Accessibility Heuristics for Vibe Coding Interfaces

Shalini Madan
shalinii@umich.edu
School of Information,
University of Michigan
Ann Arbor, Michigan, USA

Sreelakshmi Surabiyil Bindu
sreelaks@umich.edu
School of Information,
University of Michigan
Ann Arbor, Michigan, USA

Venkatesh Potluri
potluriv@umich.edu
School of Information,
University of Michigan
Ann Arbor, Michigan, USA

Abstract

AI coding tools are transforming programming, shifting it from a highly editorial process into a conversational activity. Popular Vibe coding tools such as Replit and Cursor integrate natural language interfaces with traditional development environments. While these tools promise simplicity, increased productivity, and automation, they also introduce new accessibility challenges for blind or visually impaired (BVI) developers. A systematic identification of these accessibility challenges necessitates comprehensive guidelines that account for the complex interactions in these tools. To address this need, we develop accessibility heuristics to assess the accessibility of AI conversational programming tools. Our heuristics combine web accessibility guidelines, best practices to design conversational interfaces, and accessibility needs specific to BVI developers. Our evaluation of three widely used conversational programming tools shows that most accessibility challenges arise from complex keyboard interactions, poor focus management, and insufficient feedback and access to the various actions and output of the tools.

CCS Concepts

• **Human-centered computing → Accessibility design and evaluation methods**.

Keywords

Heuristics, Developer Tools, AI Coding Assistants

ACM Reference Format:
Shalini Madan, Sreelakshmi Surabiyil Bindu, and Venkatesh Potluri. 2025. Accessibility Heuristics for Vibe Coding Interfaces. In *The 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25), October 26–29, 2025, Denver, CO, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3663547.3759729

1 Introduction

Programming and code generation using Artificial Intelligence (AI) is becoming prevalent among software developer communities [15]. AI coding tools offer varying degrees of assistance, from enhanced code completion to end-to-end software engineering tasks. End-user programming and Vibe coding Tools such as Cursor[5] and Firebase Studio[25] offer increased assistance by taking in natural language instructions and operating on the user's developer tool, allowing the user to intervene and edit code manually. Agentic coding tools such as Google Jules[13] and Codex[18] provide end-to-end AI-assisted coding support with minimal visibility into their actions[23]. The agentic tools combine interactions found in conversational interfaces, the web, and integrated development environments (IDEs) to offer conversational programming experiences with varying degrees of AI assistance and user agency.

While the emergence of AI coding assistance in conversational programming tools offers great potential to lower the floor for programming [4], it presents new accessibility barriers to blind or visually impaired (BVI) screen reader users. For example, AI code completion tools constantly suggest blocks of code as grayed-out ghost text, giving screen reader users no controls to inspect these suggestions before acting on them. This leads to unintended code edits, overwhelming users by interfering with their cognitive processing.[9] Vibe coding tools increase this cognitive overload by requiring users to switch between the prompt (natural language instruction), generated code (response), and the codebase. In a nonvisual context, where content must be listened to sequentially, this overload makes it difficult to assess the relevance of information to the task at hand, leading to unnecessary context switches [1, 3, 9]. These new challenges compound long-standing accessibility barriers in code navigation, debugging, and access to documentation [2, 20, 24]. This abundance of accessibility gaps signals an immediate need to study the accessibility of conversational programming tools. A holistic and systematic investigation, however, may be challenging due to the lack of clear guidelines and a framework to assess their accessibility.

While accessibility heuristics offer clear guidelines for websites (WCAG) [31], authoring tools (ATAG)[30], and integrated development environments [20], they operate in silos and may not address the accessibility challenges posed by emerging end-user programming paradigms that converge natural language, web, and programming interfaces. [1] To address this gap, our work answers the following research question: *How do we systematically evaluate the accessibility of conversational programming tools?* We contextualize and adapt heuristics from multiple domains to suit conversational AI programming environments. We draw from WCAG and Heuristics for usable Conversational Interfaces and contextualize them to the coding tool needs of BVI developers. To validate the applicability of our heuristics, we evaluate three popular conversational programming tools for accessibility. We close with next steps to formalize our heuristics.

[1]We use 'conversational programming tools' [21] to describe vibe coding and agentic tools that integrate code generation with natural language interaction, such as GitHub Copilot with Agent Mode, Replit, and Google Jules.

