

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

Niloofer Azizi¹ and Alan Blackwell²

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 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 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.

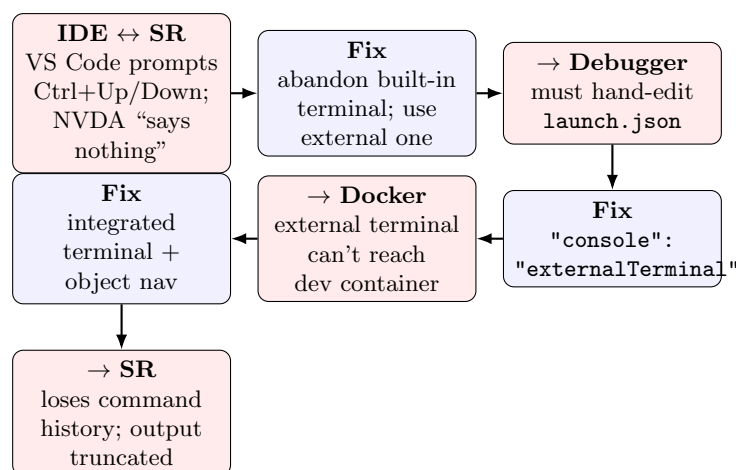


Fig. 1. Each accessibility fix relocates the barrier into the next tool. No single component is inaccessible; the failure is in their composition. Red: interoperability failure. Blue: workaround. Source: Program-L thread “*how to navigate terminal in vscode with nvda?*” (Jan. 2023).

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%
<i>Any solution proposed</i>	<i>1,116</i>	<i>70.4%</i>

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)	<i>keep</i> — 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	<code>threads_for_llm_all.json</code>	2378
Successful Gemini outputs	<code>gemini_outputs_all/</code>	2344
Gemini failures / parsing failures	<code>gemini_failed_all/</code>	100
Filtered accessibility-related thematic outputs	<code>gemini_thema_analysis_out/</code>	1585
Valid thematic files loaded for clustering	semantic clustering input	1583
Threads assigned to semantic clusters	<code>cluster ≠ -1</code>	851
Noise / unassigned threads	<code>cluster = -1</code>	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 *vi/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*.