Coding Non-Visually in Visual Studio Code: Collaboration Towards Accessible Development Environment for Blind Programmers

JooYoung Seo
School of Information Sciences, University of Illinois at Urbana-Champaign
Champaign, USA

Megan Rogge
Microsoft
Redmond, USA

ABSTRACT

This paper delineates a fruitful collaboration between blind and sighted developers, aiming to augment the accessibility of Visual Studio Code (VSCode). Our shared journey is portrayed through examples drawn from our interaction with GitHub issues, pull requests, review processes, and insider's releases, each contributing to an improved VSCode experience for blind developers. One key milestone of our co-design process is the establishment of an accessible terminal buffer, a significant enhancement for blind developers using VSCode. Other innovative outcomes include Git Diff audio cues, adaptable verbosity settings, intuitive help menus, and a targeted accessibility testing initiative. These tailored improvements not only uplift the accessibility standards of VSCode but also provide a valuable blueprint for open-source developers at large. Through our shared dedication to promoting inclusivity in software development, we aim for the strategies and successes shared in this paper to inspire and guide the open-source community towards crafting more accessible software environments.

CCS CONCEPTS

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

KEYWORDS

nonvisual programming, accessibility, integrated development environment, visual studio code

ACM Reference Format:
JooYoung Seo and Megan Rogge. 2018. Coding Non-Visually in Visual Studio Code: Collaboration Towards Accessible Development Environment for Blind Programmers. In *Proceedings of The 25th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

ASSETS, October 22–25, 2023, New York, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

An integrated development environment (IDE) is an application that conveniently provides essential functions for the entire programming process, including source editing, compiling and interpreting, and debugging. IDEs have become an essential tool for not only software developers but also STEM engineers and data scientists in many fields to efficiently manage their computing environments [3, 5, 7]. However, blind developers[1] are not able to take advantage of the many features that graphical user interface (GUI)-based IDEs offer [12]. For example, syntax highlighting, code autocompletion and autosuggestion, diagnostics and linting, variable watches and breakpoints are underutilized even among experienced blind programmers, and many blind developers are still working manually with simple text like Notepad, along with runtime and compile terminals [1, 2, 9]. Behind this problem are intertwined issues of accessibility and learnability. Because different IDEs use different architectures and have different levels of accessibility compliance, blind developers face a new learning curve each time they use an IDE. Blind developers also face the additional challenge of learning the non-visual workaround of accessing an IDE with a screen reader [9]. Although there is a community of blind programmers called Program-L [8] where blind programmers help and support each other, IDEs remain a daunting barrier for blind people.

These difficulties are a major socio-technical barrier to blind developers reaching their full potential in the computing field and to social and professional participation. From the perspective of the social model [10], which recognizes that an individual's disability may stem from structures and cultures that sociotechnically limit their access rather than from physical, sensory, cognitive, or emotional issues, we can see that IDE accessibility issues are no longer a group-specific problem that blind people must endure, but a collective task for the technology community to reduce barriers together. Specifically, to address these issues, blind and sighted developers need to work together to understand the challenges that blind developers face in using IDEs and then collaboratively find ways to address those challenges. This

[1]We use the identity-first language (i.e., blind people) instead of the person-first language (i.e., people with visual impairments or vision loss) when addressing this population, guided by the perspective of the National Federation of the Blind.

perspective is consistent with the "interdependent framework" [4, 6] that other accessibility researchers have advocated to move away from the dependency of accessibility on the individual with disabilities and instead view accessibility as a shared responsibility of people with and without disabilities and the environment surrounding them.

This paper is the empirical product of blind and sighted developers who have thought deeply about these issues and actively collaborated. We describe how the first author, who is blind, and the second author, who is sighted, have been working together to make the open source IDE Visual Studio Code (VSCode) non-visually accessible and what specific accessibility features have been implemented as a result of our collaboration. In the following sections, we start with some background on how our collaboration began, then present our methods and deliverables. Finally, we'll share some insights from our collaboration.

