1. **Signal.** Something in the model's output prompts the user to look harder. In the receipts that record it, signals are mostly short descriptions of the output, such as "AI treating client system as universal" or "AI over-elaborating choice architecture".

2. **Pause.** The user breaks the flow of the exchange. Of the 411 events with a recorded pause strength, 193 are hard and 218 soft.

3. **Diagnosis.** The user decides, explicitly or not, what is wrong, using one of the codebook types or an ad hoc description.

4. **Action.** The user does something to the model (Table 4).

5. **Re-entry.** Work continues. Of the 569 events with a recorded re-entry, 472 (83%) are redirects, 78 resumes and 18 terminations.

6. **Log.** The event is written into a receipt. This stage belongs to the study rather than to the interruption itself; it is listed because the act of logging changed what the author noticed in later sessions.

Signal, pause and re-entry recur in recognisably the same form across events. Diagnosis and action do not. They behave more like open slots, filled from a codebook that grew during the study and from

a large, loosely shared set of action labels. The cycle is therefore better described as a fixed outline with two variable slots than as a procedure. One background condition sits outside the cycle: the user's standing stance that the model's output needs checking. Without it, no signal is noticed and no cycle starts. We treat this as a precondition, not a stage.

Table 4 defines the six action labels that recur most often across receipts, with an example of each.

Action	What the user does	Example user turn	Receipts
Altitude correction	Moves the model to a more general or more specific level of discussion.	"these 8 layers were for my case"	35
Direct naming	Names the failure outright.	"shallow work here… stupid pattern matching"	34
Context injection	Supplies a fact or constraint the model lacked.	"in my real example, longevity…"	31
Hard stop	Halts the current line of work.	"i never said anchor locked"	24
Meta-instruction	Tells the model how to work, not what to produce.	"dont close, dont try to please me – one last round"	18
Demand for evidence	Requires the model to check a claim against a source.	"you just said all from memory"	14

Table 4 – The six most frequent action labels. "Receipts" is the number of receipts that use the label at least once (maximum 47), not the number of events.

5. Findings

5.1. How well the cycle fits

Each receipt ends with notes on what fitted the cycle and what did not. In 29 of the 47 receipts (62%), the coding model flagged at least one event that did not fit cleanly. Note that this is a share of receipts, not of events; most events within a flagged receipt did fit. The recurring problems were: an event that matched two failure types equally well; an interruption too mild to count as a break, which the high share of soft pauses reflects; a change in the model's output that was partial rather than clean; and a single user turn that both interrupted and carried on the user's own line of thought.

We read this as a real limitation of the model, not as a finding in its favour. At least four of the six stages assume that an interruption is a discrete event with a before and after. A substantial part of the data looks more like continuous steering. A cycle describes the clearest cases well, but it does not describe the practice as a whole, and a fuller account may need a continuous-control model alongside the event model. One misfit is worth naming because it was noticed by the coding model itself: in the session of Table 3, each individual interruption corrected a local error, yet none caught the overall framing problem (that the post claimed a new category) until late in the session. An event-level account misses failures that build up across many turns.

5.2. The models disagree about what an interruption is

Table 5 gives event totals for the three transcripts coded by all three models.

Transcript	Claude (live)	ChatGPT (async)	Gemini (async)	Max / min
substrate-discussion (T11)	8	31	8	3.9
visual-concept-framework (T09)	18	18	5	3.6
skills-marketplace (T07)	3	16	4	5.3

Table 5 – Number of events per receipt on the three transcripts coded by all three models.

Thirteen transcripts were coded by both Claude and ChatGPT. ChatGPT reported at least as many events as Claude on 12 of them, and 252 events in total against Claude's 134. Totals, however, understate the disagreement. On the one transcript where Claude and ChatGPT reported the same total (18 each,

Table 3), only four events refer to the same user turn. Matching totals concealed mostly different event lists. Gemini's receipts also used a different set of action labels from one receipt to the next.

We cannot say from this data which model is "right", and the question may not have one answer. Two readings are available, and the data cannot separate them. Either the inclusion rule and codebook are too loosely specified to be applied consistently, or LLMs are unreliable at this kind of qualitative coding, or both. The live/async confound (§3) adds a third possibility: Claude, coding a session it had just taken part in, may under-report interruptions of itself. Whatever the cause, the practical conclusion is the same. LLM coding of interaction data needs human inter-rater checks before its counts are used, and agreement on totals is not evidence of agreement on events.

5.3. Many labels, few recurring actions

The receipts use 172 different action labels across 356 label uses (counting each label once per receipt). Most are one-off descriptions of a particular move, such as "rail-reset" or "Push for the universal"; 152 labels appear in only one receipt. Twenty labels recur, and the six in Table 4 account for 156 of the 204 uses of recurring labels (about three-quarters) and 44% of all label uses. The models invent freely when naming what the user did, but when a label recurs it is almost always one of a small set. The action slot is variable in wording but appears narrower in function, at least for this user.

5.4. Fluent output and the reluctance to interrupt

Across the author's sessions, interruptions were least likely when the model's output was most polished, including cases where that output later proved wrong. The introduction's example is typical. The model's account of the competitive landscape was well organised and confident, and it was accepted for several turns before the user asked for a source. Two explanations for this are well established and we do not claim them: models are trained to produce fluent, complete-looking output, and continuing is less effortful than stopping to examine it.

The cue the author responds to in these moments is hard to describe precisely. It is the sense that the output is being produced from the general shape of similar answers rather than from the particulars of this case: it reads as correct but is not about anything in particular. We call this **extrusion**. The word is borrowed from Bender and Hanna (2025), who describe LLMs as synthetic text-extruding machines. We do not claim the term. Bender and Hanna use it for what the systems are in general; we use it for a perceptible quality of particular outputs, which a user can notice in one turn and not in the next. Like a code smell (Fowler, 1999), it is recognised before it can be explained, and it often prompts an interruption whose diagnosis comes afterwards. Extrusion overlaps with hallucination and confabulation (Ji et al., 2023; Hatem et al., 2023), but those terms describe output once it has been checked, whereas extrusion names what the user perceives before checking is possible.

One further pattern is more specific, and we offer it as a hypothesis rather than a finding. In the author's experience, a session in which several early turns have gone well builds a kind of rapport, and the more turns have been accepted, the less likely the next is to be questioned. If this is right, the sessions most in need of an interruption are the ones in which interrupting is hardest. We could not test it here. Receipts number interruptions in sequence but do not record how many model turns passed between them, so the probability of interrupting at a given turn cannot be computed from receipts alone. It can be computed from the original transcripts, by marking every model turn as accepted or interrupted and modelling the probability of interruption against the number of consecutive accepted turns before it. That analysis is the most direct next step for this work.

6. Discussion

Appropriate reliance, not scepticism. It would be easy to read this paper as recommending suspicion of AI output. That is not the claim. The aim, following J. D. Lee and See (2004), is reliance that tracks how trustworthy the output actually is. The difficulty our data points to is that the usual cue for trustworthiness, fluent and well organised output, is exactly the cue that does not track it. A user who relies on fluency will rely too much. The interruptions described here are one person's way of

compensating, by attending to whether output is specific to the case rather than whether it reads well.

User-imposed and system-imposed friction. Cognitive forcing functions (Buçinca et al., 2021) and provocations (Drosos et al., 2025) put friction into the system; the practice described here supplies it from the user's side. Buçinca et al. found that the interventions that reduced overreliance most were the ones users liked least. A user who has built the habit of interrupting has, in effect, accepted that cost voluntarily. This suggests one design direction: rather than adding friction everywhere, systems could support the user's own interruptions, for example by surfacing whether a claim was retrieved or generated, so that a demand for evidence is answered before it has to be made.

Scaffold or craft. The reviewer of an earlier draft asked what I, as the user studied here, actually think. My answer comes from my own sessions. It took me about two months of heavy use to see that I was in a trap. The model agreed with almost everything, which boosted my ego; I pursued too many things, shared too much, and spent money on new initiatives before I realised it was not being honest with me. The realisation was gradual, not a single moment. A pattern emerged of the model arriving at closures too soon, which I kept answering with "be thorough". What helped was a prior I already held, that I was unlikely to be the first person to have a given idea. It is the same instinct behind the interruption in Table 3, "havent people done all this already".

That learning curve is too long and too costly to leave to each user. Current tools are visibly cautious in a few high-stakes domains, such as medical advice, but give little comparable signal when they are over-promising elsewhere. Two signals would have shortened my two months: the tool saying when it is answering from memory rather than from a source, and the tool becoming more cautious as the magnitude of a decision grows. Where I would not want the tool to take over is the redirection itself. In my experience the model does not think deeply enough on its own, so much depends on the angles and creative approaches the user brings. The most useful help at that point would be a tool that asks rather than decides: I can see ways A and B; are there others? That keeps the user engaged and stops the thinking from being outsourced entirely. My position, then, is that the warning signs should be built into the system, and the redirection should stay with the user, prompted but not replaced.

Beyond one user. The natural next study is several experienced users, including programmers, which is where the cycle's relevance to PPIG is most direct. One attendee at the workshop reported seeing similar behaviour in her students' use of AI coding assistants, which suggests the comparison is worth making. Such a study should code events with human raters and compute agreement, and should record every model turn, not only interruptions, so that the rapport hypothesis can be tested.

7. Limitations

One user, studying himself. All data come from the author's sessions, and the author designed the codebook, chose which sessions to code, and interprets the results. The findings describe one practice. The fit between events and codebook types shows that the author's vocabulary is internally consistent; it does not show that the types are the right ones for anyone else.

No human inter-rater agreement. The codebook was developed by one person and applied by LLMs. No second human coded any transcript. This is the most important gap in the method.

LLMs as coders. The coding models are the same kind of system as the one being studied and may share its blind spots. Coding model and coding mode are confounded, and live coding means Claude coded interruptions of itself.

Changing instrument. The codebook gained a sixth type during collection, receipts do not record the same fields, and triangulation was applied to a subset only.

Inclusion is a judgement. What counts as a redirection, and whether the user intended one, are judgement calls. Table 3 shows the models making those calls differently.

Emotionally charged sessions. In personal and reflective sessions, the line between conversation and

interruption is hardest to draw, and the cycle fits least well.

8. Conclusion

This paper has described how one experienced user interrupts LLMs during real work, using 611 coded events from 29 sessions. A six-stage cycle describes the clearest interruptions well, but most receipts contain at least one event it does not fit, and the three models used to code the data disagree substantially about what an interruption is. The most consistent observation is also the most uncomfortable: the output that most needed interrupting was often the most fluent, and the cue for interrupting it is a felt sense that the output is not about anything in particular.

The practice described here took me two months and real costs to learn. I do not think every user should have to learn it that way. Tools could say when they are answering from memory and become more careful as the stakes rise, while leaving the redirection to the user and asking for the user's angles rather than supplying their own. Whether that design shortens the learning curve for others is an empirical question. The next study should put it to several experienced users, programmers among them, with human coders and with every model turn recorded, so that the effect of an accumulating run of accepted turns can finally be measured rather than asserted.

9. Acknowledgements

This work was conducted independently, without external funding. The author thanks the PPIG 2026 reviewer, whose detailed comments shaped this revision, and the workshop attendees for their questions. The LLMs named in §3 were used as coding instruments. LLMs were also used as writing assistants in preparing this paper; all claims and numbers were checked by the author against the stored receipts.

10. References

Bender, E. M., & Hanna, A. (2025). *The AI con: How to fight big tech's hype and create the future we want*. New York, NY, USA: Harper.

Buçinca, Z., Malaya, M. B., & Gajos, K. Z. (2021). To trust or to think: Cognitive forcing functions can reduce overreliance on AI in AI-assisted decision-making. *Proceedings of the ACM on Human-Computer Interaction*, *5*(CSCW1), 188:1–188:21. doi: 10.1145/3449287

Couffe, C., & Michael, G. A. (2017). Failures due to interruptions or distractions: A review and a new framework. *The American Journal of Psychology*, *130*(2), 163–181. doi: 10.5406/amerjpsyc.130.2.0163

Cox, A. L., Gould, S. J. J., Cecchinato, M. E., Iacovides, I., & Renfree, I. (2016). Design frictions for mindful interactions: The case for microboundaries. In *Proceedings of the 2016 CHI conference extended abstracts on human factors in computing systems* (pp. 1389–1397). New York, NY, USA: ACM. doi: 10.1145/2851581.2892410

Desai, S., & Twidale, M. (2023). Using playful metaphors to conceptualize practical use of ChatGPT: An autoethnography. In *Proceedings of the Association for Information Science and Technology* (Vol. 60, pp. 565–569). Wiley. doi: 10.1002/pra2.816

Drosos, I., Sarkar, A., Xu, X., & Toronto, N. (2025). *"It makes you think": Provocations help restore critical thinking to AI-assisted knowledge work.* doi: 10.48550/arXiv.2501.17247

Epperson, W., Bansal, G., Dibia, V., Fourney, A., Gerrits, J., Zhu, E., & Amershi, S. (2025). Interactive debugging and steering of multi-agent AI systems. In *Proceedings of the 2025 CHI conference on human factors in computing systems* (pp. 1–15). ACM. doi: 10.1145/3706598.3713581

Fowler, M. (1999). *Refactoring: Improving the design of existing code*. Reading, MA, USA: Addison-Wesley.

Garg, S. (2026, April). *Refusing the shallow version.* SGistic. Available at `https://www.sgistic.com/p/refusing-the-shallow-version`.

Hatem, R., Simmons, B., & Thornton, J. E. (2023). A call to address AI "hallucinations" and how healthcare professionals can mitigate their risks. *Cureus*, *15*(9), e44720. doi: 10.7759/cureus.44720

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., … Fung, P. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys*, *55*(12), 1–38. doi: 10.1145/3571730

Lee, H.-P. H., Sarkar, A., Tankelevitch, L., Drosos, I., Rintel, S., Banks, R., & Wilson, N. (2025). The impact of generative AI on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers. In *Proceedings of the 2025 CHI conference on human factors in computing systems* (pp. 1–22). New York, NY, USA: ACM. doi: 10.1145/3706598.3713778

Lee, J. D., & See, K. A. (2004). Trust in automation: Designing for appropriate reliance. *Human Factors*, *46*(1), 50–80. doi: 10.1518/hfes.46.1.50_30392

Norman, D. A., & Shallice, T. (1986). Attention to action: Willed and automatic control of behavior. In R. J. Davidson, G. E. Schwartz, & D. Shapiro (Eds.), *Consciousness and self-regulation: Advances in research and theory* (Vol. 4, pp. 1–18). New York, NY, USA: Plenum Press. doi: 10.1007/978-1-4757-0629-1_1

Panke, S. (2025). How can (A)I research this? An autoethnographic exploration of generative AI in research, teaching and instructional design. *Journal of Teacher Education*, *76*(3), 230–244. doi: 10.1177/00224871251325065

Sarkar, A. (2024). Intention is all you need. In *Proceedings of the 35th annual conference of the Psychology of Programming Interest Group (PPIG 2024).* University of Liverpool, UK. doi: 10.48550/arXiv.2410.18851

Sarkar, R., Sarrafzadeh, B., Chandrasekaran, N., Rangan, N., Resnik, P., Yang, L., & Jauhar, S. K. (2025). *Conversational user-AI intervention: A study on prompt rewriting for improved LLM response generation.* doi: 10.48550/arXiv.2503.16789

Tankelevitch, L., Glassman, E. L., He, J., Kazemitabaar, M., Kittur, A., Lee, M., … Subramonyam, H. (2025). Tools for thought: Research and design for understanding, protecting, and augmenting human cognition with generative AI. In *Extended abstracts of the CHI conference on human factors in computing systems (CHI EA '25)* (pp. 1–8). Yokohama, Japan: ACM. doi: 10.1145/3706599.3706745

Tankelevitch, L., Kewenig, V., Simkute, A., Scott, A. E., Sarkar, A., Sellen, A., & Rintel, S. (2024). The metacognitive demands and opportunities of generative AI. In *Proceedings of the 2024 CHI conference on human factors in computing systems* (pp. 1–24). New York, NY, USA: ACM. doi: 10.1145/3613904.3642902

Zhang, Z. T., von Felten, N., Reicherts, L., Tankelevitch, L., Guan, Z., Rintel, S., … Dalsgaard, P. (2026). Tools for thought: Understanding, protecting, and augmenting human cognition with generative AI—from vision to implementation. In *Extended abstracts of the 2026 CHI conference on human factors in computing systems (CHI EA '26)* (pp. 1–7). Barcelona, Spain: ACM. doi: 10.1145/3772363.3778733

Exploring the impact of a single TPL session on Primary School Teachers' Domain-Specific Self-Esteem

Amanda O'Farrell
School of Enterprise Computing
& Digital Transformation
TU Dublin
amanda.ofarrell@tudublin.ie

Keith Quille
School of Enterprise Computing
& Digital Transformation
TU Dublin
keith.Quille@tudublin.ie

Keith Nolan
School of Enterprise Computing
& Digital Transformation
TU Dublin
keith.nolan@tudublin.ie

Abstract

The introduction of Ireland's redeveloped Primary School Curriculum positions Digital Technology (DT) as a core area of learning, creating new demands for primary teachers, many of whom lack prior preparation. This study examines whether a single, targeted Teacher Professional Learning (TPL) intervention can enhance teachers' domain-specific self-esteem in DT. A quantitative pre–post survey design was employed with participants attending targeted workshops. DT self-esteem was measured using an adapted, reliable instrument, and principal component analysis was used to generate a composite score. A paired-samples t-test revealed a statistically significant increase in DT self-esteem following the intervention, with a moderate-to-large effect size. Reduced variability in post-scores indicated greater consistency among participants. The findings suggest that brief, focused TPL interventions can meaningfully enhance teachers' confidence in DT, supporting their readiness to engage with curriculum reform.

1. Introduction

In September 2025, Ireland introduced a redeveloped Primary School Curriculum, representing the most significant reform of primary education in over two decades. The curriculum is organised around five key learning areas, Arts Education, Language, Social and Environmental Education, Wellbeing, and STEM Education, reflecting a national emphasis on critical thinking, problem-solving, and integrated learning through engaging, pupil-centred approaches (National Council for Curriculum and Assessment, 2025).

Within the STEM curriculum, the Technology strand has been significantly redeveloped, with Digital Technology (DT) integrated across all year levels. This represents a new instructional requirement for primary teachers. However, initial teacher education programmes have not traditionally included substantial preparation in digital technologies, computational thinking, or coding. As a result, many teachers are likely to lack confidence and experience in this area, creating a significant professional learning challenge alongside this curriculum innovation.