ASSETS '25, Denver, CO, USA

ACM ISBN 979-8-4007-0676-9/2025/10
https://doi.org/10.1145/3663547.3759729

2 Background

We summarize existing guidelines to evaluate conversational interfaces, providing examples to demonstrate how they may not be applicable to assess accessibility of conversational programming tools. We then discuss the accessibility of AI programming assistants and summarize efforts to use heuristics to assess and improve the accessibility of developer experiences for BVI users.

2.1 Accessibility Gaps in Usability Guidelines

Heuristic guidelines for usable conversational AI agentic interactions [14] build on Nielsen's Usability Heuristics[17] for user interface design, a widely adopted standard for evaluating the usability of an interface. Nielsen's heuristics, such as Visibility of System Status (system keeps users informed through timely feedback) and Flexibility and Efficiency of Use (supporting novice and expert users through shortcuts and adaptable workflows), and conversational heuristics like Context Preservation (system maintains conversational context within and across sessions), offer guidance for evaluating system feedback, interaction flow, and user control. While these heuristics state that system status must be perceivable, the implementations often rely heavily on visual cues (pop-ups, color changes, and persistent system status bars) to communicate system actions and status. AI tools and vibe coding platforms may pass this usability heuristic, but may present accessibility barriers for BVI users Similarly, tools may pass heuristics such as Consistency and Standards (users should not have to wonder whether different words, options, or actions mean the same thing), but screen reader users still face challenges when working with AI-generated code. The process is often time-consuming and cognitively demanding, requiring navigation across multiple views, such as the editor, message list, and accessible view [3]. These challenges call for a need to examine how well existing accessibility guidelines address user concerns.

2.2 Evaluating Accessibility in Programming

The widely recognized Web Content Accessibility Guidelines[31] provides testable criteria for accessible web content. These criteria have been augmented with domain-specific needs to uncover accessibility insights and suggest improvements to developer tools. Potluri and Singanamalla *et al.*[19] assess the accessibility of computational notebooks through metrics derived from WCAG, best practices to visualize data [6], and author notebooks. The Notebooks4All project builds on foundational criteria from WCAG, and adapts them to the structure and interaction patterns of computational notebooks to recommend best practices for authoring accessible notebooks[7].

We complement this body of work by exploring the application of WCAG standards and conversational heuristics to offer a systematic approach to evaluate the accessibility of conversational programming platforms.

3 Heuristics Development

We describe our iterative approach to developing these heuristics and summarize the developed set. Conversational programming tools combine web user experiences, conversational interfaces, and traditional programming tools. We reviewed Nielsen's and Conversational heuristics and Web Accessibility Guidelines for this diversity, and to contextualize them for accessibility needs of BVI developers. [14, 20, 30, 31].

Criteria	Parameters	Description
H1. Agent Interface	1.1 Accessible Elements & Media 1.2 Color	Criteria to evaluate if users perceive the interactive and informational elements in the agent's interface [1, 27].
H2. Agent Operation	2.1 Status 2.2 Result 2.3 Action Sequence	Criteria to evaluate if users can access and follow the actions taken by the agent. [3, 14]
H3. User Operation	3.1 Keyboard Access 3.2 Consistent Navigation 3.3 Context Switching 3.4 Focus Management	Criteria to assess full keyboard operability [29], navigation [2, 20], and whether users retain context when switching between interfaces to prompt and code[9, 10].
H4. Help & Recovery	4.1 Error Recovery 4.2 Error Prevention 4.3 Help	Criteria to evaluate how the system helps users recognize and recover from errors [14], and how users can access help and documentation [12, 24].
H5. Adaptability	5.1 Compatibility 5.2 Flexibility	Criteria to assess how well the system accommodates varying user needs.[14, 28]

Table 1: Overview of the Accessibility Heuristics

3.1 Iterative Development

We began our development by identifying a list of over 20 evaluation questions based on prior accessibility guidelines and known screen reader challenges. Through discussions, we identified parameters and their associated questions, consolidating overlapping items and removing repetitions. Subsequent iterations refined the wording of these questions and grouped the parameters into five major heuristic categories (H1-H5), each representing an aspect of the user's interaction with the conversational programming tool. Within each category, related parameters were organized under subheadings (1.1, 1.2) to encompass all evaluations within a broader theme. For example, *1.1: Accessible Elements and Media* was a parameter designed to evaluate the perceivability of visual elements in conversational AI interfaces. Prior research reports that many chat-based AI interfaces include unlabeled or mislabeled UI elements,

making them inaccessible to BVI users [1] and hence, the parameter included assessing whether interface components such as icons, images, and AI-generated UI artifacts were properly labeled with accessible names or ARIA attributes. Evaluation questions grounding this heuristic included: *(1) Do all visual elements have ARIA labels or accessible names interpretable by screen readers? (2) Are all images, icons, and illustrations given meaningful alt text or labels? (3) Are agent-generated visual artifacts accessible via screen reader focus?* We mapped these questions to WCAG 2.1 Success Criterion 1.1.1 (Non-text Content) and 4.1.2 (Name, Role, Value), which state that the provision of text alternatives for non-text content and the requirement that interface elements be programmatically determinable. These criteria are important in the context of AI-assisted coding tools where users interact not only with predefined UI components, but also with buttons and previews generated by the AI during runtime.