2 BACKGROUND: VISUAL STUDIO CODE AND ACCESSIBILITY

Visual Studio Code is a lightweight, free, and powerful open-source code editor[2] which runs on the desktop and on the web. It is available for Windows, macOS, and Linux. It has built-in support for JavaScript, TypeScript, and Node.js and a rich ecosystem of extensions for other languages and runtimes, such as C++, C#, Java, Python, PHP, Go, .NET, and more. Accessibility has been a core priority for VSCode since its inception. Among the many architectural elements of VSCode, the following, in particular, has contributed to its accessibility. First, VSCode is a cross-platform application built with the Chromium-based Electron Framework. In other words, VSCode is an application built using web technologies, which gives it the flexibility to follow web accessibility guidelines and respond to the accessibility of various screen readers and assistive technologies regardless of the operating system. Second, Monaco, the primary editor of VSCode, has its own screen reader compatibility mode, which is designed to be selectively turned on and off depending on the user's intent. Third, Microsoft's xterm.js terminal, used by VSCode, also provides a separate screen reader accessibility switch in accordance with the Web Accessibility Guidelines. Finally, VSCode is an open-source project where anyone can suggest and fix features on GitHub, and a daily insiders version is built so that real users can quickly use the alpha version and provide feedback to the developers, which in turn leads to a higher quality, user-centered stable version.

The accessibility benefits of VSCode and tips on how to take advantage of them have been shared among members of the Program-L mailing list, a community of blind programmers. In addition, due to its growing popularity among blind programmers, there has been a recent spate of research and development of accessible plug-ins based on VSCode [11, 15]. Nevertheless, the fact that VSCode is accessible compared to other IDEs does not necessarily mean that it is easy for blind programmers to use. For example, there is still a constant stream of questions on Program-L about VSCode, not only about its basic usage, but also about features that have already been made accessible in VSCode, such as the terminal, debugging, and the Jupyter Notebook extension, which suggests that many blind programmers are often frustrated by the tricky usability of VSCode accessibility. The following section describes how the authors of this paper collaborated to address this usability issue of VSCode accessibility.

[2]In this paper, the terms integrated development environment and code editor are used interchangeably.

3 METHODS

3.1 Author Profiles and Collaboration Context

The first author of this paper is blind with only light perception, currently working as an assistant professor in the School of Information Sciences at the University of Illinois at Urbana-Champaign. At the university, he teaches introductory data science courses using R and Python to undergraduate and graduate students. As a lifelong non-visual programmer, he has experience with a variety of IDEs, including Visual Studio, Eclipse, and Net Bean, and text editors such as Emacs/Emacspeak, VIM, and NotePad++, on Linux, Mac, and Windows operating systems, using a variety of screen readers (e.g., JAWS, NVDA, Narrator, VoiceOver, and Orca) and refreshable braille displays. He is a certified professional in accessibility core competencies (CPACC) from the International Association of Accessibility Professionals and has contributed code to a number of open-source data science projects to improve screen reader accessibility, including RStudio IDE Server and the web-based data science dashboard Shiny, reproducible technical publishing systems (e.g., R Markdown, bookdown, and Quarto), and the data table package gt. He is also a member of Program-L. In this community, he has experienced first-hand the challenges that blind programmers face in using IDEs and how they overcome them by interacting with other blind programmers and participating in discussions. To improve these community-wide challenges, he created his first issue on the Microsoft VSCode public GitHub site on May 31, 2020, and has since created a total of 164 contributions (87 issues; 76 post comments and mentions; 1 pull request) to actively suggest usability improvements for blind programmers in VSCode and interact with other open source developers[3].

The second author is a VSCode software engineer. She has worked on the product since graduating from the University of North Carolina at Chapel Hill in 2020 with the highest distinction and highest honors for her research and work with Dr. Gary Bishop on semi-automated gaming for users with a wide range of disabilities. About 10 months ago, Megan requested to take over responsibility for the product's accessibility. Since then, she has been working closely with JooYoung and the community to understand accessibility issues and collaborate on solutions.

[3]see online appendix at https://github.com/jooyoungseo/assets2023_vscode

3.2 Co-Design and Expert Review

Our collaborative approach utilized the strategies of co-design and expert review. The co-design methodology fosters a joint creation process between the user and developer, enabling the developer to grasp the user's requirements and, in turn, develop a product aligning with these needs [14]. In this framework, JooYoung acted as an expert, given his multi-faceted role as a daily VSCode user, an experienced open-source contributor, a data science educator, and an accessibility professional. He outlined his varied computing experiences to Megan and swiftly assessed his accessibility patches. Our communication began asynchronously via GitHub, debating on issues and potential solutions. Following a few weeks of this pattern, we mutually agreed that scheduled meetings could prove more efficient and productive. JooYoung's wealth of ideas and insights complemented Megan's eagerness to learn and her drive to enhance the product's accessibility. In these sessions, JooYoung demonstrated his use of VS Code by sharing his screen on Zoom, posing queries, and suggesting alterations. Conversely, Megan provided her insights, questioned various aspects, and noted down bugs or features requiring attention. These exchanges facilitated Megan's understanding of JooYoung's usage of VS Code and enabled JooYoung to comprehend the product components, which could otherwise remain confusing or undiscovered.