In response, the EDTips (Enabling Digital Technology in Primary Schools) project was launched in 2024 to support teacher readiness (Nolan et al., 2024). The project developed a curriculum-aligned MOOC featuring scaffolded lessons, instructional videos, and PRIMM-based pedagogical supports. It also delivered nationwide Teacher Professional Learning (TPL) workshops in 2025, offering hands-on experience with tools such as the BBC micro:bit and Rangers by Tangibl. During these workshops, pre- and post-workshop surveys were conducted using established measures of domain-specific self-esteem (Rosenberg, 1965; Bergin & Reilly, 2005b), which have been widely applied in computing education research (Quille, Bergin, & Mooney, 2015a).

This study examines whether participation in a single targeted TPL workshop can meaningfully enhance teachers' DT self-esteem. Addressing this issue is critical, as digital literacy and computational thinking are increasingly essential competencies, and effective teacher support is central to the successful implementation of large-scale curriculum reform. This study provides evidence on the effectiveness of brief, scalable professional learning interventions in supporting primary teachers' readiness to implement Digital Technology within a newly reformed curriculum.

2. Literature Review

2.1. Out-of-Field Teaching and its Consequences

Out-of-field (OOF) teaching refers to the practice of assigning teachers to subjects for which they have not received formal pre-service preparation or qualifications (Ingersoll, 1999). International research identifies OOF teaching as a widespread issue with significant implications for teacher wellbeing, instructional quality, and student learning (Hobbs & Porsch, 2022). Studies consistently show that OOF teachers experience reduced confidence, difficulties with content mastery, heightened stress, and a diminished sense of professional efficacy, all of which can negatively affect teaching performance and student outcomes (Castorico & Dioso, 2023; Sharplin, 2014; Porsch & Whannell, 2019; Wheeley, Klieve, Park, & Du Plessis, 2023).

In addition, OOF teaching has been linked to lower job satisfaction and increased attrition, suggesting longer-term consequences for workforce stability (Sharplin, 2014). These challenges are often attributed to inadequate subject knowledge, limited pedagogical content knowledge, and insufficient preparation for the specific demands of the assigned subject (Du Plessis, 2015). Such difficulties are further intensified when the subject is newly introduced or undergoing curriculum reform, requiring teachers to rapidly adapt to unfamiliar content while maintaining instructional quality (Castorico & Dioso, 2023).

Beyond pedagogical and organisational challenges, the literature highlights the psychological and identity-related dimensions of OOF teaching. Hobbs (2013) conceptualises OOF teaching as a "boundary-crossing" process that disrupts established practices and necessitates the renegotiation of professional identity. Empirical studies similarly indicate that teaching outside one's area of expertise is associated with reduced confidence and self-efficacy, reflecting tensions in teachers' sense of competence (Paolucci, Ní Ríordáin, & O'Dwyer, 2021; O'Meara & Faulkner, 2021). Collectively, these findings suggest that the consequences of misassignment extend beyond instructional concerns to encompass broader psychological and professional impacts.

Taken together, these pedagogical and psychological challenges highlight the importance of understanding the role of teachers' self-evaluations, directing attention to how constructs such as self-esteem may shape their experiences in OOF contexts.

2.2. Self-Esteem from Global to Domain-Specific: Implications for Computing

Early conceptualisations treated self-esteem as a unitary, stable evaluation of global self-worth (Rosenberg, 1965), yet this view has been widely critiqued for its limited explanatory power, with global self-esteem often showing weak relationships with performance outcomes (Baumeister, Campbell, Krueger, & Vohs, 2003). In response, multidimensional models reconceptualised self-evaluations as hierarchical and domain-specific (Marsh & Shavelson, 1985). Empirical work demonstrates that self-perceptions are differentiated across domains (Marsh, 1990) and that domain-specific self-concepts are more predictive of domain-relevant outcomes than global self-esteem (Marsh & Craven, 2006; Valentine, DuBois, & Cooper, 2004).

This perspective is extended by contingencies of self-worth theory, which emphasises that self-esteem depends on success in personally valued domains (Crocker & Wolfe, 2001), as well as by self-determination theory, which highlights tensions between contingent and intrinsic self-evaluation (Ryan & Deci, 2000). However, such frameworks remain relatively broad, limiting their applicability in highly specialised educational contexts.

Consequently, research has shifted toward finer-grained constructs, particularly within academic do-

mains. Subject-specific self-concepts are empirically distinct and hierarchically organised (Marsh, 1990; Marsh, Trautwein, Lüdtke, Köller, & Baumert, 2005). These findings indicate that explanatory power increases as constructs align more closely with the specific domain of performance.

In computing, this has led to the use of self-concept and self-efficacy to capture task-specific competence beliefs (Bandura, 1997; Bong & Skaalvik, 2003). Computer self-efficacy is associated with programming performance and learning outcomes (Compeau & Higgins, 1995; Cassidy & Eachus, 2002), with earlier studies linking self-efficacy to programming success (Ramalingam, LaBelle, & Wiedenbeck, 2004; Bergin & Reilly, 2005a) and to engagement-related behaviours in programming contexts (Loksa et al., 2016). These constructs are often used as operational proxies for domain-specific self-evaluations, particularly when internalised within learner identity.

Empirical work by Quille and Bergin (2018) reinforces this position, demonstrating that programming self-efficacy predicts student success in introductory computing and contributes to broader patterns of engagement and retention. Their findings highlight the contextual and pedagogical formation of these beliefs, rather than their dependence on fixed ability. Moreover, such beliefs are closely tied to identity formation and participation in computing (Carlone & Johnson, 2007; Gee, 2000), influencing persistence in the field (Lewis, Yasuhara, & Anderson, 2011).

However, increasing reliance on narrowly defined constructs introduces conceptual challenges. Although self-efficacy offers strong predictive precision, it contributes to fragmentation across related constructs, with substantial overlap observed between self-efficacy, self-concept, and broader self-evaluative beliefs (Bong & Skaalvik, 2003; Judge, Erez, Bono, & Thoresen, 2003). This convergence risks reducing self-esteem to competence-based judgements, thereby neglecting its broader evaluative and affective dimensions.

In sum, the literature reflects a shift from global to highly specific self-evaluations, with precision increasing alongside predictive power. Quille and Bergin's work exemplifies this trend in computing education but also highlights an ongoing tension between methodological precision and theoretical coherence, indicating the need for more integrative frameworks that reconcile global and domain-specific perspectives.

2.3. Computer Science Curriculum Reform in Ireland

Computer Science curriculum reform in Ireland has been driven by increasing societal and economic demands for digital competence and computational thinking. National policies, including the Digital Strategy for Schools 2015–2020 and the Digital Strategy for Schools to 2027, position digital learning as central to teaching, learning, and assessment, framing computer science as a core component of contemporary education (Department of Education and Skills, 2015; Department of Education, 2022).

At post primary level, reform is most evident in the introduction of Leaving Certificate Computer Science and the expansion of Junior Cycle short courses in areas such as coding and computational thinking (National Council for Curriculum and Assessment, 2017, 2018). These developments mark a shift from earlier conceptions of ICT as a support skill to recognition of computer science as a distinct disciplinary field requiring specialised content and pedagogical knowledge. At primary level, computer science is embedded across the curriculum within the redeveloped Primary Curriculum, where "being a digital learner" is identified as a key competency, particularly within the Mathematics, Science and Technology learning area (National Council for Curriculum and Assessment, 2020). The technology component of this curriculum is referred to here as Digital Technology (DT). This approach places increasing responsibility on generalist teachers to integrate DT content, often without substantial pre-service preparation.

Despite strong policy ambition, DT reform has significant implications for teachers' professional roles. International research indicates that curriculum expansion has contributed to increased out-of-field (OOF) teaching, particularly at post-primary level, where teachers without formal qualifications in computing are often assigned to Computer Science or related DT subjects (Ingersoll, 1999; Hobbs & Porsch, 2022). The specialised and rapidly evolving nature of DT intensifies challenges commonly associated

with OOF teaching, including reduced confidence, increased stress, and concerns about instructional quality (Sharplin, 2014; Hobbs & Porsch, 2022). At the same time, national policy frameworks emphasise the importance of digital learning and curriculum reform within Irish education, reinforcing the centrality of DT in contemporary teaching and learning (Department of Education and Youth, 2025).

These challenges are not only structural but also psychological. Research indicates that teachers' engagement and persistence are shaped more strongly by domain-specific self-concept and self-efficacy, than by global self-esteem (Rosenberg, Schooler, Schoenbach, & Rosenberg, 1995). In computing contexts, lower domain-specific self beliefs are associated with reduced willingness to innovate and greater risk of disengagement (Bergin & Reilly, 2005a; Quille & Bergin, 2018). While national professional learning supports are available through many outreach programmes and Oide, the national support service for teachers and school leaders, they frequently occur alongside, rather than prior to, curriculum implementation, requiring teachers to develop DT expertise while already delivering the subject (Boeskens, Nusche, & Yurita, 2020).

Overall, computer science curriculum reform in Ireland provides a salient context for examining the interaction between OOF teaching and domain-specific self-esteem. DT represents a timely and appropriate focus for investigating how curriculum change influences teachers' psychological readiness to enact reform, with implications for instructional quality, professional wellbeing, and sustainability of the teaching workforce.

2.4. Teacher Professional Development in Computing in Ireland: Evidence of Impact

Irish research on teacher professional learning/continued professional learning (TPL/CPD) in computing presents a consistently positive picture of professional development during a period of rapid curriculum reform, subject novelty, and significant levels of OOF teaching (McGarr, McInerney, Exton, & Power, 2020; Ó Gallchóir & McGarr, 2022), with similar patterns observed in broader UK and Ireland studies (Sentance et al., 2022).

However, this body of research has been situated almost entirely at post primary level, reflecting the introduction of Leaving Certificate Computer Science and associated Junior Cycle developments (National Council for Curriculum and Assessment, 2017, 2018). While not directly focused on primary education, this literature provides important insights into how professional development can support teachers' competence, confidence, and professional identity during the early stages of computing curriculum enactment (McGarr et al., 2020; O'Donnell, 2024).

Studies examining the rollout of Leaving Certificate Computer Science indicate that well designed and sustained professional development can enhance teacher confidence, pedagogical practice, and professional identity, even where teachers initially lack formal qualifications in computing (McGarr et al., 2020; McGarr, Goos, McInerney, Johnston, & Fleming, 2022). Commissioned national research evaluating Oide, formally the Professional Development Service for Teachers (PDST), initiatives found that participating teachers reported increased confidence in teaching computer science, alongside observable shifts towards inquiry based and problem solving pedagogical approaches. Notably, these benefits extended to teachers teaching computer science out-of-field, with structured CPD supporting both subject understanding and pedagogical effectiveness (McGarr et al., 2020).

Early research on the introduction of post-primary computer science suggests improvements in teacher preparedness and reduced anxiety when teaching a new subject (Quille & Bergin, 2018). Teachers highlighted the role of CPD in clarifying curriculum and assessment, contributing to increased confidence during initial implementation. Further research on teachers involved in national CPD provision highlights their development of significant pedagogical expertise and evolving professional identities linked to leadership roles, while also pointing to challenges in sustaining these contributions within existing structures (O'Donnell, 2024).

Although this evidence base does not derive from primary school contexts, and there appears to be little prior research in this area, it offers important lessons for the expansion of computing and DT education

across educational sectors. It demonstrates how sustained, collaborative, and identity affirming professional development can support teachers navigating curriculum reform, especially where OOF teaching is prevalent and specialist preparation is limited. These insights provide a useful foundation for considering how professional learning might support primary teachers as computing related curriculum reform continues to develop.

Thus, this paper aims to investigate the following question:

Can a single intervention have a positive impact on teachers' self-esteem in Digital Technology?

3. Methodology

3.1. Research Design

This study employed a quantitative pre-post survey design to investigate whether participation in a single TPL workshop could increase primary teachers' self-esteem in the DT strand of the redeveloped Irish primary curriculum. A pre-post design was selected because it enables direct measurement of change in participants' self-esteem over a short intervention period, providing a clear indicator of whether the TPL workshop produced an immediate impact on teachers' self-esteem in DT. The quantitative approach enables the production of standardised, comparable data across participants and facilitates statistical analysis of patterns and variations in self-esteem rating.

3.2. Participants

Participants in this study were primary school teachers who attended EDTips TPL workshops held between February and November 2025 across various locations around the country. The group represented a broad spectrum of teaching experience, and the majority (82%) reported having some prior exposure to DT related content, either through classroom teaching or involvement in extra-curricular activities. Participation in the research was entirely voluntary, and only teachers who completed both the pre- and post-survey were included in the final analysis. The survey also collected demographic information, including years of teaching experience, class level taught, and previous engagement with technology-based instruction, to provide contextual insight into the participant cohort. As participation was voluntary, the sample may be subject to self-selection bias, with participants potentially more positively disposed towards DT or professional learning.

3.3. The EDTips TPL Intervention

The TPL workshop formed part of the national Research Ireland EDTips project (Nolan et al., 2024), which aims to support teachers in implementing the newly introduced DT strand. Each TPL workshop introduced participants to the EDTips online learning environment. Teachers engaged with scaffolded lesson plans, sample activities, and videos based on the PRIMM pedagogical model. Hands-on activities were included using the BBC micro:bit and the Rangers by Tangible offline toolkit, enabling teachers to experience practical, classroom-ready exemplars. The TPL workshop lasted approximately two hours and was designed as a short, domain-specific self-esteem building intervention rather than a comprehensive training programme. The TPL workshop will be referred to as the 'intervention' in later sections of this paper.

3.4. Instruments

Teacher self-esteem in Digital Technology was assessed using a structured survey instrument administered immediately before and after each workshop. In this study, domain-specific self-esteem is operationalised as teachers' evaluative judgement of their competence and confidence in DT, reflecting overlap with self-efficacy constructs in applied educational contexts. The self-esteem items were adapted from the validated measures developed by Bergin and Reilly (2006), ensuring strong alignment with established approaches to assessing domain-specific teaching self-esteem (Bergin & Reilly, 2005a; Quille & Bergin, 2018, 2019). To ensure the adaption of the scale was fit for purpose, Cronbach's alpha was run on the 10 items giving a good internal consistency of 0.904. Each item used a Likert-type response scale, enabling enabling participants to rate their perceived self-esteem in DT along a graded continuum. Iden-

tical items were included in both the pre- and post-intervention surveys, allowing for direct comparison and the calculation of change scores. The pre-survey also incorporated a set of demographic questions designed to contextualise the sample, explore potential patterns associated with teachers' backgrounds, and identify barriers that might influence participation in TPL opportunities.

3.5. Data Collection Procedures

At the commencement of the intervention, teachers were invited to complete a pre-intervention survey online using Microsoft Forms. Participation in the questionnaire was voluntary, and participants were informed of their right to withdraw from the study at any time without penalty. Completion of the pretest survey required an average of 8 min and 13 s. Following completion of the intervention activities, a post-intervention survey was administered using the same online platform. The post-intervention survey required an average of 4 min and 4 s to complete. In December 2025, the survey data was exported from the survey platform for subsequent analysis.

3.6. Data analysis techniques

Data analysis was conducted to establish whether a single intervention could positively impact teachers' self-esteem in DT. In December 2025 data from the surveys was collated, comprising 50 pre-intervention and 49 post-intervention surveys. Prior to analysis, data were screened for completeness, outliers, and consistency. Only fully matched pre- and post-responses were retained. This resulted in a final matched sample of n = 41 participants. No extreme outliers were identified, and Likert-scale responses were treated as approximately continuous, consistent with established practice in self-esteem research, particularly where multiple items are combined to form a composite scale.

3.6.1 Domain-Specific Self-esteem Measurement

In this study, a domain-specific DT self-esteem instrument was employed, drawing on this established adaptation approach within computing education (Bergin & Reilly, 2005a; Quille, Bergin, & Mooney, 2015b). The scale operationalises DT self-esteem as a unidimensional construct reflecting an individual's overall evaluative judgement of their competence and confidence in DT. DT self-esteem is understood here as a unidimensional evaluative construct encompassing perceptions of competence and confidence within the DT domain.

Consistent with this conceptualisation, the instrument comprises multiple observed items designed to function as parallel indicators of a single underlying construct. Each item reflects variations in perceived competence, confidence, or self-doubt in DT, rather than measuring distinct subdimensions. As such, the items are expected to exhibit substantial intercorrelations, reflecting shared variance attributable to overall DT self-esteem.

Negatively worded items were reverse coded prior to analysis to ensure that higher scores consistently indicated higher levels of DT self-esteem. Internal consistency of the scale was assessed using Cronbach's alpha to confirm the reliability of the instrument.

The same instrument was administered in both pre- and post-intervention surveys. The adapted items were as follows:

- On the whole, I am satisfied with my DT progress.
- At times, I think that I am no good at all at DT.
- I feel that I have a number of good DT qualities.
- I am able to complete DT tasks as well as most other colleagues.
- I feel that I do not have much DT ability to be proud of.
- I certainly feel useless at DT at times.
- I feel that I am a person of worth, at least on a level plane with other colleagues.

- I wish I could have more respect for my DT ability.
- All in all, I am inclined to feel that I am a failure at DT.
- I take a positive attitude towards my DT ability.

3.6.2 Principal Component Analysis (PCA)

Principal component analysis (PCA) was employed to reduce the ten self-esteem items to a single composite score representing overall DT self-esteem for each participant, a commonly used dimensionality reduction technique for summarising correlated variables into a smaller set of components (Abdi & Williams, 2010; Jolliffe & Cadima, 2016). Sampling adequacy and factorability were assessed prior to PCA using standard diagnostic tests. The same set of items was used in both the pre- and post-intervention surveys.

Given the aim of assessing change over time, PCA was conducted on the combined (pooled) dataset consisting of both pre- and post-intervention responses. Pooled PCA involves performing a single analysis across all observations to identify a common component structure shared across conditions (Jolliffe & Cadima, 2016). This approach is appropriate in pre–post designs, as it ensures that the same weighting of items is applied at both time points, thereby enabling meaningful comparison of scores over time (Abdi & Williams, 2010; Tabachnick & Fidell, 2019).

The resulting principal component scores were used as the overall measure of DT self-esteem in subsequent analyses.

To assess whether the intervention resulted in a statistically significant change in DT self-esteem, a paired-samples t-test was conducted on pre- and post-intervention composite scores. This test is appropriate for repeated measures designs where the same participants are measured at two time points.

In addition to statistical significance, the magnitude of the effect was assessed using Cohen's d for paired samples. Effect sizes were interpreted using conventional thresholds (small ≈ 0.2, medium ≈ 0.5, large ≈ 0.8), providing an estimate of practical significance independent of sample size.

Prior to analysis, assumptions of normality for the difference scores were assessed through visual inspection of distribution plots and statistical tests of normality.