3.2 Deriving the Final Parameters

We further iterated and revised the parameters to evaluate the accessibility of AI coding interfaces. Some parameters were merged or redefined as we found overlap. For example, we initially included a parameter *Action Confirmation*, which evaluated whether the system communicated what it was about to do and whether users have control to make changes. However, we did not take this parameter in its original form, as the parameter as a standalone did not have direct relevance to accessibility challenges of screen reader users. We instead used the underlying intent, which was to evaluate if users received meaningful feedback and maintained control, into a more contextualized heuristic for AI-coding agent tool accessibility, and contextualized it as one of the parameters under *H4: Agent Operation*. We provide a summary of the proposed heuristic categories and the 14 final parameters in table 1. We provide the full expanded Heuristic set in the Supplementary Material.

4 Preliminary Validation

To validate the relevance and applicability of our heuristics, we conducted preliminary heuristic evaluations on three end-user conversational programming tools: Visual Studio Code with Copilot Agent Mode [16], Replit Agent[22], and Jules[13]. Evaluations were conducted using JAWS, by two team members. During each tool's evaluation, one member performed the task, and the other member made note of the tools behavior. Both team members discussed and reached an agreement regarding the score of every parameter. We consulted a team member with screen reader expertise to ensure that the evaluations and the tasks performed on the AI tool were using the appropriate screen reader functions.

We designed a use case consisting of 3 steps for our evaluation. Step 1 consisted of prompting the tool to 'Create a simple portfolio website with an about page, a contact form, and placeholders for four projects'. In step 2, we prompted the tool to make minor modifications to the generated website (adding a project placeholder, updating the contact form to include a drop-down menu for inquiry type, and editing the about section with custom text). Step 3 consisted of switching between different views to understand the generated output. The portfolio-creation task was inspired by related HCI studies [3, 8, 11], and chosen to keep tasks straightforward yet complex enough to explore the functionalities supported by the vibe coding tool. The series of steps further helped us examine how vibe coding tools support multi-step workflows involving code editing, iteration, and validation. Our goal was to see if we could make edits and accessibly switch between the generated output files. While the agent was generating a response, we noted any significant discrepancies between the visual display and the screen reader output.

To evaluate each heuristic category, we applied a pass/fail scoring system accompanied by a brief justification explaining the rationale behind the score. In addition to this binary scoring, we introduced a severity scale: Mild, Moderate, and Severe, to characterize the impact of violations for each parameter within these categories. This scale was based on the quality and completeness of information provided by the screen reader and the extent to which the issue created an accessibility barrier for users. For example, unlabeled buttons, unclear agent responses, and the absence of status messages were noted as severe violations and marked as fail, as these elements are important for screen reader users to navigate and interact with the interface. In the context of AI-assisted end-user programming, real-time agent feedback and system state play an important role, and missing or unclear information prevents users from understanding what the tool is doing or how to respond. This loss of control and transparency, hence, not only causes issues with usability, but is a major accessibility barrier.

5 Heuristic Evaluation Result

Our goal was to assess how well our heuristics can surface accessibility strengths and issues within conversational programming environments. We summarize our evaluation below and preview the accessibility of the three chosen tools. We found that most heuristic violations occurred within Agent Operation (H2) and User Operation (H3). Within the Agent Operation (H2) heuristic category, we observed multiple mild-to-severe violations, particularly related to the communication of status updates (2.1). When working with Replit and Jules, the screen reader did not inform users about the actions the tool was taking or it's progress, severely violating H2.1. VS Code offered more visibility into its actions, although through minimal progress indicators. Sighted VS Code users could see completion percentages and real-time information about the tool's actions. However, BLVI developers were limited to audio cues indicating the agent was performing the task and did not have access to granular information about the agent's actions in real time. The sequence of actions taken by the agent was announced only after the agent completed the task, making it difficult for users to follow the agent's behavior. We assigned a mild violation to VS Code for 2.1 as users could still access the agent's actions after it was complete. Within User Operation (H3), we identified moderate-to-severe violations in keyboard accessibility (3.1), consistent navigation(3.2) and focus management (3.4) for both Replit and Jules. The tools did not shift focus to dialog boxes or buttons requiring user attention (3.4), had keyboard traps in some instances (3.1) and shifted focus to unexpected locations while navigating with a keyboard (3.2). Moreover, none of the previews of the AI-generated artifacts were keyboard accessible (3.1).