Despite the implementation of regular meetings, asynchronous communication via GitHub and email persisted. Megan regularly composed follow-up emails encapsulating their meeting discoveries prior to circulating them to the entire VSCode team. JooYoung further scrutinized these issues, providing comments if anything was overlooked or during the fix-testing process.

4 CO-DESIGNED DELIVERABLES

While nearly all VS Code accessibility fixes and features within the past year are products of this collaboration, below are several of the highlights.

4.1 Terminal Buffer

As discussed in Section 2, xterm.js, the terminal UI utilized by VSCode, incorporates a screen-reader accessibility mode for blind people. However, a discernible gap emerged between accessibility (the ability to access information) and usability (the convenience of use), which led to recurring concerns among blind programmers.

Consider the following scenario: you type and execute the command `echo hello; echo world;` in the terminal. You will observe `hello` and `world` as two separate lines of output. The existing accessibility mode of xterm.js presented this content through a screen reader using an aria-live alert and permitted a line-by-line review of the terminal output history with the `Ctrl+UpArrow` and `Ctrl+DownArrow` keys. This worked well for short and simple outputs, but for lengthy outputs with intricate error messages or computational results, a swift speech-to-text message was insufficient for capturing substantial information in human working memory.

An additional concern is that `Ctrl+Up/DownArrow` navigation keys, designed to review terminal history, delivered the entire contents of the focused line to the screen reader as a single object. This made a detailed examination of terminal contents on a character or word basis challenging. Blind users had to switch the reading mode using the screen reader's virtual cursor (i.e., browse mode in NVDA; QuickNav mode in VoiceOver) to review the terminal content more thoroughly. To resume terminal input, they had to disable the virtual cursor and return to forms mode (focus mode in NVDA; QuickNav off in VoiceOver), leading to significant inconvenience.

JooYoung initiated a discussion on the official Microsoft VSCode GitHub page, bringing attention to these issues and proposing solutions (see microsoft/vscode#98918: Terminal output div container should be more accessible for screen readers). Megan, meanwhile, developed terminal shell integration, a feature allowing VS Code to comprehend terminal activities, facilitating user-friendly command navigation, command output copying, and more. JooYoung demonstrated that the terminal buffer remained inaccessible for screen reader users, as it didn't support arrow key navigation. He proposed that the output view's accessible experience be integrated into the terminal. Upon discussing with a colleague, Megan incorporated the same underlying component into the terminal, making the previously inaccessible terminal buffer navigable via arrow keys for blind users. More specifically, he suggested replacing the terminal output with a text editor buffer supporting standard arrow-key navigation. The implementation, requiring over a year of technical experimentation and collaborative testing, yielded fruitful results. Initial efforts to redirect the terminal output web container, designated as "list", to aria "document" or "textbox" landmarks proved unsatisfactory due to varying screen reader and platform support levels for aria. The terminal output was then converted into a text area with "contenteditable" and "read only" attributes, which also did not gel with the screen reader's speech buffer. Eventually, we created a separate accessible terminal buffer by transferring the terminal output to VSCode's native Monaco editor, ensuring optimal accessibility and usability for all blind users on all platforms and screen readers. This feature, well-received by many blind users in the Program-L community, was officially introduced in the VSCode stable version 1.75.

In Figure 1, a screenshot depicts the terminal accessible buffer in action. A screen reader is shown focusing on an error message from a task terminal and audibly announcing: "[watch-client] [12:41:01] Error: /Users/meganrogge/Repos/vscode/vscode/src/vs/workbench/contrib/accessibility/browser/accessibilityContributions.ts(198,63): ')' expected.". Users can navigate the displayed content using standard arrow keys without having to switch between different navigation modes.