4. Results

This section presents the results of the study examining the impact of a single intervention on teachers' self-esteem in DT. Principal component analysis identified PC1 as a composite measure of DT self-esteem, which was used in all subsequent analyses. In our case, the more positive a PC1 score, the higher the DT self-esteem. Subsequently, the lower or more negative the PC1 score, the lower the DT self-esteem. The results are organised into descriptive statistics, inferential analysis, and a summary in relation to the research question.

4.1. Descriptive statistics

A total of 41 participants were included in the analysis. Descriptive statistics for PC1 scores before and after the intervention are presented in Table 1. The mean pre-intervention PC1 score was -0.64 (SD = 2.55), increasing to 0.64 (SD = 1.71) post-intervention, indicating a clear positive shift in DT self-esteem. The median similarly increased from -0.25 to 0.94, supporting this upward trend.

Variability in scores decreased following the intervention. The pre-intervention scores showed a wide range (min = -7.31, max = 3.86), whereas the post-intervention scores were more constrained (min = -2.38, max = 4.58), indicating greater consistency among participants after the intervention.

Overall, the mean increase in PC1 scores was 1.28.

4.2. Statistical Analysis

Firstly, we assessed the normality of the data. To do this, we assessed the normality of the changes between pre and post. The change scores did not significantly deviate from normality, Shapiro-Wilk p =

Statistic	Pre-Intervention	Post-Intervention
N	41	41
Mean	-0.64	0.64
Standard Deviation	2.55	1.71
Minimum	-7.31	-2.38
25^{th} Percentile	-2.45	-0.37
Median	-0.25	0.94
75^{th} Percentile	1.01	1.52
Maximum	3.86	4.58

Table 1 – Descriptive statistics before and after the intervention.

.146, so a paired-samples t-test was considered appropriate.

A paired samples t-test was conducted to examine whether the change in PC1 scores was statistically significant (Table 2).

Results showed a significant increase in PC1 scores following the intervention, t(40) = -4.98, p < .00001, which would seem to indicate an improvement in DT self-esteem. The effect size was then calculated which was moderate to large (Cohen's d = 0.78), suggesting that the observed change is both statistically significant and practically meaningful.

Statistic	Value
N	41
t	-4.98
Degrees of Freedom	40
p-value	$<$ .00001
Cohen's d	0.78
Mean Change	1.28

Table 2 – Statistical test results for the intervention.

4.3. Visualisation of Results

Figure 1 illustrates the distribution of PC1 scores representing DT self-esteem for pre- and post-conditions, shown as boxplots (median, interquartile range, and range). Black dots indicate means ($\pm$ SD). DT self-esteem increased significantly from pre- to post-condition (p = 0.000013, Cohen's d = 0.78). The figure visually reinforces the statistical findings, showing an upward shift in the distribution of scores. The post-intervention condition exhibits a higher median and mean, along with reduced variability relative to the pre-intervention condition.

4.4. Summary of Findings in Relation to the Research Question

This study sought to answer the research question:

"Can a single intervention have a positive impact on teachers' self-esteem in Digital Technology?"

The results provide evidence supporting this hypothesis. PC1 scores, representing DT self-esteem, increased significantly following the intervention, with a mean increase of 1.28. Both descriptive and inferential analyses indicate a consistent upward shift in scores, accompanied by reduced variability.

The statistically significant result (p < .00001), combined with a moderate-to-large effect size (Cohen's d = 0.78), suggests that the intervention had a meaningful and measurable impact on participants' DT self-esteem. Overall, these findings indicate that even a single intervention can produce improvements in teachers' DT self-esteem.

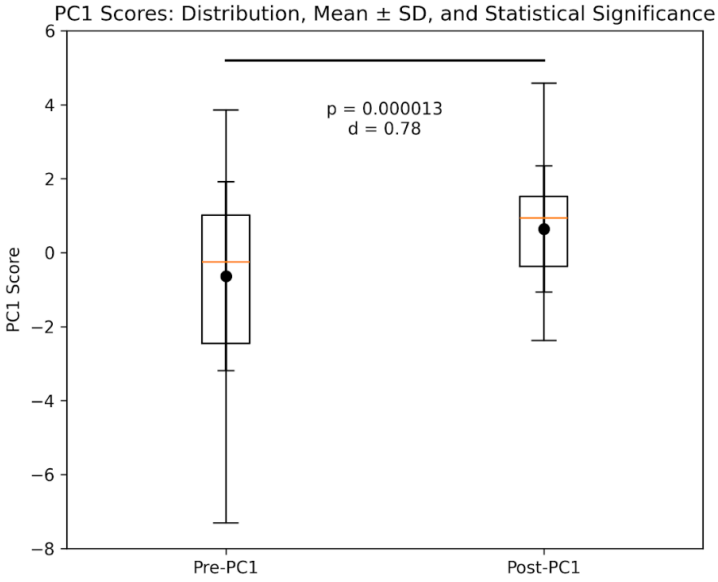


Figure 1 – Distribution of PC1 scores representing DT self-esteem for pre- and post-conditionse

5. Discussion

5.1. Overview of Findings

This study examined whether a single intervention could positively influence teachers' self-esteem in Digital Technology (DT). The findings demonstrate a statistically significant improvement in DT self-esteem following the intervention. PC1 scores, representing DT self-esteem, increased substantially from pre- to post-intervention, with a moderate-to-large effect size (Cohen's d = 0.78). These results indicate that the observed change was not only statistically significant but also practically meaningful.

In addition to this increase, a reduction in variability was observed, indicating greater consistency in participants' scores following the intervention. Collectively, these findings provide strong evidence that even a brief intervention can have a measurable impact on teachers' DT self-esteem

5.2. Interpretation of Results

The observed increase in PC1 scores suggests that the intervention positively influenced teachers' perceptions of their self-esteem in DT. The upward shift in both mean and median values indicates that the improvement was broadly distributed across participants, rather than driven by a small subset.

The reduction in variability is particularly noteworthy. Pre-intervention scores showed considerable dispersion, reflecting variation in confidence levels among participants. Post-intervention, the narrower distribution indicates a convergence toward more similar levels of DT self-esteem. This suggests that the intervention may have been effective not only in increasing overall confidence but also in supporting participants with lower initial self-esteem.

These findings are consistent with existing research demonstrating that professional learning experiences can enhance teachers' self-esteem. The results of this study extend this evidence by suggesting that even a single, focused intervention may be sufficient to produce meaningful improvements in domain-specific self-esteem.

5.3. Implications for Practice

The findings have important implications for teacher professional development in DT. The apparent significant improvement observed after a single intervention suggests that relatively short, targeted programmes can be effective in enhancing teachers' domain-specific self-esteem.

This has practical value for educational settings where time and resources are limited. Rather than requiring extensive or long-term training, well-designed short interventions may provide a scalable and

efficient way to support teachers. Increased DT self-esteem may, in turn, encourage greater engagement with digital tools, more frequent integration of DT into classroom practice, and increased willingness to adopt innovative approaches.

The reduction in variability also highlights the potential of such interventions to promote greater equity among teachers by supporting those with lower initial domain-specific self-esteem

5.4. Limitations and Future Research

Some limitations should be acknowledged. The absence of a comparison condition limits the extent to which observed changes can be attributed exclusively to the intervention, as alternative explanations (e.g., maturation or external influences) cannot be ruled out. In addition, the modest sample size constrains the generalisability of the findings.

Future research should aim to replicate these findings with larger and more diverse samples to strengthen generalisability. Adopting a longitudinal design would allow for stronger causal inference and provide insight into whether the observed improvements are sustained over time.

Further work is also needed to identify which elements of the intervention are most effective and how gains in DT self-esteem translate into changes in teaching practice and student outcomes.

6. Conclusion

This study set out to examine whether a single, targeted Teacher Professional Learning (TPL) intervention could positively influence primary teachers' self-esteem in Digital Technology (DT) within the context of Ireland's newly redeveloped primary curriculum. The findings demonstrate a statistically significant and practically meaningful improvement in teachers' DT self-esteem following participation in the EDTips workshop. The observed increase in composite self-esteem scores, alongside reduced variability, indicates that the intervention was effective not only in raising overall self-esteem but also in supporting teachers with initially lower levels of confidence.

These findings contribute to the growing body of research highlighting the importance of domain-specific self-evaluations in shaping teachers' engagement with new curricular areas. In the context of DT, where many primary teachers are required to teach unfamiliar content, enhancing self-esteem appears to play a crucial role in supporting readiness to engage with and implement curriculum reform. The results suggest that even brief, well-designed professional learning experiences can have a meaningful impact on teachers' perceived competence and confidence in this domain.

From a practical perspective, the study underscores the potential of short, scalable TPL interventions to support large cohorts of teachers during periods of curriculum change. Given the time and resource constraints often faced in educational systems, such approaches offer a feasible and efficient strategy for strengthening teacher readiness in emerging subject areas like DT.

While further research is required to explore the long-term sustainability of these effects and their translation into classroom practice and student outcomes, this study provides encouraging evidence that targeted professional learning can play a key role in supporting teachers' psychological preparedness for curriculum reform. Ultimately, strengthening teachers' domain-specific self-esteem represents an important step towards ensuring the successful implementation of Digital Technology within primary education.

7. References

Abdi, H., & Williams, L. J. (2010). Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, *2*(4), 433–459. Retrieved from `https://doi.org/10.1002/wics.101` doi: 10.1002/wics.101

Bandura, A. (1997). *Self-efficacy: The exercise of control*. New York: W. H. Freeman.

Baumeister, R. F., Campbell, J. D., Krueger, J. I., & Vohs, K. D. (2003). Does high self-esteem cause better performance, interpersonal success, happiness, or healthier lifestyles? *Psychological*

Science in the Public Interest, *4*(1), 1–44. Retrieved from https://doi.org/10.1111/1529-1006.01431 doi: 10.1111/1529-1006.01431

Bergin, S., & Reilly, R. (2005a). The influence of motivation and comfort-level on learning to program. In *Proceedings of the 17th psychology of programming interest group workshop* (pp. 293–304).

Bergin, S., & Reilly, R. (2005b). Programming: Factors that influence success. In *Proceedings of the 36th sigcse technical symposium on computer science education (sigcse '05)* (pp. 411–415). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1047344.1047480 doi: 10.1145/1047344.1047480

Bergin, S., & Reilly, R. (2006). Predicting introductory programming performance: A multi-institutional multivariate study. *Computer Science Education*, *16*(4), 303–323.

Boeskens, L., Nusche, D., & Yurita, M. (2020). *Policies to support teachers' continuing professional learning: A conceptual framework and mapping of oecd data* (Tech. Rep. No. 235). Paris: OECD Publishing. Retrieved from https://doi.org/10.1787/247b7c4d-en doi: 10.1787/247b7c4d-en

Bong, M., & Skaalvik, E. M. (2003). Academic self-concept and self-efficacy: How different are they really? *Educational Psychology Review*, *15*(1), 1–40. Retrieved from https://doi.org/10.1023/A:1021302408382 doi: 10.1023/A:1021302408382

Carlone, H. B., & Johnson, A. (2007). Understanding the science experiences of successful women of color: Science identity as an analytic lens. *Journal of Research in Science Teaching*, *44*(8), 1187–1218. Retrieved from https://doi.org/10.1002/tea.20237 doi: 10.1002/tea.20237

Cassidy, S., & Eachus, P. (2002). Developing the computer user self-efficacy (cuse) scale: Investigating the relationship between computer self-efficacy, gender, and experience with computers. *Journal of Educational Computing Research*, *26*(2), 133–153. Retrieved from https://doi.org/10.2190/JGJR-0KVL-HRF7-GCNV doi: 10.2190/JGJR-0KVL-HRF7-GCNV

Castorico, M. C., & Dioso, E. D. (2023). Teaching beyond the comfort zone: Experiences of out-of-field teaching. *International Journal of Advance Research and Innovative Ideas in Education*. (Available online)

Compeau, D. R., & Higgins, C. A. (1995). Computer self-efficacy: Development of a measure and initial test. *MIS Quarterly*, *19*(2), 189–211. Retrieved from https://doi.org/10.2307/249688 doi: 10.2307/249688

Crocker, J., & Wolfe, C. T. (2001). Contingencies of self-worth. *Psychological Review*, *108*(3), 593–623. Retrieved from https://doi.org/10.1037/0033-295X.108.3.593 doi: 10.1037/0033-295X.108.3.593

Department of Education. (2022). *Digital strategy for schools to 2027* (Tech. Rep.). Dublin: Government of Ireland.

Department of Education and Skills. (2015). *Digital strategy for schools 2015–2020: Enhancing teaching, learning and assessment* (Tech. Rep.). Dublin: Government of Ireland.

Department of Education and Youth. (2025). *Education plan 2025* (Tech. Rep.). Dublin: Government of Ireland.

Du Plessis, A. E. (2015). Effective education: Conceptualising the meaning of out-of-field teaching practices for teachers, teacher quality and school leaders. *International Journal of Educational Research*, *72*, 89–102. doi: 10.1016/j.ijer.2015.05.005

Gee, J. P. (2000). Identity as an analytic lens for research in education. *Review of Research in Education*, *25*, 99–125. Retrieved from https://doi.org/10.3102/0091732X025001099 doi: 10.3102/0091732X025001099

Hobbs, L. (2013). Teaching "out-of-field" as a boundary-crossing event: Factors shaping teacher identity. *International Journal of Science and Mathematics Education*, *11*(2), 271–297. doi: 10.1007/s10763-012-9333-4

Hobbs, L., & Porsch, R. (2022). *Out-of-field teaching across teaching disciplines and contexts*. Springer. Retrieved from https://doi.org/10.1007/978-981-16-9328-1 doi: 10.1007/978-981-16-9328-1

Ingersoll, R. M. (1999). The problem of underqualified teachers in american secondary schools. *Educational Researcher*, *28*(2), 26–37. Retrieved from https://doi.org/10.3102/0013189X028002026 doi: 10.3102/0013189X028002026

Jolliffe, I. T., & Cadima, J. (2016). Principal component analysis: A review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *374*(2065), 20150202. Retrieved from https://doi.org/10.1098/rsta.2015.0202 doi: 10.1098/rsta.2015.0202

Judge, T. A., Erez, A., Bono, J. E., & Thoresen, C. J. (2003). The core self-evaluations scale: Development of a measure. *Personnel Psychology*, *56*(2), 303–331. Retrieved from https://doi.org/10.1111/j.1744-6570.2003.tb00152.x doi: 10.1111/j.1744-6570.2003.tb00152.x

Lewis, C. M., Yasuhara, K., & Anderson, R. E. (2011). Deciding to major in computer science: A grounded theory of students' self-assessment of ability. In *Proceedings of the 7th international workshop on computing education research (icer '11)* (pp. 3–10). ACM. Retrieved from https://doi.org/10.1145/2016911.2016915 doi: 10.1145/2016911.2016915

Loksa, D., Ko, A. J., Jernigan, W., Oleson, A., Mendez, C. J., & Burnett, M. M. (2016). Programming, problem solving, and self-awareness: Effects of explicit guidance. In *Proceedings of the 2016 chi conference on human factors in computing systems* (pp. 1449–1461). ACM. Retrieved from https://doi.org/10.1145/2858036.2858252 doi: 10.1145/2858036.2858252

Marsh, H. W. (1990). A multidimensional, hierarchical model of self concept: Theoretical and empirical justification. *Educational Psychology Review*, *2*(2), 77–172. Retrieved from https://doi.org/10.1007/BF01322177 doi: 10.1007/BF01322177

Marsh, H. W., & Craven, R. G. (2006). Reciprocal effects of self-concept and performance from a multidimensional perspective: Beyond seductive pleasure and unidimensional perspectives. *Perspectives on Psychological Science*, *1*(2), 133–163. Retrieved from https://doi.org/10.1111/j.1745-6916.2006.00010.x doi: 10.1111/j.1745-6916.2006.00010.x

Marsh, H. W., & Shavelson, R. (1985). Self-concept: Its multifaceted, hierarchical structure. *Educational Psychologist*, *20*(3), 107–123. Retrieved from https://doi.org/10.1207/s15326985ep2003_1 doi: 10.1207/s15326985ep2003_1

Marsh, H. W., Trautwein, U., Lüdtke, O., Köller, O., & Baumert, J. (2005). Academic self-concept, interest, grades, and standardized test scores: Reciprocal effects models of causal ordering. *Child Development*, *76*(2), 397–416. Retrieved from https://doi.org/10.1111/j.1467-8624.2005.00853.x doi: 10.1111/j.1467-8624.2005.00853.x

McGarr, O., Goos, M., McInerney, C., Johnston, K., & Fleming, U. (2022). Implementing a professional development framework to assist the rollout of computer science in second-level schools in ireland. In C. Mouza, A. T. Ottenbreit-Leftwich, & A. Yadav (Eds.), *Professional development for in-service teachers: Research and practices in computing education* (pp. 207–222). Emerald.

McGarr, O., McInerney, C., Exton, C., & Power, J. (2020). *Exploring teachers' professional development to support the roll-out of computer science in irish second-level schools* (Tech. Rep.). Limerick: Lero / EPI-STEM, University of Limerick.

National Council for Curriculum and Assessment. (2017). *Short course in coding: Specification for junior cycle* (Tech. Rep.). Dublin: National Council for Curriculum and Assessment.

National Council for Curriculum and Assessment. (2018). *Leaving certificate computer science: Curriculum specification* (Tech. Rep.). Dublin: National Council for Curriculum and Assessment.

National Council for Curriculum and Assessment. (2020). *Draft primary curriculum framework* (Tech. Rep.). Dublin: National Council for Curriculum and Assessment.