Overall, we observed that VS Code with Copilot Agent mode demonstrated a comparatively higher degree of accessibility, with most interface elements being perceivable and operable via a screen reader and audio cues provided by the tool. In VS Code with Copilot Agent Mode, mild-to-moderate violations were observed in agent feedback clarity (2.1) and agent plan communication (2.3). Replit and Jules, both browser-based tools, had moderate-to-severe violations across all five heuristic categories. The tools handled basic prompts, but editing or switching between outputs was still inaccessible. We include comprehensive evaluations for the 3 tools in the Supplementary Material.

6 Conclusion & Next Steps

To conclude, our early exploration contributes a first look at a systematic approach to assess the accessibility of emerging programming experiences driven by conversational interactions. Our preliminary evaluations demonstrate the applicability of our approach to identifying nuanced accessibility challenges posed by AI conversational programming interfaces. We lay the foundation for evaluation toolkits to ensure that the ongoing paradigm shift in programming remains accessible. Though comprehensive, results derived from our current heuristic set may not account for accessibility barriers that low vision users may experience when consuming content through visual access technologies such as magnifiers[26]. To address this, will expand the evaluation criteria for our heuristics to assess the accessibility of the user experience when conversational programming tools are used with visual modifications such as inverted colors and zoom. We acknowledge that the validation of the heuristics was only by the team. The next phase includes recruiting and interviewing various stakeholders such as accessibility professionals, BLVI developers, academics, and user researchers to strengthen the validity of the heuristics.

References

[1] Rudaiba Adnin and Maitraye Das. 2024. "I look at it as the king of knowledge": How Blind People Use and Understand Generative AI Tools. In *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility* (St. John's, NL, Canada) *(ASSETS '24)*. Association for Computing Machinery, New York, NY, USA, Article 64, 14 pages. doi:10.1145/3663548.3675631

[2] Khaled Albusays, Stephanie Ludi, and Matt Huenerfauth. 2017. Interviews and Observation of Blind Software Developers at Work to Understand Code Navigation Challenges. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility* (Baltimore, Maryland, USA) *(ASSETS '17)*. Association for Computing Machinery, New York, NY, USA, 91–100. doi:10.1145/3132525.3132550

[3] Nan Chen, Luna K. Qiu, Arran Zeyu Wang, Zilong Wang, and Yuqing Yang. 2025. Screen Reader Users in the Vibe Coding Era: Adaptation, Empowerment, and New Accessibility Landscape. arXiv:2506.13270 [cs.HC] https://arxiv.org/abs/2506.13270

[4] Parmit K. Chilana, Celena Alcock, Shruti Dembla, Anson Ho, Ada Hurst, Brett Armstrong, and Philip J. Guo. 2015. Perceptions of non-CS majors in intro programming: The rise of the conversational programmer. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 251–259. doi:10.1109/VLHCC.2015.7357224

[5] Cursor 2025. Cursor: The AI Code Editor. https://www.cursor.com/

[6] Frank Elavsky, Cynthia Bennett, and Dominik Moritz. 2022. How Accessible is My Visualization? Evaluating Visualization Accessibility with Chartability. In *Eurographics Association and John Wiley & Sons Ltd.*

[7] Tony Fast. 2023. Notebook Authoring Accessibility Checklist. Iota-School "Notebooks for All" (GitHub Pages). https://iota-school.github.io/notebooks-for-all/exports/resources/event-hackathon/notebook-authoring-checklist/ Accessed: 2025-06-25.