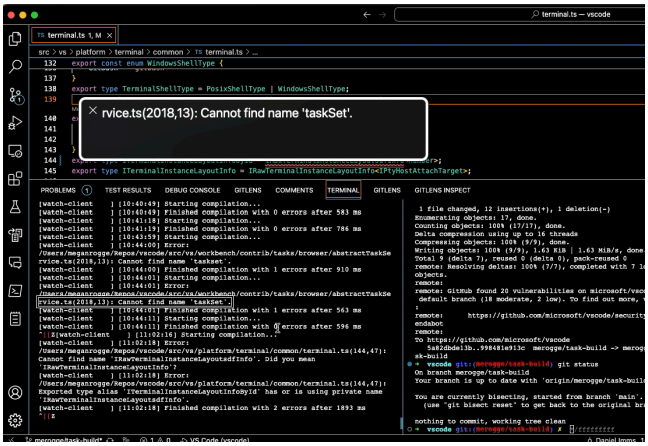

Figure 1: The terminal accessible buffer.

4.2 Git Diff and Audio Cues

Git has been around for decades as a version control tool like SVN, but its popularity has really taken off with the rise of open-source social coding platforms based on Git, such as GitHub and GitLab. Naturally, there have been many personal and social needs for blind people to utilize Git in collaborative environments. Git is originally a Unix-based command-line tool, so in terms of accessibility, blind people can use a screen reader to fully utilize Git in a terminal. However, since Git has over 100 core Git commands, and the number of possible combinations could be in the millions, using Git via the command line takes a lot of effort and time to become proficient. In response, various tools have emerged that allow users to utilize Git as a GUI, and VSCode is a very popular IDE that supports a collaborative environment using Git.

Git's `git diff` feature enables users to track changes and compare differences between files or commits in an asynchronous collaborative setting. Using this command, new additions are marked in green with a `+` prefix, while deletions appear in red with a `-` prefix. VSCode ensured that the `git diff` feature is accessible to screen reader users. When users had the desired files or commits open for comparison, they could quickly navigate to the differences using the `F7` key (Go to Next Difference) and `Shift+F7` (Go to Previous Difference). These differences were prefixed with a `+` or `-` sign to denote additions or deletions, respectively. Of course, this approach was fine from an accessibility standpoint, but there was room for improvement in terms of usability and convenience for blind users. For example, visual affordances like color coding and `+-` signs in `git diff` allowed sighted people to skim quickly, but blind people had to listen to additional speech prefixes, pronounced + (plus) and - (dash), serially and wait for information before each change. Furthermore, depending on the punctuation pronunciation settings of the screen reader, the +- sign could be omitted to the screen reader.

To ameliorate this, JooYoung suggested adding non-visual, non-speech, and audible affordances to `git diff` in addition to `+-` signs, so that blind people can hear and understand them easily (see microsoft/vscode#147226). Audio cues, often referred to as "earcons", are non-speech sound effects that enhance accessibility. While VSCode and Microsoft's Visual Studio have only recently started supporting them, their importance in non-visual programming was highlighted decades ago by TV Raman, a blind computer scientist. He introduced the concept of earcons as a counterpart to visual icons when he developed Emacspeak [13]. For example, audio cues allow the editor to quickly recognize if the current line of code contains an error or a warning, instead of just saying "error" or "warning" verbally, the editor will read out the unique sound associated with the error or warning. These sounds can also be delivered in parallel with text-to-speech information from a screen reader, allowing blind programmers to quickly perceive the context of the code, similar to the benefits of quickly scanning code with different color coding for those who receive visual feedback on code with their eyes. JooYoung had several Zoom meetings with Megan and Amnon Freidlin (Microsoft's sound designer), and through an iterative process, finalized the three audio cues used in the `git diff` context. These were the diff line Inserted sound, which is heard when something new is added (`+`), the diff line Deleted sound, which is heard when something existing is removed (`-`), and the diff line Modified sound, which is heard when something existing is modified (`+-`, `-+`). Our success came with some trial and error. For example, an early problem was that the Diff Line Inserted and Diff Line Deleted sounds had a similar range and texture, making it difficult to distinguish between them. JooYoung realized that this was a common complaint in Program-L beyond his personal experience, so he worked with the sound designer to test and finalize a sample file that was as self-explanatory as possible and didn't interfere with the sound of the screen reader speech. Of course, we had to leave the potential issue of the static audio cues we chose not being able to adequately accommodate users with hearing impairments in certain ranges as a future work in progress, but this feature greatly improved the usability of our non-visual programming.