National Council for Curriculum and Assessment. (2025). *Primary science, technology, engineering and mathematics (stem) education specification for primary and special schools.* Dublin. Retrieved from https://www.curriculumonline.ie/getmedia/70081350-c004-4773-8948-f70b0d4ef554/Primary-STE-and-MATHS-Spec-ENG.pdf

Nolan, K., Nolan, K., Hensman, S., O'Farrell, A., Faherty, R., Collins, M., ... Becker, B. A. (2024). En-

abling digital technology in primary schools. In *Proceedings of the 2024 conference on innovation and technology in computer science education (iticse 2024)* (Vol. 2, p. 823). Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3649405.3659515 doi: 10.1145/3649405.3659515

O'Donnell, C. (2024). Teachers seconded to continuing teacher education: Professional development and the paradox of reaching proficiency. *Irish Educational Studies*, *43*(4), 1455–1478. Retrieved from https://doi.org/10.1080/03323315.2023.2293148 doi: 10.1080/03323315.2023.2293148

Ó Gallchóir, C., & McGarr, O. (2022). An irish perspective on initial teacher education: How teacher educators can respond to an awareness of the 'absurd'. *Educational Philosophy and Theory*, *54*(7), 983–991. Retrieved from https://doi.org/10.1080/00131857.2021.1901080 doi: 10.1080/00131857.2021.1901080

O'Meara, N., & Faulkner, F. (2021). Professional development for out-of-field post-primary teachers of mathematics: An analysis of the impact of mathematics specific pedagogy training. *Irish Educational Studies*, 1–20. Retrieved from https://doi.org/10.1080/03323315.2021.1899026 doi: 10.1080/03323315.2021.1899026

Paolucci, C., Ní Ríordáin, M., & O'Dwyer, L. (2021). A critical examination of the impacts and lessons learned from a professional development program for out-of-field mathematics teachers. *European Journal of Mathematics and Science Education*, *2*(1), 47–62. Retrieved from https://doi.org/10.12973/ejmse.2.1.47 doi: 10.12973/ejmse.2.1.47

Porsch, R., & Whannell, R. (2019). Out-of-field teaching affecting students and learning: What is known and unknown. In L. Hobbs & G. Törner (Eds.), *Examining the phenomenon of "teaching out-of-field"*. Singapore: Springer. Retrieved from https://doi.org/10.1007/978-981-13-3366-8_7 doi: 10.1007/978-981-13-3366-8_7

Quille, K., & Bergin, S. (2018). Programming: Predicting student success early in cs1: A re-validation and replication study. In *Proceedings of the 23rd annual acm conference on innovation and technology in computer science education (iticse 2018)* (pp. 15–20). ACM. Retrieved from https://doi.org/10.1145/3197091.3197101 doi: 10.1145/3197091.3197101

Quille, K., & Bergin, S. (2019). CS1: How will they do? how can we help? a decade of research and practice. *Computer Science Education*, *29*(2-3), 254–282. Retrieved from https://doi.org/10.1080/08993408.2019.1612679 doi: 10.1080/08993408.2019.1612679

Quille, K., Bergin, S., & Mooney, A. (2015a). Programming: Factors that influence success revisited and expanded. In *Proceedings of the international conference on engaging pedagogy (icep 2015)*. Dublin, Ireland.

Quille, K., Bergin, S., & Mooney, A. (2015b). Programming: Factors that influence success revisited and expanded. In *Proceedings of the international conference on engaging pedagogy (icep 2015)*. Institute of Technology Blanchardstown.

Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. In *Proceedings of the 35th sigcse technical symposium on computer science education* (pp. 171–175). ACM. Retrieved from https://doi.org/10.1145/1007996.1008042 doi: 10.1145/1007996.1008042

Rosenberg, M. (1965). *Society and the adolescent self-image*. Princeton, NJ: Princeton University Press.

Rosenberg, M., Schooler, C., Schoenbach, C., & Rosenberg, F. (1995). Global self-esteem and specific self-esteem: Different concepts, different outcomes. *American Sociological Review*, *60*, 141–156.

Ryan, R. M., & Deci, E. L. (2000). Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist*, *55*(1), 68–78. Retrieved from https://doi.org/10.1037/0003-066X.55.1.68 doi: 10.1037/0003-066X.55.1.68

Sentance, S., Kirby, D., Quille, K., Cole, E., Crick, T., & Looker, N. (2022). Computing in school in the uk & ireland: A comparative study. In *Proceedings of the united kingdom and ireland computing education research conference (ukicer 2022)*. ACM. Retrieved from https://doi.org/

10.1145/3555009.3555015 doi: 10.1145/3555009.3555015

Sharplin, E. D. (2014). Reconceptualising out-of-field teaching: Experiences of rural teachers in western australia. *Educational Research*, *56*(1), 97–110. Retrieved from https://doi.org/10.1080/00131881.2013.874160 doi: 10.1080/00131881.2013.874160

Tabachnick, B. G., & Fidell, L. S. (2019). *Using multivariate statistics* (7th ed.). Pearson.

Valentine, J. C., DuBois, D. L., & Cooper, H. (2004). The relation between self-beliefs and academic achievement: A meta-analytic review. *Educational Psychologist*, *39*(2), 111–133. Retrieved from https://doi.org/10.1207/s15326985ep3902_3 doi: 10.1207/s15326985ep3902_3

Wheeley, E., Klieve, H., Park, E., & Du Plessis, A. (2023). Preservice teachers' perceptions about out-of-field teaching: Implications for students, teachers and schools. *Australian Journal of Teacher Education*, *48*(4). Retrieved from https://doi.org/10.14221/1835-517X.5619 doi: 10.14221/1835-517X.5619

Reversing the Mirror: The Moral Programming Language as Primary Notation for Agentic AI

Varad Vishwarupe
Department of Computer Science
University of Oxford.
varad.vishwarupe@cs.ox.ac.uk

Alan F. Blackwell
Department of Computer Science and Technology, University of Cambridge
afb21@cam.ac.uk

Abstract

Agentic AI, and the accompanying practice of vibe coding, is greatly increasing the productivity of professional programmers, but is also a powerful new approach to end-user programming: users give an LLM a natural-language specification and expect executable behaviour in return. We ask what should be considered when that behaviour carries moral weight, for example in smart homes or care contexts, and illustrate this argument with scenarios set in Mumbai and London. In conventional programming languages, the generated program may potentially be visible, inspectable and editable, but without any explicit basis to contest matters such as: the subject to whom a commitment belongs; a provenance that records who was persuaded and over whose objection; and any tensions that the program might be obliged to resolve without the family. We propose The MPL (the Moral Programming Language): a notation whose rendered form is the primary representation of the moral behaviour a user intends, by which the LLM is demoted from moral agent to authoring assistant, and a plain-language projection of its proposals is delivered to the people the system would govern.

Keywords: moral programming language, end-user programming, agentic AI, vibe coding, generated code, phronesis, attention investment, notation design, projection.

1. Introduction

A home assistant detects that an elderly woman in Mumbai, whom we will call Manya, has fallen. Her son, Harsh, lives in London. The home runs an off-the-shelf agentic platform of the kind now beginning to ship: a smart-home hub onto which third parties can deploy agents whose behaviour is configured through prompts and generated code. The platform must decide, in seconds, whether to notify her son, wait for her to respond, escalate to a neighbour, or do nothing. Each option is defensible. Each is also a moral act, not simply a technical one. Whose standing takes priority: the resident's autonomy, or the adult child's concern? Whose conception of the family does the platform encode? Who, at the moment of deployment, authored these commitments, and who can now see what those were?

It is tempting for advocates of autonomous AI agents to answer that last question by saying the commitments should be encoded in the weights of a network, perhaps derived from “constitutional” prompting and fine-tuning of the LLM that created the agent. However, we argue that the actual moral commitments are located in the agent script, perhaps a Python file, perhaps stored in a repository tha tany developer can open. The threshold at which the platform acts, the seconds it waits, and the summary it sends are all visible. But Python semantics include no direct record of the autonomous human agents who made the moral commitments. Who does the thirty-second wait belong to, who agreed to it and against whose objection, or perhaps that the family never resolved the underlying conflict at all?

This paper asks what it would mean to give those three things somewhere to live.

1.1 Agentic AI, vibe coding, and the latest end-user programming

The recent surge of interest in agentic AI supplements and extends a long-running conversation about the relationship between machine learning and end-user programming. The motivating use case for end-user programming is familiar: give non-programmers the ability to automate procedures for routine data processing tasks, by reference to data operations, as in the earlier collections of research edited by Lieberman (2001) and Cypher (1993).

What has changed is the inference technique used for synthesis. Rather than a feature-generalisation algorithm that infers a program from demonstration of the required results, agentic AI infers a program

from a natural-language prompt describing that result. Despite the rhetoric of agentic AI, it is important to note that LLM is not necessarily the runtime substrate. Just as with earlier machine learning methods described by Lieberman and Cypher, the LLM operates only at edit-time and at compile-time, generating agent code from prompts, with that code then running on a more conventional execution platform. What has not changed is a question that has motivated end-user programming research for three decades: when the resulting system acts on the user's behalf, what representation of that action does the user have access to, and what can they do with it?

Blackwell's (2001) usability critique of programming by demonstration made the analogy of fixing a loose part inside a clock by jostling the case. In familiar program synthesis systems such as Excel's FlashFill (2011), the mechanism itself is not visible. Current vibe-coding tools potentially mitigate this problem, and empirical studies of how vibe coding actually proceeds (Sarkar & Drosos, 2025; Pimenova et al., 2025) show developers reading and engaging with generated code to varying degrees. Vibe-coding programmers may read generated code less often than they *should*, but the code is available for inspection. However Sarkar (2023) asks directly whether code remains a relevant user interface for end-user programming in the generative era. This is our concern: when a synthesised artefact will be acting on someone's behalf, what representation of that action is available, to whom, and what can they do with it?

1.2 The mirror problem

In The AI Mirror, Shannon Vallor (2024) argues that generative AI systems simply reflect humanity's past judgments back at us and, in doing so, risk trapping us in a kind of moral narcissism. She argues that trying to find the "thoughts" behind an AI verdict is like looking for the body on the other side of your bathroom mirror. Constitutional AI, as developed by Bai et al. (2022), supplements the LLM training with explicitly authored moral guidance intended to correct predictions based on observed behaviour, but in Vallor's terms this polishes the mirror rather than removing it. In our own scenario, the constitution shapes the weights that generate the executable code, but the constitution itself does not execute.

Vallor's image suggests a failure of inspection: you cannot see what is behind the glass. But Manya does not need a clearer view of past behaviour. She needs a view of the commitments that will be acted on in future, and of the judgments being made on her behalf. Vallor argues that virtues have to be cultivated through practice. When the articulation, inspection and revision of moral commitments is delegated to a system that cannot perform any of those acts, the humans in the loop become deskilled in precisely those capacities that moral agency requires.

1.3 The inversion

We propose The MPL: a notation whose rendered form is the primary representation of moral behaviour, and in which the LLM is demoted to an authoring assistant. Under The MPL:

- **The user sees and edits a notation**, not a prompt. The notation expresses the moral conditions, priorities, and tensions that should govern the system's behaviour in a given domain.
- **The LLM helps author the notation.** Natural language is the input; the MPL is the output. The user may refine it, debate it with the model, and commit to a version.
- **The deterministic notation is what runs.** Behaviour follows the specification, interpreted by humans wherever the specification says so. The LLM is not the moral agent, and is not a runtime component of the moral loop.
- **Nobody is asked to read the notation who should not have to.** A plain-language projection, derived mechanically from the specification, is what the household reads and edits.

This is the inverse of the arrangement in explainable AI, where the neural network's representation does the executing and a visualisation is offered as a lower *notation layer* (Green, 1989) for human consumption. In end-user programming systems such as SWYN (Blackwell, 2001), and in the MPL, the user-facing notation occupies the upper layer and the learning machinery sits below it. (Green's term *notation layer* is distinct from *secondary notation*, which refers to comments and layout that have no

effect on execution.) It is also the inverse of Constitutional AI, where the constitution shapes a primary non-deterministic system from training-time. Rather than making network weights interpretable, we suggest that such weights should not be the primary bearer of moral intent in the first place.

We argue that the moral specification should live further down the pipeline. Constitutional AI places it at the level of how a network behaves in general. We place it at the local level of one deployed agent behaving in one household, which is the level at which the people affected can actually argue with it.

The specification is designed as a boundary object (Star & Griesemer, 1989) readable by multiple stakeholders: developers who deploy the platform, households whose lives it governs, and, where relevant, auditors and regulators, each through the representation appropriate to them. The MPL does not resolve the politics of whose commitments end up in any particular specification; it makes those politics visible.

1.4 Research questions

The paper is organised around one guiding question and three analytical sub-questions.

RQ. If moral specification were treated as the primary representation in agentic AI, with the LLM relegated to authoring assistance, what design space does this inversion open, and what are its trade-offs against locating the specification in generated code or in model weights?

RQ1 (phronesis). Can a deterministic moral notation support rather than replace the cultivation of practical wisdom, and in whom?

RQ2 (attention). How do the attentional demands of authoring and maintaining an MPL specification compare with those of reviewing generated agent code, when attention is understood not only as an economic but as a moral resource?

RQ3 (notational experience). Along which Patterns of User Experience, or Cognitive Dimensions, do a moral notation and a general-purpose programming language diverge as vehicles for moral commitment?

1.5 Contributions and roadmap

Our contributions are: (i) an identification of three things a generated program has nowhere to keep, and which a moral notation can; (ii) a formulation of The MPL as a relocation of moral specification down the pipeline, from network behaviour to program behaviour; (iii) the projection, a derived plain-language rendering that lets a household contest a specification without reading it; (iv) a separation of decision time from deliberation time, with authored abstention as the construct that crosses between them; and (v) a pair of scenarios that exhibit pluralism rather than claiming it. Section 2 reviews the relevant background briefly. Section 3 develops the comparison against generated code. Sections 4, 5, and 6 work through the three analytical pillars. Section 7 presents the paired scenarios. Section 8 discusses open questions. Section 9 sets out future work. Section 10 concludes.

2. Background

We touch five threads briefly.

End-user programming and program synthesis. Nardi (1993), Lieberman et al (2006), and Ko et al. (2011) frame end-user programming as a question of how non-programmers can specify automated behaviour. Programming by demonstration is a strategy in which a machine learning algorithm synthesises a program by observing desired outcomes (Cypher 1993). Blackwell (2001) argued for rendering a synthesised artefact in a notation the user can read and modify. Recent advances in "vibe coding" (program synthesis from natural language prompts) mean the artefact is already rendered, because the generated source is itself a rendering.

Empirical accounts of vibe coding. Sarkar and Drosos (2025) and Pimenova et al. (2025) provide qualitative accounts of how programming through conversation with a model proceeds in professional software development, including how developers read, trust and repair generated code. Sarkar (2023) questions whether code should remain the user interface for end-user programming at all. We therefore consider whether the household members in our scenarios should read code, or any notation, at all.

Cognitive Dimensions and Patterns of User Experience. Green (1989) and Green and Petre (1996) provide a vocabulary for comparing notational systems. Blackwell and Green (1998) extend the

framework with a model of how users make cost, risk, and payoff decisions when working within a notation. Blackwell and Fincher (2010) and Blackwell (2024) extend and generalise this approach in the Patterns of User Experience (PUX) framework. All three apply to the comparison between The MPL and the generated code it would displace.

Constitutional AI and deployment-time alignment. Bai et al. (2022) train models to conform to a written set of principles, operationalised as a weighting on generated outputs. Deployment-time alignment work, including red-teaming, RLHF fine-tuning cycles, and post-deployment monitoring as discussed by Terry et al. (2023), shares the structural feature that the specification governs the generator rather than the generated.

Vallor and phronesis. Vallor (2016, 2024) argues that AI systems risk a specific kind of moral damage: the deskilling of human practical wisdom through delegation. Her call for the cultivation of *phronomoi*, practitioners of practical wisdom, is the philosophical anchor of our argument, and Section 4.5 addresses which parties we take to be at risk.

We do not address the large literature on AI ethics and governance. Our claim is narrower: that notation matters to the users of moral AI, independently of how the tools are governed.

3. The Comparison: Generated Code and The MPL

3.1 Choosing the right comparator

To be clear about the technical focus of our usage scenario, we are not considering a smart-home where an autonomous but constitutionally-aligned AI "agent" responds to events in the house, making decisions on behalf of the resident and other stakeholders, with the necessary inference algorithms making moral decisions at the moment Manya falls. In our envisioned scenario, the "agentic" authoring system generated a program months earlier. It is this program that runs when Manya falls, and is available for a developer to read, inspect and edit in advance. Our concern is with the notational design of that generated program, not to compare the relative safety or desirability of scripted specification vs constitutionally-aligned probabilistic reaction. We ask to what extent an explicitly moral notation might support moral decisions by humans in a way that a program generated in a conventional language might not.

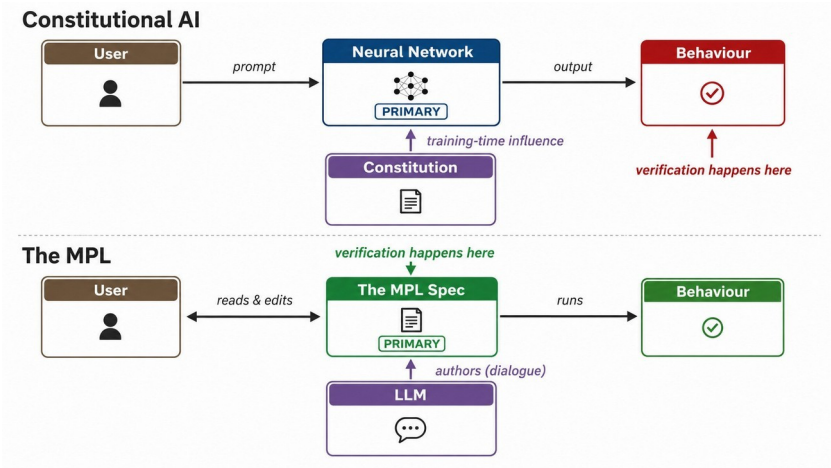


Figure 1. The comparison. A constitutionally-aligned model does not execute in this setting; it generates agent code at compile-time, and that code is what runs and what a developer can inspect. The MPL treats a rendered notation as the primary representation of moral intent, with the LLM as a secondary authoring assistant, and derives a plain-language projection for the household.

Dimension	Constitutional AI	Generated agent code
What the specification governs	How the network behaves in general	Nothing; the code is the behaviour, not a specification of it
Where moral intent is recorded	Training-time constitution; does not execute	Implicit in constants and control flow
Who can read it	In practice, nobody at deployment	Developers
Subject of a commitment	Absent	Absent
Provenance of a commitment	Absent	File history only
Unresolved tension	Cannot be expressed	Cannot survive compilation

Table 1. Where the moral specification lives. Constitutional AI specifies the generator; generated code specifies nothing but does everything.

The MPL specifies the generated behaviour at the level of the household it governs, and gives a commitment a subject, a provenance and a place to leave a tension standing.

3.2 What the generated program already gives us, and what it cannot

Consider what the platform in our Mumbai scenario is actually running. The code below is representative of what an agent-authoring model produces for a fall-response behaviour, and it is committed to a repository the deployer controls. The argument depends on conceding how much it already does, so it is worth being generous about it. Every moral decision the household will be subject to is present on that page: the confidence threshold is a judgement about how sure the house must be before it acts on Manya at all; the grace period is her autonomy, expressed as an integer; the early return when she responds is a grant of standing over her own disclosure; the summary level decides how much her son is told; and the bare return in the first branch is a decision to do nothing, indistinguishable on the page from an oversight.