[8] Claire Ferrari, Amy Hurst, and Scott Fitzgerald. 2019. Blind Web Development Training at Oysters and Pearls Technology Camp in Uganda. In *Proceedings of the 16th International Web for All Conference* (San Francisco, CA, USA) *(W4A '19)*. Association for Computing Machinery, New York, NY, USA, Article 18, 10 pages. doi:10.1145/3315002.3317562

[9] Claudia Flores-Saviaga, Benjamin V. Hanrahan, Kashif Imteyaz, Steven Clarke, and Saiph Savage*. 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 (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 1164, 17 pages. doi:10.1145/3706598.3714008

[10] Claudia Flores-Saviaga, Benjamin V. Hanrahan, Kashif Imteyaz, Steven Clarke, and Saiph Savage*. 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 (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 1164, 17 pages. doi:10.1145/3706598.3714008

[11] Mina Huh and Amy Pavel. 2024. DesignChecker: Visual Design Support for Blind and Low Vision Web Developers. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 142, 19 pages. doi:10.1145/3654777.3676369

[12] Jazette Johnson, Andrew Begel, Richard Ladner, and Denae Ford. 2022. Program-L: Online Help Seeking Behaviors by Blind and Low Vision Programmers. In *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–6. doi:10.1109/VL/HCC53370.2022.9833106

[13] Google Jules. 2025. https://jules.google/ Accessed: 2025-06-24.

[14] Raina Langevin, Ross J Lordon, Thi Avrahami, Benjamin R. Cowan, Tad Hirsch, and Gary Hsieh. 2021. Heuristic Evaluation of Conversational Agents. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 632, 15 pages. doi:10.1145/3411764.3445312

[15] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA.

[16] Microsoft. 2025. GitHub Copilot. https://github.com/features/copilot/

[17] Jakob Nielsen. 1995. 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. https://www.nngroup.com/articles/ten-usability-heuristics/

[18] OpenAI. 2025. ChatGPT. https://chat.openai.com/

[19] Venkatesh Potluri, Sudheesh Singanamalla, Firn Tieanklin, and Jennifer Mankoff. 2023. Notably Inaccessible – Data Driven Understanding of Data Science Notebook (In)Accessibility. In *Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility* (New York, NY, USA) *(ASSETS '23)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3597638.3608417

[20] Venkatesh Potluri, Priyan Vaithilingam, Suresh Iyengar, Y. Vidya, Manohar Swaminathan, and Gopal Srinivasa. 2018. CodeTalk: Improving Programming Environment Accessibility for Visually Impaired Developers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–11. doi:10.1145/3173574.3174192

[21] Alexander Repenning. 2011. Making programming more conversational. In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 191–194. doi:10.1109/VLHCC.2011.6070398

[22] Replit, Inc. 2025. Replit: From idea to app, fast. https://replit.com. Accessed: 2025-06-25.

[23] Ranjan Sapkota, Konstantinos I. Roumeliotis, and Manoj Karkee. 2025. AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenges. arXiv:2505.10468 [cs.AI] https://arxiv.org/abs/2505.10468

[24] Kevin M Storer, Harini Sampath, and M. Alice Alice Merrick. 2021. "It's Just Everything Outside of the IDE that's the Problem": Information Seeking by Software Developers with Visual Impairments. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 487, 12 pages. doi:10.1145/3411764.3445090

[25] Google Firebase Studio. 2025. https://firebase.google.com/docs/studio Accessed: 2025-06-24.

[26] Sarit Felicia Anais Szpiro, Shafeka Hashash, Yuhang Zhao, and Shiri Azenkot. 2016. How People with Low Vision Access Computing Devices: Understanding Challenges and Opportunities. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility* (Reno, Nevada, USA) *(ASSETS '16)*. Association for Computing Machinery, New York, NY, USA, 171–180. doi:10.1145/2982142.2982168

[27] W3C Web Accessibility Initiative (WAI). 2018. Understanding Success Criterion 4.1.2: Name, Role, Value. https://www.w3.org/WAI/WCAG21/Understanding/name-role-value.html. Accessed: 2025-06-24.

[28] W3C Web Accessibility Initiative (WAI). 2018. Understanding Success Criterion: Compatible. https://www.w3.org/WAI/WCAG21/Understanding/compatible.html. Accessed: 2025-06-25.

[29] W3C Web Accessibility Initiative (WAI). 2023. Understanding Guideline 2.1: Keyboard Accessible. https://www.w3.org/WAI/WCAG22/Understanding/keyboard-accessible.html. Accessed: 2025-06-25.

[30] World Wide Web Consortium (W3C). 2015. Authoring Tool Accessibility Guidelines (ATAG) 2.0. https://www.w3.org/TR/ATAG20/. Accessed: 2025-06-25.

[31] World Wide Web Consortium (W3C). 2023. Web Content Accessibility Guidelines (WCAG) 2.2. https://www.w3.org/TR/WCAG22/. Accessed: 2025-06-25.