4.3 Verbosity Settings and Help Menus

JooYoung created issues pointing out places where minor tweaks to the order or content of an aria-label could yield massive productivity improvements for screen reader users. Megan fixed some such instances and pointed team members toward others, providing guidance about best practices going forward. Megan started self-hosting with a screen reader shortly after this in order to proactively identify other problems. She felt overwhelmed by the noise and noticed some content was repeated ad nauseum, so created an issue and sought the feedback of JooYoung, who suggested that screen reader verbosity settings remedy this and a similar approach could be applied to VS Code's aria content. Additionally, JooYoung shared that while it was helpful to meet and learn

about the new features via our meetings, most screen reader users did not have this luxury. Megan and her colleague, Daniel, brainstormed about a discoverable way for screen reader users to find out about terminal features. Upon terminal focus, an aria-label conveyed how to access the terminal's accessibility help menu. To reduce noise, this hint could be disabled with a verbosity setting. Since then, help menus and verbosity settings have been added for the Copilot inline and panel chat, notebook, and other features. For example, the terminal accessibility help menu contains important information for screen reader users such as commands to run like "Run Recent Command (Ctrl+R)" (Figure 2). A screen reader user can use arrow keys to read the content line by line, character by character.

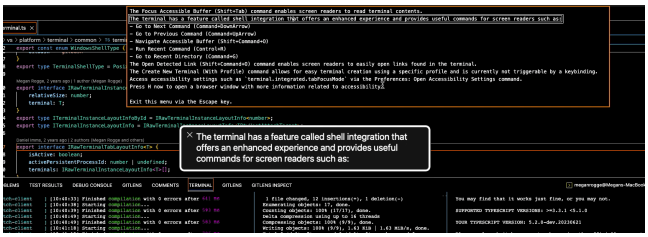

Figure 2: The terminal help menu

4.4 Accessibility Testing Initiative

JooYoung consistently provided feedback on GitHub, highlighting issues with both old and new features in VS Code. His active engagement and insights were pivotal in spotlighting an overlooked area. Megan, a key accessibility member of the VSCode team, noticed that despite testing new features on every platform - MacOS, Linux, and Windows at the end of each month before a release, the screen reader experience had been neglected. Inspired by JooYoung's observations, Megan advocated for a new protocol: after a feature is released, it will be tested with screen readers in the next iteration. Additionally, she initiated retroactive tests on features to rectify this historical oversight.

5 DISCUSSION AND CONCLUSION

In our collaborative journey, blending the expertise of both sighted and blind developers, we've unearthed pivotal insights about open-source accessibility. Our endeavors with Visual Studio Code serve as a case in point. Megan and JooYoung's livestream seminar accentuated the profound impact of merging accessibility considerations with open-source development.

Open-source platforms are foundational in the tech realm. Our reach and influence cascade into multiple offshoot applications. Hence, embedding accessibility in these parent projects can have a magnified effect, promoting inclusivity across numerous derivative platforms. The participatory nature of open-source projects, welcoming feedback from a diverse array of users, is both a strength and a challenge. JooYoung's collaboration, while external to Microsoft, highlights this open engagement. Yet, a pertinent concern is the potential marginalization of voices unfamiliar with platforms like GitHub. For open-source teams, this underscores the necessity to proactively engage with and seek feedback from these underrepresented communities. Such engagement is not just about hearing, but understanding and integrating feedback. The synergy between Megan and JooYoung exemplifies the potential outcomes when such engagements are cultivated. A recurring theme from our experience is the importance of proactive accessibility considerations. Post-design modifications often present challenges, emphasizing the need for early integration of accessibility measures. Drawing these threads together, our co-design experience underscores the imperative of fostering an inclusive ethos in open-source development. By ensuring a platform that is receptive to diverse voices, we can move closer to a universally accessible coding ecosystem.

ACKNOWLEDGMENTS

We extend our gratitude to Program-L, an online community of blind programmers, for their invaluable feedback and testing of VS Code, as well as their insightful accessibility suggestions. We also appreciate the VS Code team for their commitment to accessibility. Special thanks to Isidor Nikolic, Kai Maetzel, and Daniel Imms for their dedication and invaluable insights; to Raymond Zhao and Roberto Perez for enhancing the site's accessibility; to José Vilmar Estácio de Souza, Amnon Freidlin, Marie Robbins, and Gino Scarpino for their diligent testing and collaboration.