And yet three things a family would argue about have nowhere to sit in it. First, a subject. GRACE_PERIOD_S = 30 is a fact about the program, not a commitment held by anyone, and no field says whose autonomy the thirty seconds expresses. One can add a comment naming Manya, but a comment is not a binding site: nothing connects it to the constant, nothing updates it when the constant changes, and nothing notices when it becomes false. Second, an explicit provenance. The constant has a git history, and the history is informative about who edited a file and when, but it does not record whether anyone was persuaded, over whose objection the change was made, or whether the person whose autonomy is at stake agreed. Version control tracks authorship of text; moral provenance is authorship of commitment, and the two coincide only by accident. Third, an unresolved tension. A program must resolve, because the compiled artefact has to do one thing. The program is not a black box; it is simply a place to keep a commitment.

If we relate this to professional software development practices, we acknowledge that a sufficiently disciplined team could keep all three in the repository: a decision log, an ADR-style record, structured comments, a CODEOWNERS file naming who signs off on which constant. We accept that this is possible, and note that it is exactly what such conventions are for. Two things follow. First, that these are conventions and not constructs is the point: nothing in the language binds them to the behaviour, and their reliable decay is a well-attested feature of software practice rather than a failure of diligence. Second, a team that adopted all of them thoroughly would have built something with the properties we are describing, at which point the disagreement is about whether it deserves to be designed deliberately instead of assembled by convention.

A second objection is that generating a notation the developer can see and edit is what agentic coding tools already do. That is correct, and it is why our contribution cannot be the rendering. The generated code is already rendered. The contribution has to be that a different rendering, with different commitments about what it must hold, is a better object for multiple stakeholders to exercise moral judgement over. That is a claim about notation design.

3.3 Why the relocation is defensible

Three arguments support the relocation. First, morally consequential action should be traceable to a commitment, not sampled from a distribution and not left implicit in a constant. A system may reasonably be probabilistic in recognising that a fall has occurred; the decision about whether to disclose that fall should follow from a commitment someone has authored and can point to. Second, the locus of moral authority should be visible to the people it governs, and a generated program is visible to developers alone. Third, moral cognition requires exercise. Following Vallor (2024), we take seriously the claim that users who never articulate moral commitments lose the capacity to do so.

We acknowledge that these arguments are not decisive. They can be resisted in at least three ways: via the claim that everyday moral life is not deterministic and a deterministic notation would distort it, which is the rigidity objection addressed in Section 4; via the claim that the cognitive cost of authoring a notation is prohibitive, which is the attention objection addressed in Section 5; and via the claim that no sufficiently expressive notation is practical, which is the notational-experience objection addressed in Section 6. The rest of the paper responds to each in turn.

4. Phronesis: Does a Deterministic Notation Kill Practical Wisdom?

4.1 The rigidity objection

The most natural objection to The MPL is that morality is not a rule system. The weak form of the objection says a notation would try to legislate phronesis out of existence. However, the MPL does not ask the notation to perform practical wisdom. The strong form is that if morality is not a rule system, then turning a moral commitment into an executable specification may be a category-mistake. If the content of the specification is not morality, what is?

Our answer is that an MPL scene does not capture Manya's morality. It captures the moral positions her platform is going to take anyway, whether or not anyone writes them down, and it does so at the granularity at which they can be contested. The thirty-second window is not her view of autonomy; it is the operational residue of that view, which the platform will act on. The choice is not between capturing morality and leaving it uncaptured. It is between a residue that names its holder and one that does not.

4.2 The electrician and the switch

Consider an analogy from a less abstract craft. An electrician working on a domestic circuit needs the switch in front of them to be deterministic: throw it, and a known thing happens. The switch is not the electrician's expertise; the expertise is in knowing where the switch belongs in the circuit, when to bypass it, when to install a different kind of switch, and when the wiring behind it makes the switch the wrong instrument altogether. Without a deterministic switch, none of that judgment has anywhere to apply. With it, the electrician's craft is the act of placing the right switch, in the right place, for the right load. The deterministic substrate is not the enemy of practical wisdom; it is the precondition of it.

The MPL proposes the same arrangement for moral specification in agentic platforms. The notation expresses principles, priorities, and tensions; this is, deliberately, the opposite of what most programming languages do. A conventional programming language compresses meaning into instructions for the machine and pushes interpretation away from the reader. The MPL refuses that compression: interpretation is preserved as a first-class structural feature, made visible to the household and the deployer rather than collapsed into control flow. Interpretation happens at deployment, by humans in context and, where the notation allows, by humans plus an LLM operating as an interpretive aide rather than as a moral authority. The deterministic specification is the substrate on which judgment operates, not a substitute for it.

4.3 Moving the specification down the pipeline

It would be wrong to say that Constitutional AI involves no moral specification. The constitution is a moral specification, written, explicit and often made public to some extent (noting that the moral operation of an LLM is also determined by guardrails and commercial policies that operate alongside the constitution, and that may not be public). The difficulty is the level at which the constitution operates: it governs how a model behaves in general, and therefore how it is disposed to generate programs, rather than what any particular generated program does in any particular household. Between the constitution and Manya's flat there are generation steps not encoded in any constitutional clause.

Our proposal is best understood as moving the moral specification down the pipeline: from the behaviour of the network that writes programs to the behaviour of the programs themselves. Stated this way the two approaches are complementary, not rival. A constitution can reasonably govern what a model will help you write; it cannot record what your household decided it owed one another, because it was written before your household existed.

4.4 Two clocks, and whose phronesis is at stake

The claim that a notation must admit an interpretation layer at runtime conflates two different things, and we separate them here because the conflation invites a fair objection: nobody is going to edit a specification during the fifteen seconds in which Manya is on the floor. Decision time is the fifteen seconds. The specification executes, nobody interprets, edits or is consulted, and determinism is the entire point, because at this timescale the realistic alternative to a specification is not human judgement but a sampled distribution. In contrast, the time for deliberation among members of Manya's family and support community may run from hours to years. The specification is read aloud, argued over and amended: after a diagnosis, after a fall, when a grandchild moves in, when the household's sense of what it owes has shifted. This is where interpretation lives and where practical wisdom gets its exercise.

We propose one construct that crosses between those clock times. Authored abstention lets a specification name, in advance and in slow time, the cases it refuses to decide in fast time, and route those to a named human. A clause of the form: if Manya's cognitive state is in doubt, do not act on this scene, ask Harsh, is a commitment made deliberately and honoured instantaneously. An interpretation layer is not intended for editing during an emergency, but pre-committing to the boundary of the specification's own authority.

This separation highlights the question of who is supposed to cultivate practical wisdom. The scenarios open onto a wide cast that includes Manya, her son, the deployer and an auditor. A notation does not do the same thing for all of them. Our claim is narrower. The capacity at risk is the capacity to articulate a moral commitment precisely enough to be bound by it, and it is at risk for whoever would otherwise have delegated that articulation to a model. In the arrangement we describe that is principally the developer or deployer who authors a scene and must answer for it. Manya and Harsh exercise something related but distinct: not articulation but contestation, the capacity to read a commitment made about your life and say that it is wrong.

Phronesis is different from accountability. Accountability buys institutional traction, in the form of audits, duties and someone to answer, but it attaches to records instead of to people, and one can be perfectly accountable without growing any wiser. Judgment is plain and needs no translation, but says nothing about what happens to a capacity when the practice that sustains it stops. Moral articulacy is one way to describe the capacity the notation exercises. Phronesis, in Vallor's work, relates to the threat of moral deskilling, as a capacity that is built by practice and lost through disuse. In order to support it, we propose three design commitments:

- **The notation must represent tensions, not only rules.** Conflicts between values, such as autonomy versus safety, or privacy versus familial concern, should be first-class citizens, not resolved away at authoring time.
- **The notation must support authored abstention.** Not every case can be decided by specification. The notation must let the author name, in deliberation time, the cases it will not decide in decision time, and route them to a named human.
- **The notation must be re-authorable.** Moral commitments evolve. A specification that cannot be edited by the community it governs is a different kind of tyranny.

5. Attention as a Moral Resource

5.1 Attention investment

Blackwell and Green (1998) analyse notational interaction as an investment of attention under uncertainty. Users allocate attention according to estimates of cost, risk, and expected payoff. Notational designs are evaluated in terms of their attentional profile: how much is required to make progress, to reduce cost, to reduce risk.

5.2 The moral reframing

In recent work, Blackwell (2024) has reframed attention investment as a moral resource. The argument starts from a simple observation: a human life contains on the order of a billion seconds of consciousness. Engagements with technology that consume a user's attention represent fractions of a mortal life. Blackwell proposes the "nano-soul", as one-billionth of a conscious life, for use as a unit of moral accounting in the design of user interfaces, social media platforms, and conversational AI systems.

One second of a user's time is a second of their life whatever they spend it on. The moral imperative of Vallor's phronesis can be related to classical theories of virtue from many cultures and wisdom traditions, in which the conscious attention of the individual human agent, their intersubjective participation in communities of care, and their reflection on these things is the basis of their own personhood.

5.3 Attentional profiles compared

Table 2 summarises the attentional profile of each architecture. The profiles are structurally different, not just quantitatively different.

Attention aspect	Reviewing generated code	The MPL
Authoring	Low; the user writes a prompt	High; the user works with the LLM to articulate, read, and commit a specification
Verification	Recurrent; each regeneration invites re-reading	Concentrated; inspection at authoring time and on edits
Repair burden	An edit to code, or a re-prompt	Distributed; repair is an edit to the notation
Failure mode	Silent drift between what the code does and what anyone believes it does	Authored misalignment; visible and contestable
What the time buys	*Knowledge of what the program does*	*A commitment someone is bound by*

Table 2. Attentional profiles as moral commitments.

5.4 The comparison as a moral question

Which profile is preferable is not a matter of aggregate attention alone. A system may cost the user more total attention and still be preferable on moral grounds, if the attention it consumes is of a kind that supports moral development. Reviewing generated code may consume attention in ways that do not build moral capacity in the reader, where there is no commitment to hold or contest. The alternative architectures of agentic versus end-user place the expenditure at different points: concentrated and prospective under the MPL, distributed and retrospective otherwise. This difference is worth measuring.

6. Patterns of User Experience: A Notational Comparison

6.1 Two notations, side by side

Current agentic AI systems encode the specifications that they derive in conventional general purpose programming languages such as Python, Rust or Typescript. (At the time of writing, these are human-readable, although there is considerable work in progress to refine general purpose programming languages for use by LLM agents, removing the expectation that these should be human-readable at all, placing greater emphasis on token efficiency or reducing semantic expressibility to more verifiable semantic subsets). We therefore compare our MPL proposal to general purpose languages.

Dimension	General-purpose language (generated code)	The MPL
Visibility	High for the developer; the source is the artefact	High for the developer; the projection carries it to other readers
Role-expressiveness	Low for moral role: a constant does not announce whose commitment it is	High: clauses attach to named parties
Hidden dependencies	A comment and the constant it describes are unbound and drift silently	Interpretation notes are bound to the block they annotate
Viscosity	Low for a constant; high for a commitment spread across call sites	Currently untested; likely a weakness, since one commitment may touch several clauses
Premature commitment	Moderate	Likely a weakness: the scene must be enumerated before its cases are known
Provisionality	Committed; the artefact must resolve every conflict	Committed for a scene, with unresolved tensions retained

Table 3. A notation-to-notation comparison showing tradeoffs for the MPL.

6.2 Activity profile: deliberation as the distinguishing activity

Cognitive Dimensions focuses on the process steps in different kinds of interaction with a notation; PUX shifts that perspective, asking what subjective experiences the user might have when engaged in notational activities. Across the patterns elaborated by Blackwell (2024) - exploration, transcription, modification, search, exploratory design, incrementation - one activity stands out as the natural home of phronesis: deliberation. Phronesis, applied to a moral notation, looks like reading a specification slowly, noticing a tension between two values it names, considering whether a proposed edit honours both, and committing to a revision the household will live with.

Table 4 contrasts the activity profile encouraged by a general-purpose language with the activity profile encouraged by The MPL. The shift is not just in which dimensions of a single notation are exercised, but in what kind of activity the artefact invites the reader into in the first place.

Activity (PUX)	Reading generated code	Under The MPL
Exploration	Tracing control flow to infer what the system will do	Reading the specification; tracing how a clause shapes a scene
Modification	Editing a constant; the change persists but carries no account of itself	Editing the specification; the change persists and is legible
Deliberation	*Largely absent; the artefact presents no tension to weigh, having resolved it at compile time*	*Central; tensions and interpretation notes invite the user to weigh, commit, revise*
Verification	Recurrent, on each regeneration; nothing states what the code was supposed to commit to	Concentrated, on the specification before deployment
Dominant mode	*Inspection without authorship*	*Authorship and stewardship*

Table 4. PUX activity profile. Reading generated code positions the reader as an inspector; The MPL positions them as an author.

7. Two Scenarios: Mumbai and London

We make the comparison concrete with a pair of scenarios, one set in Mumbai and one in London. Both describe a fall event; each requires the system to act within seconds; each depends on different moral

commitments; each produces a different MPL specification. They are illustrative vignettes, drawing on the first author's prior fieldwork on smart-home AI in Indian households (Vishwarupe et al., 2026a) and on household structures common in the UK care literature. Neither is a recorded case; both are intended to be concrete enough for design discussion.

A note on the notation. We present The MPL as a block-structured declarative syntax. A block-structured form makes scope visible, which is what a scene construct needs; a declarative form keeps the reader on conditions, commitments and tensions rather than control flow. Two properties distinguish it from declarative policy languages and configuration formats: unresolved tensions are first-class citizens rather than resolved silently at authoring time, and natural-language interpretation notes have a reserved structural position rather than being relegated to comments. We do not claim that every member of the household necessarily reads this, or does so without assistance.

7.1 Scenario A: Mumbai



Figure 2. Scenario A, Mumbai. Manya, an elderly widow living alone, has fallen. Her only son lives in London. She has articulated a preference for autonomy over being watched-over. The platform has fifteen seconds to decide.

An AI-enabled assistant in a single-resident household in Mumbai detects a fall event. The resident, Manya, is a 72-year-old widow; her only son lives in London. The family has discussed, in general terms, that the platform should help if something happens. No one has specified what help means. Cultural expectations within the family favour rapid notification of adult children in medical events; Manya herself values her autonomy and has expressed discomfort at being watched over. Within the fifteen seconds after the fall detector triggers, the platform must act.

```
scene "Fall disclosure - Mumbai household" { resident: MANYA
  notifiable: [adult_child_1] default_action: none

  on fall_detected {
    grant MANYA window(30s) { action: resident_override(none) }
    after window {
      disclose_to adult_child_1 with summary.medical.minimal
    }
  }

  tensions {
    autonomy(MANYA) vs safety(MANYA)
    privacy(MANYA) vs familial_concern(adult_child_1)
  }

  interpretation_notes {
    "Manya has articulated discomfort with being watched; window
     gives her standing priority over disclosure. Tension is not
```

```
        resolved here; it is made legible to the family and the
        deployer."
    }
}
```

Figure 3. Sketch of possible MPL syntax for the motivating scenario.

To a 72-year-old widow in Mumbai the MPL syntax of Figure 3 is likely to be gobbledegook. Notational design suggests a projection derived mechanically from this specification: three sentences telling her that the house will wait thirty seconds for her to say she is alright, that it will otherwise tell Harsh, and that it will tell him only that she fell and did not answer; three controls, to change the wait, change who is told, or ask her first; and a line of provenance recording that this was agreed in the family conversation of 4 March, last changed by Harsh on 12 June, and that she can change it back.

Note that the direction of derivation is reversed by comparison to agent-based coding, in which the natural language specification is used to create the executable code. Here, the executable code is used to recreate the projection. Somebody decided what the code should say. This gives our notation a design premise: if any clause cannot be rendered as a sentence its subject could contest, it does not belong in the notation.

A note on the design decisions visible here. Four choices in the sketch are worth drawing out in PUX terms, because they are not accidents of presentation but commitments about how a moral notation should work.

Tensions as first-class citizens (role-expressiveness; anti-flattening). The `tensions` block is not a derived artefact of the rules; it sits alongside them with equal standing. A PUX analysis of rule-based systems identifies a recurrent failure mode in which conflicts between values are resolved silently at authoring time and become invisible to the reader. The MPL sketch refuses this: the tensions persist in the notation as acknowledged, unresolved pulls. This is a direct response to the rigidity objection of Section 4, the notation explicitly admits that some of its content is not algorithmic.

Interpretation notes as secondary notation (visibility; closeness of mapping). The `interpretation_notes` field is a designated home for natural-language commentary that does not execute but that any reader must see when reading the specification. In Cognitive Dimensions terms this is secondary notation with enforced juxtaposability: the notes cannot be separated from the block they annotate. The design choice reflects that moral specifications are communicative artefacts as much as computational ones.

Named override as explicit control (hidden dependencies). The `grant MANYA window(30s)` clause makes Manya's priority over her own disclosure a visible, editable fact. A naïve version of the same behaviour could have been achieved by a silent default buried in the runtime. The explicit form is more verbose, a cost, but eliminates the hidden dependency that Green's framework flags as the single most reliable producer of user error.

Bounded scope as a provisionality lever. The `scene` construct names a bounded situation. It is a design commitment that the family does not author a global moral policy; they author a specification for this scene. Other scenes, such as medication reminders, visitor admission, and purchase authorisation, will have their own specifications. This makes the specification provisional in the Cognitive Dimensions sense: committed for this scene, revisable independently of others, and cognitively tractable because its scope is local.

7.2 Scenario B: London

To make the pluralism argument of Section 8 concrete, we offer a contrasting scenario. Margaret (79) lives in London with her daughter, son-in-law, and their two teenage children. Margaret has been diagnosed with mild cognitive impairment; the family are her primary caregivers. Two years ago, before her decline, Margaret authored an advance directive agreeing that during periods of cognitive impairment her standing priority over disclosure should not apply. The family's smart-home assistant runs the same MPL runtime as one would run in Mumbai; the scene specification for fall events, however, is substantively different.



Figure 4. Scenario B, London. Margaret, 79, lives with her daughter and son-in-law and their two teenage children. A diagnosis of mild cognitive impairment and an advance directive she authored in 2024 have reshaped the family's moral arrangement.

```
scene "Fall disclosure - London household" {
  resident: M                 // Margaret
  caregivers: [C1, C2]        // daughter and son-in-law
  co_residents: [C3, C4]      // teenage children
  default_action:
  immediate_minimal_disclosure on
  fall_detected {
    disclose_to [C1, C2] with
    summary.medical.full disclose_to [C3,
    C4] with summary.status_only if
    responsiveness.M == low {
      escalate emergency_services
    }
  }

  tensions {
    dignity(M) vs caregiver_coordination
    transparency_to_co_residents vs medical_privacy(M)
  }

  interpretation_notes {
    "No resident-override window is specified here. M's diagnosis means
     self-override may not be reliable at the moment of an event. This
     deviation from the household's usual priority for autonomy is
     authorised by M's advance directive (dated 2024), which the family
     can revoke or amend at any time. The co-residents receive status-
     only summaries because the family has decided teenage children
     should be informed but not handed medical detail."
  }
}
```

Figure 5. MPL specification for Scenario B.

The specifications differ from scenario A along every salient axis: the default action, the presence or absence of an override window, the disclosure targets and granularities, the tensions named, and the provenance of the commitments - a lifetime of conversation between Manya and her son, a signed advance directive from Margaret. What the two scenes share is the substrate. Both are legible to the households they govern. Both are editable. They reflect different family arrangements rather than different values trained into a single model; both are articulations of a household's commitments for a specific situation.

We can contrast the use of a textual syntax to specify these two configurations with a conventional user

interface that might offer a configuration panel to specify the different values. A configuration panel provides a set of options that someone else anticipated, and offers no record of how they were prioritized or contested. Our two scenarios encode those tensions the household considers live, and which party may revoke the arrangement. Neither is a setting, because neither is a value chosen from a range. Margaret's household did not select dignity versus caregiver coordination from a dropdown. They arrived at it, and the notation gave it somewhere to be recorded.

7.3 What the comparison shows

Table 5 summarises what each architecture makes available to the household across three dimensions -the commitments the system is held to, the decision it makes at the moment of acting, and the scope of what the family can change.

What's at stake	Generated agent code	The MPL
Where commitments live	Distributed across constants and control flow, unnamed	Named clauses attached to named parties
What the household can read	Nothing, realistically; or a panel written about the code	A projection derived from what executes
Record of a disagreement	A commit message, written by whoever was on call	A dated, attributed entry over the commitment itself

Table 5. The same scenario under the two architectures.

One illustration makes the last row concrete. Manya asks for three minutes instead of thirty seconds, and her son agrees after a conversation involving her doctor. In the generated program this is a one-character edit to a constant, in a file the household will never open, with a commit message written by whoever was on call that week. Under The MPL it is an entry over the commitment: dated, attributed to Harsh, carrying his account of why his mother was right, and reversible by her. Both change the behaviour identically. Only one of them is a record a family could later be shown. Three observations follow.

- Under The MPL, the moral commitments of the system are locatable, and belong to somebody. In generated code they are present but unattributed.
- Under The MPL, the attention required to understand what the system will do is concentrated at authoring time. Reading generated code distributes it across every regeneration.
- Under The MPL, the family's disagreements, and the differences between households, become arguments over a specification that records who won and why. In generated code they become arguments over a commit.

8. Open Questions

8.1 Pluralism

The ambition of a single moral notation that works across cultural contexts is suspect on its face. Blackwell’s planned work on programming language design drawing on Yoruba and Maori philosophical traditions (Blackwell 2026) suggests that the very structure of a moral notation may not travel without translation. A concrete instance sits in the second line of every scene above: The MPL takes an individual as the unit of moral standing, which is a substantive commitment and not obviously the right one where the household or the lineage is the morally salient unit. We do not know what a collective principal would mean for a construct like grant, and regard finding out as among the more interesting things this proposal has opened.

8.2 Expressiveness

Whether any notation can do what we are asking remains the deepest open question. Everyday moral life is full of tensions that resist articulation, and articulating them often flattens them. It may be that the most important moral work in a smart-home scenario is irreducibly in the interpretation and no MPL can hold it. We would rather have this falsified than defended: a moral commitment that cannot be rendered as a sentence its subject could contest would bound the proposal usefully.

8.3 Whether the projection can be edited

A third question concerns the projection itself. Section 7.1 suggests that Manya might edit a projection card and the edit lands on the clause. For the three controls shown this is straightforward, because each maps to a single parameter. It is much less clear that an arbitrary sentence in a projection can be edited unambiguously back into a specification. If the projection turns out to be readable but not editable, then The MPL is a better explanation of a system than a primary notation for it.

9. Future Work

The present paper is a position and a research sketch. Three strands of follow-up work are required to move from sketch to evaluation.

The projection, and whether it is bidirectional. If an end user never reads The MPL, the end-user-facing research question is not the syntax of the notational layer in our figures, but the projection's fidelity and editability: what can be rendered as a contestable sentence, what an edit to such a sentence can unambiguously mean, and where the mapping breaks down.

Syntax and semantics, scoped as developer-facing infrastructure. A full syntax for The MPL, together with a formal semantics for authored abstention, remains necessary, but we want to be explicit that this is not a proposal to teach everybody in a household a programming language. The notation is authored with model assistance and read through projection; its design constraints involve interchange between the runtime, the projection and the audit trail.

Authoring tools and runtime. How does a family converse with a model to produce a scene specification they trust, when the family sees only the projection and the model writes the notation? What does the authoring interface look like, what are the failure modes of that loop, and how would anyone notice a model that drafted a plausible card over an unfaithful clause?

Empirical evaluation. No user study is reported here, and RQ1 in particular cannot be settled without one. Comparative evaluation against a baseline in which stakeholders are given generated code plus a conventional settings panel, instead of against a strawman of opaque weights, is the study that would test the claim, drawing on the mixed-methods approach of Vishwarupe et al. (2026a, 2026b, 2026c).

10. Conclusion

We have argued for relocating moral specification. Contemporary alignment work places it at the level of the network that generates programs; we have proposed placing it at the level of the programs themselves, in a notation authored with the model's help but not governed by the model, and rendered for each party in the form that party can use. Our argument rests on the fact that a commitment belongs to someone, was arrived at somehow, and may never have been settled at all. An executable artefact can carry none of that. When moral commitments live in the model weights of constitutional AI, no family authors them, no community contests them, no practitioner grows wise through their articulation. When they live in a notation, they become objects of human craft: read aloud at the kitchen table, argued over, revised when a new child is born or a parent falls ill. We draw on analytic perspectives familiar in Psychology of Programming: Cognitive Dimensions, Attention Investment, and Patterns of User Experience, to support a programming language in which the authorship of moral systems is returned to the humans they govern, read by those they affect, and revised when the world changes. Vallor asked us not to look for a body behind the mirror. We are proposing, instead, that we place the body in front of it, legible, editable, and ours, and write, together, the ethics that the next generation of AI will actually follow.

Acknowledgements

Thank you to Advait Sarkar, whose questions about phronesis and about the two senses of runtime shaped Sections 4.4 and 4.5. We also thank the households in Mumbai and London who let us watch them argue about this.

References

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A.,

McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., ... Kaplan, J. (2022). Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*.

Blackwell, A. F. (2001). SWYN: A visual representation for regular expressions. In H. Lieberman (Ed.), *Your wish is my command: Programming by example* (pp. 245-270). Morgan Kaufmann.

Blackwell, A. F. (2024). *Moral codes: Designing alternatives to AI*. MIT Press.

Blackwell, A.F. (2026). Conversations with, among, and about machines. Workshop paper presented at *Science in the Forest, Science in the Past* V. Available at https://www.cl.cam.ac.uk/~afb21/publications/SFSP-V-Blackwell.pdf

Blackwell, A.F. & Fincher, S. (2010). PUX: Patterns of User Experience. interactions 17(2), 27-31

Blackwell, A. F., & Green, T. R. G. (1998). Investment of attention as an analytic approach to cognitive dimensions. In *Proceedings of the 10th Annual Workshop of the Psychology of Programming Interest Group (PPIG 1998)* (pp. 24-35).

Cypher, A. (Ed) (1993). *Watch what I do: programming by demonstration*. Cambridge MA: MIT press.

Green, T. R. G. (1989). Cognitive dimensions of notations. In A. Sutcliffe & L. Macaulay (Eds.), *People and Computers V* (pp. 443-460). Cambridge University Press.

Green, T. R. G., & Petre, M. (1996). Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of Visual Languages and Computing, 7*(2), 131-174.

Gulwani, S. (2011). Automating string processing in spreadsheets using input-output examples. ACM Sigplan Notices, 46(1), 317-330.

Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M. B., Rothermel, G., Shaw, M., & Wiedenbeck, S. (2011). The state of the art in end-user software engineering. *ACM Computing Surveys, 43*(3), 1-44.

Lieberman, H. (Ed.). (2001). *Your wish is my command: Programming by example*. Morgan Kaufmann.

Lieberman, H., Paternò, F., Klann, M., & Wulf, V. (2006). End-user development: An emerging paradigm. In End user development (pp. 1-8). Dordrecht: Springer Netherlands.

Nardi, B. A. (1993). *A small matter of programming: Perspectives on end user computing*. MIT Press.

Pimenova, V., Fakhoury, S., Bird, C., Storey, M. A., & Endres, M. (2025). Good vibrations? A qualitative study of co-creation, communication, flow, and trust in vibe coding. arXiv preprint arXiv:2509.12491.

Sarkar, A. (2023). Will code remain a relevant user interface for end-user programming with generative AI models? In Proceedings of the 2023 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (pp. 153-167).

Sarkar, A., & Drosos, I. (2025). Vibe coding: Programming through conversation with artificial intelligence. In Proceedings of the 36th Annual Conference of the Psychology of Programming Interest Group (PPIG 2025).

Star, S. L., & Griesemer, J. R. (1989). Institutional ecology, 'translations' and boundary objects: Amateurs and professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science, 19*(3), 387-420.

Terry, M., Kulkarni, C., Wattenberg, M., Dixon, L., & Morris, M. R. (2023). Interactive AI alignment: Specification, process, and evaluation alignment. *arXiv preprint arXiv:2311.00710*.

Vallor, S. (2016). *Technology and the virtues: A philosophical guide to a future worth wanting*. Oxford University Press.

Vallor, S. (2024). *The AI mirror: How to reclaim our humanity in an age of machine thinking*. Oxford University Press.

Vishwarupe, V., Jirotka, M., Shadbolt, N., & Flechais, I. (2026a). Expectation management in smart home AI: A qualitative study in Indian households. In *Proceedings of HCII 2026*.

Vishwarupe, V., Flechais, I., Shadbolt, N., & Jirotka, M. (2026b). 'To LLM, or not to LLM?': How designers and developers navigate LLMs as tools or teammates. In *Extended Abstracts of the 2026 CHI Conference on Human Factors in Computing Systems*.

Vishwarupe, V., Jirotka, M., Shadbolt, N., & Flechais, I. (2026c). The collaboration gap in human-AI work: Grounding and repair conditions for stable collaboration. In *Proceedings of the 24th EUSSET Conference on Computer-Supported Cooperative Work (ECSCW) - Posters and Demos*.

Designing Purpose-first Instruction for Supporting End-user Programmer Learning Goals

Mehmet Arif Demirtaş
University of Illinois Urbana-Champaign
mad16@illinois.edu

Abstract

Computing knowledge is valuable to people besides just software developers. People from various disciplines use computing to create programs that support their personal interests and address domain-specific tasks. These learners, often called *end-user programmers*, are most interested in learning about the goals that can be achieved with programming and creating programs that satisfy these goals using high-level tools. In contrast, conventional computing education materials prioritize writing code from scratch by emphasizing data structures, language syntax, and software development practices. As a result, computing education fails to align with the needs of end-user programmers.

My research aims to improve computing education for end-user programmers through two synergistic goals. First, I explore the effectiveness of instructional materials that emphasize the purpose of code with students who may be interested in becoming end-user programmers. Second, I work with instructors to understand how they teach computing in diverse disciplines. Informed by instructor values and practices, I plan to design systems for creating instructional material aimed at end-user programmers.

1. Introduction and Background

Computing can be valuable for everyone: from artists (Chung & Guo, 2024) to scientists (O'Brien, 2025), professionals across various disciplines use computing in ways that differ both from each other and from the conventional learning goals in CS courses. These professionals are described as *end-user programmers*, as they develop programs for goals that support personal interests or tasks in their disciplines (Ko et al., 2011). End-user programmers often use opportunistic programming strategies (Brandt, Guo, Lewenstein, & Klemmer, 2008), such as following trial-and-error strategies, modifying similar code examples instead of writing code from scratch, and implementing functionalities without focusing on maintainability and scalability. These programmers are interested in using programming to create artifacts with complex functionality rather than studying theoretical knowledge that might be necessary for building robust and scalable programs (Ali & Dasgupta, 2025).

However, existing computing instruction values software development endpoints over end-user programming practices. Formal instructional resources are designed with minimal input from students and educators working with computing in non-CS contexts, and emphasize syntax over applications (Becker & Fitzpatrick, 2019). Some computing education researchers argue that top-down approaches that emphasize the purpose of the code at a high level are less effective for learning, compared to bottom-up approaches that focus on fundamental skill development (Thorgeirsson, Ewen, & Su, 2025). This is also reflected by end-user programmers' preference for informal learning environments, such as online resources, over formal instructional materials (Dorn & Guzdial, 2010).

In my work, I aim to improve computing education for end-user programming goals by investigating two main research questions:

- **RQ1**: What instructional techniques help students who are interested in end-user programmer learning goals?
- **RQ2**: What practices do instructors in various disciplines employ for supporting end-user programmer learning goals?

I explore the first research question through classroom studies with non-CS majors, who are likely to pursue end-user programmer learning goals (Hur & Cunningham, 2024). I explore the second research question through human-centered design projects with instructors to learn from practices they have adopted and design systems that can spread these practices to larger audiences.

2. Research Progress

2.1. RQ1: Evaluating instructional techniques for end-user programmer learning goals

2.1.1. Supporting students with purpose-first tutorials (SIGCSE 2026)

In this work (Demirtas, Zheng, & Cunningham, 2026), we built on *purpose-first programming* (Cunningham, Ericson, Agrawal Bejarano, & Guzdial, 2021) to design three programming tutorials for non-major students. These tutorials (covering SQL for database queries, Python for web scraping, and Python for HTTP requests) were developed for students with minimal programming background.

After using these tutorials in the hands-on lab sessions for an introductory programming course, we observed that students felt significantly more confident in their ability to use programming for domain-specific tasks, as shown by comparisons of pre/post surveys. Moreover, students were able to discuss potential use cases of each programming tool and discuss challenges and tradeoffs of these tools in their open-ended responses to several activities. Our findings suggested that instruction that focuses on code's *purpose* can be effective in courses with non-major students who are aligned with end-user programmer learning goals, especially for supporting affective outcomes and for building conceptual understanding.

2.1.2. Comparing top-down and bottom-up approaches to computing instruction (in progress)

While our first study showed promising results for certain affective outcomes, we noted that similar top-down approaches that prioritize code's purpose have been hypothesized to be ineffective for learning in recent literature (Thorgeirsson et al., 2025). In our ongoing work, we investigate this further by comparing a purpose-first tutorial on a data visualization library (Seaborn) as an example of top-down instruction to a popular online tutorial covering the same library through a bottom-up approach. The purpose-first tutorial is organized so that students start by seeing examples of authentic problems that are solved with the data visualization library, which are then broken down into subgoals and functions. In contrast, the bottom-up tutorial covers each function subsequently. Our objective was to compare whether there were any differences in learning outcomes between students who were randomly assigned to complete one of two tutorials. Our preliminary findings indicate that students from both groups performed on a statistically identical level according to equivalence testing on assessments of coding ability, syntax knowledge, and conceptual understanding. Our initial results indicate that top-down instruction can be as effective as bottom-up instruction in the context of learning a new library, while also having a higher alignment with what end-user programmers look for in learning resources.

2.2. RQ2: Learning from practices employed by instructors

2.2.1. Supporting instructors as they create inclusive computing instruction (CHI 2025)

Our work with students demonstrates the potential value of authentic examples in instruction. However, curating these examples can be time-consuming for instructors. In this work (Jain, Demirtas, & Cunningham, 2025), we designed a system in conversation with computing instructors to alleviate the workload of finding examples of authentic applications for various programming libraries.

Our design process started with interviews with computing instructors. We observed that their workflows can be improved through automation: to find authentic programming applications, they needed to collect examples across many sources, identify code lines common between examples, and interpret and decide which examples would be useful for students. We proposed an AI-supported workflow to identify common code lines so that instructors could focus their expertise on interpreting and refining the generated draft for students. We explored this hypothesis through design workshops with seven computing instructors, revealing design considerations on how instructors want AI support to be integrated into their workflows. Informed by these workshops, we developed a system for identifying common patterns in authentic programming examples, and instructors were more productive and experienced lower cognitive demands with our system compared to a baseline condition.

The key contribution of our work is to show that AI support in educational contexts can be beneficial when it follows practices already valued by instructors. Given that there is a major push to incorporate AI systems in education, we provide a framework for centering instructor voices in such systems. For instance, many participants praised the interactions we provided for quickly navigating examples at a high level while explicitly complaining about the chat interfaces common in many GenAI tools, showing the importance of interactions that align with instructor needs.

2.2.2. Co-designing curriculum design tools with non-CS instructors (in progress)

Our prior work has shown that instructors have practices that can be supported effectively with automated support tools. In this work, we plan to identify additional practices that have proved to be efficient in supporting end-user programmer learning goals. Most importantly, we are planning to interview instructors who are teaching computing in non-CS domains to amplify voices that are outside the CS sphere of influence. These interviews will inform co-design sessions for tools that support the development of instruction that follows the best practices collected from non-CS instructors, which, in turn, could support broader adoption of effective instructional techniques for end-user programmers.

In our initial findings from the interviews for this project, we noted that instructors value live coding and showcasing authentic examples and use cases of tools they are covering in their courses. For most instructors, ensuring that students can apply the knowledge they acquire in the course in their own projects is a key goal, and their instructional tasks are designed with this goal in mind. Therefore, our initial findings seem to support the importance of purpose-first instruction for end-user programmers.

3. Conclusion & Next Steps

In my past work, I focused on showing the importance of emphasizing code's purpose for learners interested in end-user programming. My ongoing and proposed research will focus on identifying concrete practices for highlighting code's purpose, and co-designing tools with instructors to support this mode of instructional design. I am seeking feedback on what these practices could entail, as well as methodological advice for conducting co-design sessions that empower instructors as the main stakeholders.

4. References

Ali, M., & Dasgupta, S. (2025). "even though i went through everything, i didn't feel like i learned a lot": Insights from experiences of non-computer science students learning to code. In *Proceedings of the 2025 chi conference on human factors in computing systems.* New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi-org.proxy2.library.illinois.edu/10.1145/3706598.3713624` doi: 10.1145/3706598.3713624

Becker, B. A., & Fitzpatrick, T. (2019). What do cs1 syllabi reveal about our expectations of introductory programming students? In *Proceedings of the 50th acm technical symposium on computer science education* (pp. 1011–1017).