REFERENCES

[1] Khaled Albusays and Stephanie Ludi. 2016. Eliciting Programming Challenges Faced by Developers with Visual Impairments: Exploratory Study. In *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '16)*. Association for Computing Machinery, New York, NY, USA, 82–85. https://doi.org/10.1145/2897586.2897616

[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 (ASSETS '17)*. Association for Computing Machinery, New York, NY, USA, 91–100. https://doi.org/10.1145/3132525.3132550

[3] Raghavendra Rao Althar and Debabrata Samanta. 2021. Building Intelligent Integrated Development Environment for IoT in the Context of Statistical Modeling for Software Source Code. In *Multimedia Technologies in the Internet of Things Environment*, Raghvendra Kumar, Rohit Sharma, and Prasant Kumar Pattnaik (Eds.). Springer, Singapore, 95–115. https://doi.org/10.1007/978-981-15-7965-3_7

[4] Cynthia L. Bennett, Erin Brady, and Stacy M. Branham. 2018. Interdependence as a Frame for Assistive Technology Research and Design. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '18)*. Association for Computing Machinery, New York, NY, USA, 161–173. https://doi.org/10.1145/3234695.3236348

[5] James H. Cross and T. Dean Hendrix. 2007. jGRASP: An Integrated Development Environment with Visualizations for Teaching Java in CS1, CS2, and Beyond. *Journal of Computing Sciences in Colleges* 23, 2 (Dec. 2007), 170–172.

[6] Lilian De Greef, Dominik Moritz, and Cynthia Bennett. 2021. Interdependent Variables: Remotely Designing Tactile Graphics for

an Accessible Workflow. In *The 23rd International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, Virtual Event USA, 1–6. https://doi.org/10.1145/3441852.3476468

[7] Jan Janssen, Sudarsan Surendralal, Yury Lysogorskiy, Mira Todorova, Tilmann Hickel, Ralf Drautz, and Jörg Neugebauer. 2019. Pyiron: An Integrated Development Environment for Computational Materials Science. *Computational Materials Science* 163 (June 2019), 24–36. https://doi.org/10.1016/j.commatsci.2018.07.043

[8] 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)*. IEEE, 1–6. https://doi.org/10.1109/VL/HCC53370.2022.9833106

[9] S. Mealin and E. Murphy-Hill. 2012. An Exploratory Study of Blind Software Developers. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, Innsbruck, 71–74. https://doi.org/10.1109/VLHCC.2012.6344485

[10] Mike Oliver. 2013. The Social Model of Disability: Thirty Years On. *Disability & Society* 28, 7 (Oct. 2013), 1024–1026. https://doi.org/10.1080/09687599.2013.818773

[11] Venkatesh Potluri, Maulishree Pandey, Andrew Begel, Michael Barnett, and Scott Reitherman. 2022. CodeWalk: Facilitating Shared Awareness in Mixed-Ability Collaborative Software Development. In *The 24th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, Athens Greece, 1–16. https://doi.org/10.1145/3517428.3544812

[12] 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*. ACM, Montreal QC Canada, 1–11. https://doi.org/10.1145/3173574.3174192

[13] T. V. Raman. 1996. Emacspeak—a Speech Interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '96)*. Association for Computing Machinery, New York, NY, USA, 66–71. https://doi.org/10.1145/238386.238405

[14] Elizabeth B.-N. Sanders and Pieter Jan Stappers. 2008. Co-Creation and the New Landscapes of Design. *CoDesign* 4, 1 (March 2008), 5–18. https://doi.org/10.1080/15710880701875068

[15] Bernhard Stöger, Klaus Miesenberger, Walther Neuper, Makarius Wenzel, and Thomas Neumayr. 2022. Designing an Inclusive and Accessible Mathematical Learning Environment Based on a Theorem Prover. In *Computers Helping People with Special Needs (Lecture Notes in Computer Science)*, Klaus Miesenberger, Georgios Kouroupetroglou, Katerina Mavrou, Roberto Manduchi, Mario Covarrubias Rodriguez, and Petr Penáz (Eds.). Springer International Publishing, Cham, 47–55. https://doi.org/10.1007/978-3-031-08648-9_7