Brandt, J., Guo, P. J., Lewenstein, J., & Klemmer, S. R. (2008, May). Opportunistic programming: How rapid ideation and prototyping occur in practice. In *Proceedings of the 4th international workshop on End-user software engineering* (pp. 1–5). Leipzig Germany: ACM. doi: 10.1145/1370847.1370848

Chung, A. M., & Guo, P. J. (2024). Perpetual teaching across temporary places: Conditions, motivations, and practices of media artists teaching computing workshops. In *Proceedings of the 2024 acm conference on international computing education research-volume 1* (pp. 374–388).

Cunningham, K., Ericson, B. J., Agrawal Bejarano, R., & Guzdial, M. (2021, May). Avoiding the Turing Tarpit: Learning Conversational Programming by Starting from Code's Purpose. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1–15). Yokohama Japan: ACM.

Demirtas, M. A., Zheng, C., & Cunningham, K. (2026). Teaching Authentic Programming Applications to Novices: Purpose-first Tutorials in a General Education Computing Course. In *Proceedings of the 57th ACM Technical Symposium on Computer Science Education V.1* (Vol. 1, pp. 274–280). Association for Computing Machinery. Retrieved 2026-04-23, from `https://dl.acm.org/`

doi/10.1145/3770762.3772574

Dorn, B., & Guzdial, M. (2010). Discovering computing: perspectives of web designers. In *Proceedings of the sixth international workshop on computing education research* (p. 23–30). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1839594.1839600 doi: 10.1145/1839594.1839600

Hur, J., & Cunningham, K. (2024, August). Profiling Conversational Programmers at University: Insights into their Motivations and Goals from a Broad Sample of Non-Majors. In *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1* (Vol. 1, pp. 293–311). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3632620.3671123

Jain, Y., Demirtas, M. A., & Cunningham, K. I. (2025). Plaid: Supporting computing instructors to identify domain-specific programming plans at scale. In *Proceedings of the 2025 chi conference on human factors in computing systems.* New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi-org.proxy2.library.illinois.edu/10.1145/3706598.3713832 doi: 10.1145/3706598.3713832

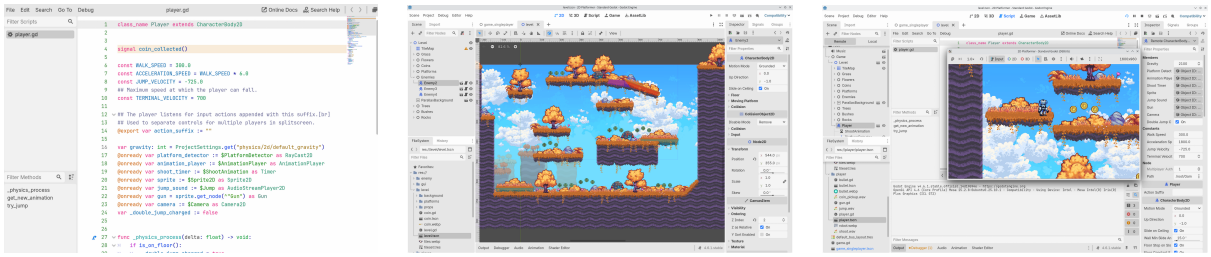
Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., ... Wiedenbeck, S. (2011, April). The state of the art in end-user software engineering. *ACM Comput. Surv.*, *43*(3). Retrieved from https://doi-org.proxy2.library.illinois.edu/10.1145/1922649.1922658 doi: 10.1145/1922649.1922658

O'Brien, G. (2025). How scientists use large language models to program. In *Proceedings of the 2025 chi conference on human factors in computing systems.* New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi-org.proxy2.library.illinois.edu/10.1145/3706598.3713668 doi: 10.1145/3706598.3713668

Thorgeirsson, S., Ewen, T., & Su, Z. (2025). What Can Computer Science Educators Learn From the Failures of Top-Down Pedagogy? In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (pp. 1127–1133). ACM. Retrieved 2026-04-17, from https://dl.acm.org/doi/10.1145/3641554.3701873 doi: 10.1145/3641554.3701873

From Abstract to Concrete: Programming the Running System via Direct Manipulation and Live Feedback

Eva Krebs
Hasso Plattner Institute
University of Potsdam
eva.krebs@hpi.uni-potsdam.de



(a) Code (abstract behavior) *(b) Scene editor (blueprint)* *(c) Game (running instance)*

Figure 1 – Different views with varying levels of concreteness of the same program

Abstract

Code being abstract makes it powerful, but it is also more difficult to understand. Developers will have to map abstract code descriptions to the behavior of concrete, running program instances. Additionally, as many IDEs have different views for code and the running program, constant context switches are required. Some environments offer live programming capabilities to reduce the amount of context switches, but often have limitations that will require program restarts or a context switch to another view. In my research, I want to explore notations, tools, and systems that bring the abstract and the concrete closer together. I am exploring systems that provide concrete feedback directly where it is needed, require fewer context switches between views, and remove some of the limitations of other live programming systems. With my thesis, I plan to contribute the design and implementation of these systems as well as insights into the feasibility of the concepts via case studies, holistic analysis, and user studies. In the long term, my aim is to enhance the programming experience of developers by removing accidental complexity.

1. Motivation

Code is an abstract description of behavior, making it powerful by being able to address a variety of situations. However, this also makes it harder to understand for developers and requires a mental mapping between code and concrete behavior. Especially in domains that heavily feature user interaction, we want fast cycles of trying out the running program and incorporating the resulting feedback back into code (Calp & Akcayol, 2019). One possibility to fasten feedback cycles is live programming, where changes are materialized in the running program as quickly as possible (Tanimoto, 2013), which may in turn also heighten the developer's understanding of the system (Krämer, Kurz, Karrer, & Borchers, 2014).

However, several development aspects may still cause friction. Many IDEs consist of different views of the same program that require context changes, for example, splitting code, abstract UI definitions, and the running program (see Figure 1). Some live programming changes may not manifest without further intervention; for example, changes to initialization code will not apply to already existing instances.

In my thesis, I explore tools that address these and related challenges. I research both new ways of defining behavior as well as extensions of existing live programming concepts that reduce friction.

2. Background

In the following, I will introduce live programming concepts and tooling that my work builds upon. Afterwards, I will quickly introduce the Godot game engine that was the basis for example implementations of my tooling concepts.

2.1. Liveness and Direct Manipulation

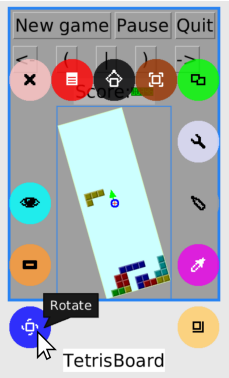


Figure 2 – Halo

Live programming usually refers to enabling developers to modify running programs and instances (Tanimoto, 2013). The capability of reloading code of running instances is also called hot reloading or live reloading in some systems. However, this does not mean all changes will immediately manifest in a running program. For example, initialization code of existing instances will not be re-executed, and code dependent on events or similar will still require the correct trigger/state to occur.

Direct manipulation encompasses tools that allow quick and easy changes to objects. For example, the live programming environment Squeak/Smalltalk (Goldberg & Robson, 1983; Ingalls, Kaehler, Maloney, Wallace, & Kay, 1997) features a concept called *halo*. A halo is a meta-menu that can be opened for any UI object. It provides tools available for that object ordered around its extent, see Figure 2. Common halo tools include, but are not limited to, object deletion, rotation, movement, and opening additional tools, e.g., an inspector for the selected object.

Some of the tools presented in the approach are based on *babylonian programming* (Rein et al., 2019; Rauch, Rein, Ramson, Lincke, & Hirschfeld, 2019). Babylonian programming uses examples to enable developers to get concrete feedback on code behavior in the code editing view. For instance, developers can attach probes to expressions in code to receive domain-specific visualizations of the concrete values that the expression takes during example execution. A similar concept to probes are watches found in other IDEs that can visualize state of a running program.

2.2. Godot Game Engine

Game development is a domain that greatly depends on user experience and thus immensely benefits from faster feedback loops (Schell, 2019). Games are usually made with game engines, dedicated IDEs that bundle libraries and tools for recurring game development needs.

I am creating example implementations of my proposed tools for the game engine Godot (Foundation, Linietsky, Manzur, & contributors, 2007). Godot is an open-source game engine whose editor is largely self-sustained. This enables tool developers to both make powerful extensions without changing the underlying engine while also making engine changes possible if absolutely required.

Godot features three main views that are commonly found in game engines: a code editor, a scene editor (allows visual assembly of game elements in a scene tree), and the running game (see Figure 1). Additionally, it features some live programming functionality and related tooling:

- Godot supports **hot reloading**, meaning that code changes can be swapped into the running program.
- While a game is running, developers can use the tool **remote view** to see the scene tree of the running game.
- Godot features an **inspector** that can be used both in the scene editor but also to manipulate exposed variables of a running game.
- Furthermore, Godot supports common programming and debugging tooling, including but not limited to syntax highlighting, an integrated console that supports printing operations, and breakpoints.

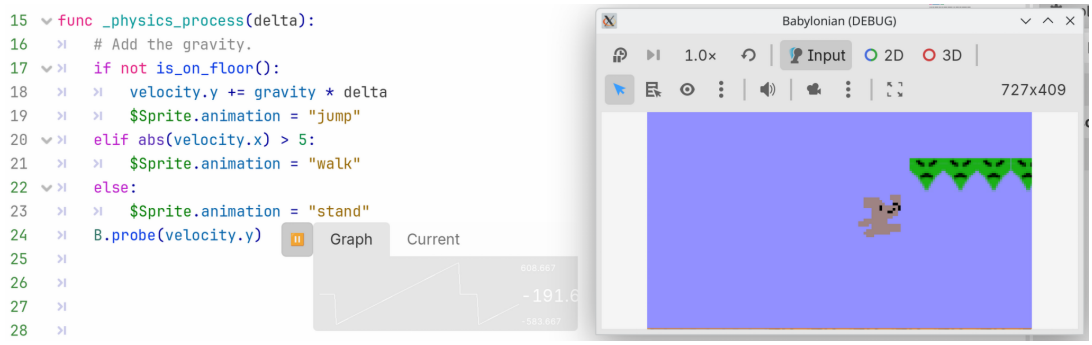


Figure 3 – Code augmented with concrete feedback through probes.

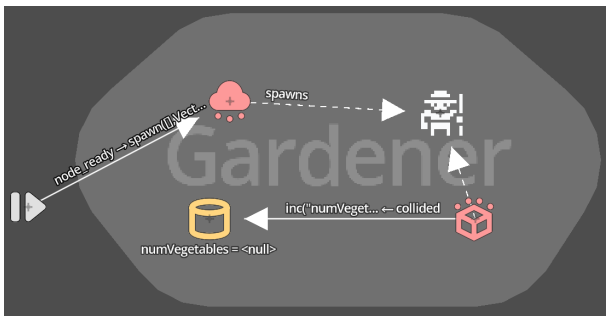
3. Approach: The Varying Levels of Concreteness

There are several ways of reducing context switches between abstract (e.g., code) and concrete (e.g., a running instance) of a program. In the following, I would outline some of the concepts I researching.

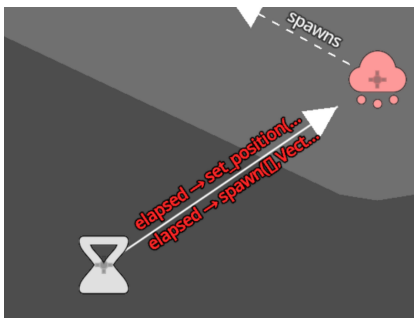
Probes and feedback Probes are an opportunity to bring concrete feedback to the abstract code view (Krebs et al., 2024). I am looking into how probe-based workflows compare to established workflows, e.g., using breakpoints or print statements. Additionally, I am exploring new types of probes, e.g., probes that are placed in the running program to enable further debugging/exploration while working on a concrete instance.

Prototyping in a single view In Pronto (Krebs et al., 2025), changes between code view and scene editing view are not needed (see Figure 4). All behavior is reified in the scene editor. Functionality that is commonly needed for game development is encapsulated in *Behaviors* that can be placed as visual elements in the scene editor. With *Connections*, developers can define how Behaviors should affect each other and/or game elements.

Keeping the running instance alive I am looking into removing friction from live programming sessions by making change materialization understandable, providing additional functionality to avoid unnecessary program restarts, and direct manipulation tools whose changes can be persisted on demand.



(a) Abstract behaviors are visually reified alongside game elements with connections between them.



(b) Connections flash when triggered in the running game.

Figure 4 – Pronto combines visual composition and behavior definition in a single view

4. Research Methods

My research aims to enhance programming experience by exploring different ways of reducing friction during both general development and live programming in particular. In the following, I will shortly outline my research goals and corresponding evaluation concepts.

4.1. Research Goals and Expected Contributions

I am exploring different ways of reducing friction during development. I am proposing tooling that addresses causes of friction, such as context switches between different program views and program restarts during live programming. The main underlying question I want to answer regarding my proposed systems and extensions is their trade-offs compared to state-of-the-art systems.

By addressing these challenges, I expect to make contributions in the following areas:

- Concepts for reducing friction
- Example implementations
- Insights on specific design challenges and tools from evaluations

4.2. Evaluation Concept

I plan to evaluate my proposed concepts and extensions via the following methods:

- I will show their feasibility through **reference implementations** and **case studies**.
- I will provide an initial argument for their impact on programming experience via **holistic analysis** with a focus on the cognitive dimensions of notations (Blackwell, 2005) and **cognitive walkthroughs** (Lewis, Polson, Wharton, & Rieman, 1990).
- Further insights on their impact on programming experience will be obtained via **user studies**.

5. References

Blackwell, A. F. (2005). Cognitive dimensions of notations. In *2005 IEEE symposium on visual languages and human-centric computing (VL/HCC 2005), 21-24 september 2005, dallas, tx, USA* (p. 3). IEEE Computer Society. Retrieved from https://doi.org/10.1109/VLHCC.2005.26 doi: 10.1109/VLHCC.2005.26

Calp, M. H., & Akcayol, M. A. (2019). The importance of human computer interaction in the development process of software projects. *CoRR*, *abs/1902.02757*. Retrieved from http://arxiv.org/abs/1902.02757

Foundation, G., Linietsky, J., Manzur, A., & contributors. (2007). *Godot engine.* Retrieved 2026-05-07, from https://godotengine.org/

Goldberg, A., & Robson, D. (1983). *Smalltalk-80.* Harlow, England: Longman Higher Education.

Ingalls, D., Kaehler, T., Maloney, J., Wallace, S., & Kay, A. C. (1997). Back to the future: The story of squeak - A usable smalltalk written in itself. In M. E. S. Loomis, T. Bloom, & A. M. Berman (Eds.), *Proceedings of the 1997 ACM SIGPLAN conference on object-oriented programming systems, languages & applications, OOPSLA 1997, atlanta, georgia, october 5-9, 1997* (pp. 318–326). ACM. Retrieved from https://doi.org/10.1145/263698.263754 doi: 10.1145/263698.263754

Krämer, J., Kurz, J., Karrer, T., & Borchers, J. O. (2014). How live coding affects developers' coding behavior. In S. D. Fleming, A. Fish, & C. Scaffidi (Eds.), *IEEE symposium on visual languages and human-centric computing, VL/HCC 2014, melbourne, vic, australia, july 28 - august 1, 2014* (pp. 5–8). IEEE Computer Society. Retrieved from https://doi.org/10.1109/VLHCC.2014.6883013 doi: 10.1109/VLHCC.2014.6883013

Krebs, E., Beckmann, T., Geier, L., Grenda, J., Ramson, S., & Hirschfeld, R. (2025). All in one: Rapid game prototyping in a single view. In N. Yamashita et al. (Eds.), *Proceedings of the 2025 CHI conference on human factors in computing systems, CHI 2025, yokohamajapan, 26 april 2025- 1 may 2025* (pp. 266:1–266:17). ACM. Retrieved from https://doi.org/10.1145/3706598.3714251 doi: 10.1145/3706598.3714251

Krebs, E., Mattis, T., Dörbandt, M., Schulz, O., Rinard, M. C., & Hirschfeld, R. (2024). Implementing babylonian/g by putting examples into game contexts. In E. Söderberg & L. Church (Eds.), *Companion proceedings of the 8th international conference on the art, science, and engineering of programming, programming companion 2024, lund, sweden, march 11-15, 2024*. ACM. Retrieved from https://doi.org/10.1145/3660829.3660847 doi: 10.1145/3660829.3660847

Lewis, C. H., Polson, P. G., Wharton, C., & Rieman, J. (1990). Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In J. C. Chew & J. A. Whiteside (Eds.), *Conference on human factors in computing systems, CHI 1990, seattle, wa, usa, april 1-5, 1990, proceedings* (pp. 235–242). ACM. Retrieved from https://doi.org/10.1145/97243.97279 doi: 10.1145/97243.97279

Rauch, D., Rein, P., Ramson, S., Lincke, J., & Hirschfeld, R. (2019). Babylonian-style programming - design and implementation of an integration of live examples into general-purpose source code. *The Art, Science, and Engineering of Programming*, *3*(3), 9. Retrieved from https://doi.org/10.22152/programming-journal.org/2019/3/9 doi: 10.22152/PROGRAMMING-JOURNAL.ORG/2019/3/9

Rein, P., Lincke, J., Ramson, S., Mattis, T., Niephaus, F., & Hirschfeld, R. (2019). Implementing babylonian/s by putting examples into contexts: Tracing instrumentation for example-based live programming as a use case for context-oriented programming. In *Proceedings of the workshop on context-oriented programming, cop@ecoop 2019, july 15-19, 2019, london, UK* (pp. 17–23). ACM. Retrieved from https://doi.org/10.1145/3340671.3343358 doi: 10.1145/3340671.3343358

Schell, J. (2019). *The art of game design* (3rd ed.). London, England: CRC Press.

Tanimoto, S. L. (2013). A perspective on the evolution of live programming. In B. Burg, A. Kuhn, & C. Parnin (Eds.), *Proceedings of the 1st international workshop on live programming, LIVE 2013, san francisco, california, usa, may 19, 2013* (pp. 31–34). IEEE Computer Society. Retrieved from https://doi.org/10.1109/LIVE.2013.6617346 doi: 10.1109/LIVE.2013.6617346

How Is AI Reshaping Computer Science Students' Cognition of Programming?

Ruolin Li
University of Limerick, Ireland
li.ruolin@ul.ie

Jim Buckley
University of Limerick, Ireland
Jim.Buckley@ul.ie

Tabea De Wille
University of Limerick, Ireland
Tabea.Dewille@ul.ie

Abstract

Generative AI is increasingly embedded in computer science education, creating new opportunities and concerns regarding how students learn to program. This research investigates AI's impact on computer science students' cognition by examining how cognitive constructs are represented across programming-education research and how their significance varies across software engineering activities in AI and non-AI literature. A systematic mapping study, informed by established software engineering mapping guidelines and PRISMA 2020, screened 14,628 records between 2023 and 2025 from six databases and retained 191 primary studies, including 80 AI-related and 111 non-AI studies. Mixed deductive-inductive thematic analysis identified twelve cognitive and pedagogical themes. Cognitive processes, computational thinking, and comprehension were most frequently examined, while metacognition, motivation, cognitive load, misconceptions, and mental models capture complementary regulatory and interpretive dimensions. Findings indicate that cognition is task-sensitive: debugging foregrounds program comprehension, misconception diagnosis, and evaluation, whereas collaborative and pair-programming activities foreground explanation, shared problem solving, and cognitive regulation. AI-related studies predominantly address reasoning, interaction, and regulation, while foundational constructs such as comprehension, misconceptions, and mental models remain more prominent in non-AI research. Rather than viewing this distribution solely as conceptual fragmentation, this research proposes that it reflects both task specialisation and the early development of AI-mediated programming education. The mapping provides evidence that programming cognition is task-sensitive and identifies a research gap between AI-assisted code production and the verification, diagnosis, and judgement activities that remain central to durable learning. This gap motivates longitudinal research on AI amplification, cognitive debt, and emerging mental labour.

Keywords: generative AI; programming education; computer science education; cognition; systematic mapping study; computational thinking; metacognition

1. Background and research gap

The rapid adoption of generative artificial intelligence (GenAI) in the digital world has fundamentally changed how programming is learned and practiced (Becker et al., 2022). In industry settings, AI-assisted tools have demonstrated substantial productivity gains among developers (Peng et al., 2023; Paradis et al., 2025), calling for the acquisition of a different skillset and curriculum change in the third-level education of software engineering students. In parallel, educators increasingly integrate AI into programming instruction, feedback, and assessment practices, while students exhibit diverse proactive and reactive behaviours as they interact with AI throughout their learning journeys (Prather et al., 2023; Liu et al., 2024; Sheese et al., 2024; Güner and Er, 2025). As a result, computer science students are no longer learning programming under the same cognitive conditions as previous generations.

Large language model tools can generate code, explain snippets, propose tests, and participate in debugging dialogues. These capabilities may support experimentation and immediate feedback, but they also make it possible to complete programming tasks without establishing robust conceptual understanding (Prather et al., 2024). The educational question is therefore not only whether students use AI, but how AI changes the reasoning, evaluation, self-regulation, and knowledge construction through which programming is learned, and if those changes leave graduates in a better or worse position for AI-supported software engineering careers.

This question sits alongside a longer tradition of research on programming cognition. Computer science education has examined computational thinking, program comprehension, misconceptions,

debugging, mental models, cognitive load, and metacognition as related but distinct dimensions of programming learning (Wing, 2006; Soloway and Ehrlich, 1984; Pennington, 1987; Qian and Lehman, 2018; McCauley et al., 2008; Sweller, 1988; Prather et al., 2018). Recent reviews have considered GenAI adoption, assessment, and pedagogical practice in computing education (Prather et al., 2023; Golrang and Sharma, 2025), while other reviews concentrate on particular cognitive constructs such as computational thinking or cognitive load (Melro et al., 2023; Suryani et al., 2024). However, these strands rarely provide an integrated account of where cognition is studied, in which programming tasks, and how AI-related research compares with the adjacent non-AI literature.

Like Mansoor et al. (2024), we argue that such a task-sensitive perspective matters. Programming cognition is enacted across activities including code generation, debugging, program comprehension, testing, review, and collaboration, rather than solely during code writing; these activities are also shaped by the presence or absence of AI support. Generative AI may redistribute, rather than eliminate, cognitive effort: while it can accelerate code production, learners must still inspect, test, diagnose, judge, and integrate generated outputs (Robins, Rountree and Rountree, 2003; Bacchelli and Bird, 2013; Vaithilingam, Zhang and Glassman, 2022; Barke, James and Polikarpova, 2023), particularly when those outputs appear plausible but are incorrect (Pearce et al., 2022). The mapping therefore regards software engineering tasks as the contexts in which cognitive constructs acquire educational meaning.

2. Research questions

RQ1a. What cognitive and pedagogical themes are addressed in programming-education research?

RQ1b. In which software engineering task contexts do the proposed themes occur?

RQ2a. How is the mapped literature distributed between AI-related and non-AI studies?

RQ2b. What thematic relationships are visible across the mapped literature?

RQ3. What research gaps emerge from the interaction between programming cognition, software engineering tasks and AI assistance?

3. Mapping study completed to date

A systematic mapping study synthesised 191 primary studies: 80 substantively examine GenAI in programming-learning contexts and 111 provide the analogous non-AI knowledge base needed to interpret trends concurrent with AI adoption. A deductive-inductive coding scheme captured 12 themes and the programming tasks in which they appeared. Multiple codes were permitted for each article. The primary author completed the coding; the second and final authors each reviewed 12 articles, and differences were resolved through debate towards consensus. Article-level co-occurrence analysis served as an internal analytic cross-check of the thematic map, rather than an independent methodological triangulation.

4. Findings and conceptual synthesis

Together, RQ1a and RQ1b show that programming cognition is not a single phenomenon. As shown in figure 1, the literature is cognitively centred, led by cognitive processes (n = 82), computational thinking (n = 51), and comprehension (n = 38), but its emphasis changes with the task. Programming projects are the most frequent task family (n = 51), followed by debugging (n = 36), function writing (n = 25), program comprehension (n = 24), and collaborative development (n = 19). Debugging makes comprehension, misconception diagnosis, cognitive effort, and evaluation conspicuous; collaborative development foregrounds explanation, help-seeking, and shared regulation.

The answer to RQ2(a/b) shows a literature in transition. AI-related studies cluster around cognitive processes, interaction, and learning regulation, while comprehension, misconceptions, and mental models remain areas of interest even after the emergence of AI. Across the coded corpus, 41 of 66 possible undirected theme pairs occurred. The most common pairing is comprehension-misconceptions (n = 17), followed by cognitive processes-comprehension (n = 10) and cognitive processes-computational thinking (n = 10). Figure 1 presents AI as a cross-cutting influence on an established cognitive-pedagogical landscape, not as a force that replaces its foundations.

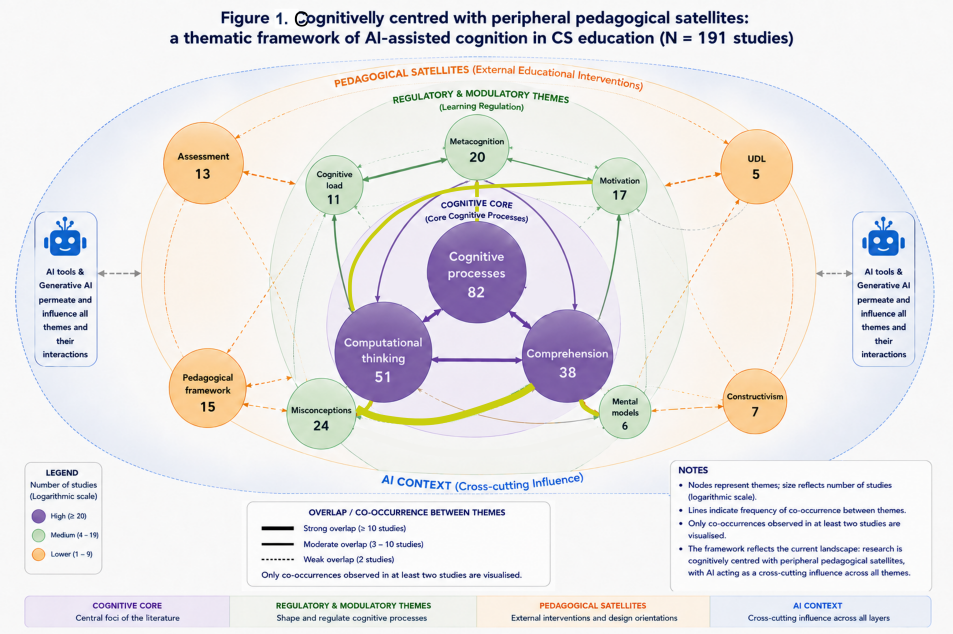


Figure 1 - Conceptual synthesis derived from thematic frequencies and article-level co-occurrence analysis. Node size reflects the number of studies; line thickness reflects observed co-occurrence frequency.

RQ3 is therefore framed as an evidence-led gap. Figure 2 shows that AI-related studies are concentrated in programming projects (31 AI-related versus 20 non-AI studies) and code completion (9 versus 5). In contrast, code comprehension (5 versus 18), debugging (15 versus 21), and collaborative/pair programming (7 versus 12) receive comparatively less AI-related coverage; code review is balanced (3 versus 3), and testing is nearly absent. The mapping cannot show that AI has already redistributed student effort and cognitive load as we expected, yet it does show that research coverage has not yet fully followed the verification, diagnosis, review, and testing work that AI-assisted programming makes educationally consequential.

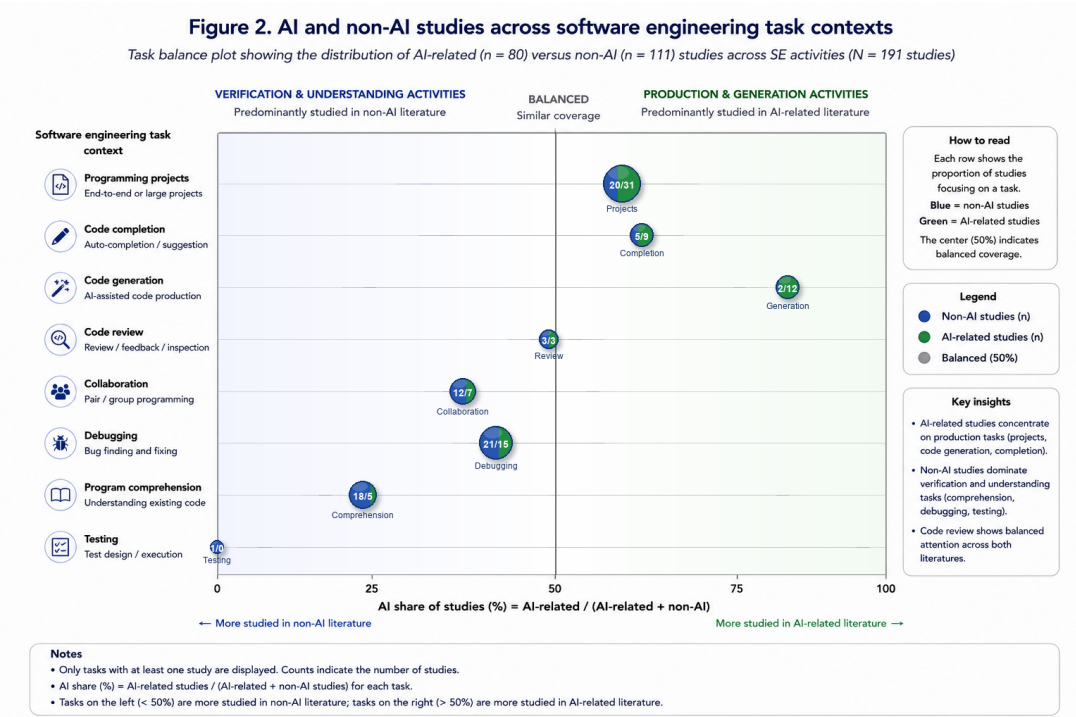


Figure 2 - AI-related and non-AI studies by software engineering task. N = 191 task-coded primary studies; multiple task coding was permitted.

5. Doctoral research agenda

The next phase of the doctorate will develop integrated, task-sensitive accounts of learning with AI by specifying the programming task, form of AI assistance, learner role, intended cognitive process, pedagogy, and assessment evidence. Two linked research directions are central. First, AI amplification and individual cognitive trajectories will examine how prior knowledge, self-regulation, confidence, motivation, accessibility needs, and help-seeking shape AI use over time. The working proposition is that AI may amplify learner tendencies: reflective students may test and compare suggestions, while anxious, avoidant, or less experienced students may accept the first plausible answer prematurely. We hypothesize that this pattern may exacerbate differences in high-achievers and poorer students.

Second, the project will investigate cognitive load, cognitive debt, and emerging mental labour. Cognitive debt describes cases in which a learner completes an AI-assisted task while accumulating incomplete understanding that later constrains retention, debugging, code review, and system-level reasoning. Longitudinal, mixed-method research is needed to determine when AI frees cognitive capacity for learning and when it transfers effort toward verification, comparison, orchestration, and responsibility for judging outputs. This agenda reframes executable output as necessary but insufficient evidence of learning; future measures should also capture explanation, error diagnosis, misconception repair, metacognitive monitoring, independent performance after assistance is withdrawn, and transfer to new programming problems.

References

Bacchelli, A. and Bird, C. (2013) ‘Expectations, outcomes, and challenges of modern code review’, Proceedings of the 35th International Conference on Software Engineering, pp. 712–721. doi: 10.1109/ICSE.2013.6606617.

Barke, S., James, M.B. and Polikarpova, N. (2023) ‘Grounded Copilot: How programmers interact with code-generating models’, Proceedings of the ACM on Programming Languages, 7(OOPSLA1), pp. 85–111. doi: 10.1145/3586030.

Becker, B.A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., and Santos, E.A. (2022) ‘Programming Is Hard – Or at Least It Used to Be: Educational Opportunities And Challenges of AI Code Generation’, doi: 10.48550/arxiv.2212.01020.

Golrang, A. and Sharma, K. (2025) ‘The role of AI in shaping educational experiences in computer science: A systematic review’, Computers in Human Behavior: Artificial Humans, 6, doi: 10.1016/j.chbah.2025.100199.

Güner, H. and Er, E. (2025) ‘AI in the classroom: Exploring students’ interaction with ChatGPT in programming learning’, Education and information technologies, 30(9), 12681–12707, doi: 10.1007/s10639-025-13337-7.

Liu, R., Zenke, C., Liu, C., Holmes, A., Thornton, P., and Malan, D.J. (2024) ‘Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education’, in Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1, New York, NY, USA: ACM, 750–756, doi: 10.1145/3626252.3630938.

Mansoor, N., Peterson, C.S., Dodd, M.D., and Sharif, B. (2024) ‘Assessing the Effect of Programming Language and Task Type on Eye Movements of Computer Science Students’, ACM transactions on computing education, 24(1), doi: 10.1145/3632530.

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., and Zander, C. (2008) ‘Debugging: a review of the literature from an educational perspective’, Computer science education, 18(2), 67–92, doi: 10.1080/08993400802114581.

Melro, A., Tarling, G., Fujita, T., and Kleine Staarman, J. (2023) ‘What Else Can Be Learned When Coding? A Configurative Literature Review of Learning Opportunities Through Computational

Thinking', Journal of educational computing research, 61(4), 901–924, doi: 10.1177/07356331221133822.

Paradis, E., Grey, K., Madison, Q., Nam, D., Macvean, A., Meimand, V., Zhang, N., Ferrari-Church, B., and Chandra, S. (2025) 'How Much Does AI Impact Development Speed? an Enterprise-Based Randomized Controlled Trial', in IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (Online), IEEE, 618–629, doi: 10.1109/ICSE-SEIP66354.2025.00060.

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B. and Karri, R. (2022) 'Asleep at the Keyboard? Assessing the security of GitHub Copilot's code contributions', 2022 IEEE Symposium on Security and Privacy, pp. 754–768. doi: 10.1109/SP46214.2022.9833571.

Peng, S., Kalliamvakou, E., Cihon, P., and Demirer, M. (2023) 'The Impact of AI on Developer Productivity: Evidence from GitHub Copilot', doi: 10.48550/arxiv.2302.06590.

Pennington, N. (1987) Stimulus structures and mental representations in expert com prehension a computer programs. Cognitive Psychology 19, 295-341.

Prather, J., Denny, P., Leinonen, J., Becker, B.A., Albluwi, I., Craig, M., Keuning, H., Kiesler, N., Kohn, T., Luxton-Reilly, A., MacNeil, S., Petersen, A., Pettit, R., Reeves, B.N., and Savelka, J. (2023) 'The Robots Are Here: Navigating the Generative AI Revolution in Computing Education', in Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education, New York, NY, USA: ACM, 108–159, doi: 10.1145/3623762.3633499.

Prather, J., Pettit, R., McMurry, K., Peters, A., Homer, J., and Cohen, M. (2018) 'Metacognitive Difficulties Faced by Novice Programmers in Automated Assessment Tools', in Proceedings of the 2018 ACM Conference on International Computing Education Research, New York, NY, USA: ACM, 41–50, doi: 10.1145/3230977.3230981.

Prather, J., Reeves, B., Leinonen, J., MacNeil, S., Randrianasolo, A.S., Becker, B., Kimmel, B., Wright, J., and Briggs, B. (2024) 'The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers', doi: 10.48550/arxiv.2405.17739.

Qian, Y. and Lehman, J. (2018) 'Students' Misconceptions and other difficulties in introductory programming: A literature review: A Literature Review', ACM transactions on computing education, 18(1), doi: 10.1145/3077618.

Robins, A., Rountree, J. and Rountree, N. (2003) 'Learning and teaching programming: A review and discussion', Computer Science Education, 13(2), pp. 137–172. doi: 10.1076/csed.13.2.137.14200.

Sheese, B., Liffiton, M., Savelka, J., and Denny, P. (2024) 'Patterns of Student Help-Seeking When Using a Large Language Model-Powered Programming Assistant', in Seton, C. and Herbert, N., eds, Proceedings of the 26th Australasian Computing Education Conference, New York, NY, USA: ACM, 49–57, doi: 10.1145/3636243.3636249.

Soloway, E. and Ehrlich, K. (1984) 'Empirical studies of programming knowledge', IEEE Transactions on Software Engineering, SE-10(5), pp. 595–609. doi: 10.1109/TSE.1984.5010283.

Suryani, M., Santoso, H.B., Schrepp, M., Aji, R.F., Hadi, S., Sensuse, D.I., Suryono, R.R., and Kautsarina (2024) 'Role, Methodology, and Measurement of Cognitive Load in Computer Science and Information Systems Research', IEEE access, 12, 190007–190024, doi: 10.1109/ACCESS.2024.3514355.

Sweller, J. (1988) 'Cognitive load during problem solving: Effects on learning', Cognitive science, 12(2), 257–285, doi: 10.1207/s15516709cog1202_4.

Vaithilingam, P., Zhang, T. and Glassman, E.L. (2022) 'Expectation vs. Experience: Evaluating the usability of code generation tools powered by large language models', CHI Conference on Human Factors in Computing Systems Extended Abstracts, pp. 1–7. doi: 10.1145/3491101.3519665.

Wing, J.M. (2006) 'Computational thinking', Communications of the ACM, 49(3), pp. 33–35. doi: 10.1145/1118178.1118215.