Node-Read: A Visually Accessible Low-Code Software Development Extension

Lachlan Anderson
Monash University
Clayton, Victoria, Australia

Briana Barker
Monash University
Clayton, Victoria, Australia

Alice Reid
Monash University
Clayton, Victoria, Australia

Kaijie Lin
Monash University
Clayton, Victoria, Australia

Hourieh Khalajzadeh
Deakin University
Burwood, Victoria, Australia

John Grundy
Monash University
Clayton, Victoria, Australia

ABSTRACT

Low-code software development environments are reliant on spatial and graphical user interfaces. As a result, many of these tools are in some way inaccessible to the visually impaired, and very few of these tools are built with visual accessibility in mind. In this paper, we evaluate the accessibility of existing low-code Integrated Development Environments (IDEs), for persons with partial or distorted vision. The aim of this study is to motivate making citizen/end-user software development accessible for users who are reliant on screen readers. We conducted a preliminary review of several low-code development environments which were open source and had a large existing user base, and identified that browser-based low-code IDEs did not integrate well with screen reader software. An extension of an open-source software, Node-RED, was created, as it was found to be suitable to our selection criteria. The extension, referred to as "Node-Read", focuses on improving compatibility with JAWS and NVDA screen readers. Node-Read's keyboard shortcuts, along with their inclusion in critical user documentation, were reported by study participants to be helpful in the basic operation of the software.

CCS CONCEPTS

• **Software and its engineering → Software notations and tools**; • **Human-centered computing → Accessibility systems and tools**.

ACM Reference Format:
Lachlan Anderson, Briana Barker, Alice Reid, Kaijie Lin, Hourieh Khalajzadeh, and John Grundy. 2022. Node-Read: A Visually Accessible Low-Code Software Development Extension. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3550356.3561591

1 INTRODUCTION

There exists the underlying assumption that software developers are sighted [11]. As such, standard development practices rely heavily on the visual presentation of code within an IDE [11] and when IDEs do contain accessibility tools, visually impaired developers are often unaware of them [17]. Fewer developers with vision impairments are entering the field due to accessibility issues faced when attempting to learn and practice programming skills [3, 11, 29]. Those who are building software solutions frequently fall short of the accessibility needs of their users [13] meaning that they are not actively building systems to be compatible with critical assistive technologies like screen readers [17]. Providing visual accessibility involves making a platform navigable and usable for people with visual impairments [13]. However, there are a large number of different types of visual impairments, and their resulting functional impacts can vary drastically [31]. There exists a set of modern web content accessibility guidelines (WCAG) [30] that developers are encouraged to follow in order to create universally accessible and inclusive experiences [13]. However, these guidelines still fail to bridge the accessibility gap, and cannot be used as a standalone checklist to ensure accessibility for visually impaired users [21].

With the rise of the citizen developer, there is a growing demand to make software development more accessible for those without a technical background [1, 10, 25]. A key roadblock to the entry of many into the field is that of coding [25]. The issue predominantly lies in the intricacies of learning new coding languages and practices. A solution is to abstract away the details of the underlying source code through the use of low-code integrated development environments [15]. Such platforms often use graphical models to represent underlying code, in a more technically accessible format [11]. This is often achieved with simple visuals, with functions represented as shapes that can be connected to form larger systems [12, 24, 26]. When developing within low-code environments, users can think less about *how* something is achieved and focus more on *what* they want their system to do [1]. In order to ensure that users are able to benefit from the abstraction that low-code IDEs provide, these platforms must provide a level of visual accessibility. Previous studies which explore visual accessibility for IDEs generally keep within the confines of full-code platforms [6]. There is limited research that explores both visual accessibility and low-code IDEs with a focus on improving the usability for those with visual impairments, especially those requiring screen readers. This paper explores how an existing low-code IDE can be extended to improve its accessibility for those who rely on screen readers.

By reviewing multiple commercially used and open source low-code IDEs, and attempting to complete their respective tutorials while using a screen reader, we identified that browser based low-code IDEs were often incompatible with common screen reader software, as some keyboard shortcuts would be intercepted prior to reaching the IDE. We selected to extend Node-RED, a widely used open source browser based Low-Code development environment. We used NVDA and JAWS screen readers to evaluate accessibility within these development environments as they are widely used within the visually impaired community. We found that our extension, Node-Read, improved participants ability to interpret the system output, including audio feedback and text labels. Participants reported that keyboard shortcuts made the processes such as adding a node easier, and that Node-Read key-board shortcuts were helpful to the process.

2 RELATED WORK

The largest hurdle for blind developers is the inaccessibility of debuggers and IDEs [17], formulating high-level overviews of the code, finding targeted information and interpreting visual elements. Several studies identified that the debugging process is a major hurdle for visually impaired developers [2]. Wicked Audio Debugger (WAD) [28] and CodeTalk [20], are plugins for Visual Studio, which replace the existing debugging process of the IDE. In WAD's study, participants were able to comprehend 86% of a program's behaviour through audio alone, which suggests audio based software development is not only possible, but can enable visually impaired developers to operate at a similar level to sighted developers when implemented correctly. Struct-Jumper [6] found that relationships present in Object-Oriented programming languages were frequently represented spatially, and were therefore difficult to interpret through screen readers alone [17]. Therefore, Struct-Jumper attempts to better visualise a Java class as a tree view to allow screen reader users to understand how various methods relate with an easier to interpret representation [6]. While there are many examples of tools aimed at helping screen reader users interpret coding specific elements, keyboard input is predominant. Spoken Java [7] has attempted to provide alternatives, for code-writing speech input. This sparks potential direction of development towards speech based interaction, which can be used by both sighted and visually impaired developers. As software development is largely team based, common software and development techniques make collaboration, and by extension development itself, easier [9]. Audio Programming Language [27] is a programming language designed by and for blind users. Unlike similar tools, this research intends to develop an environment independent of a visual user interface. Their study suggests that languages can be constructed to fit the mental models of blind learners to help them to enter to the programming world.

Milne et al. [18] identified key accessibility barriers for visually impaired children when using existing blocks-based environments. Through interviewing a teacher of the visually impaired and formative studies on a touchscreen blocks-based environment, they explored options and distilled their findings on usable touchscreen interactions into guidelines for designers of blocks-based environments. However, we found no research conducted on evaluating the accessibility of low-code development environments and understanding how they can be improved for screen reader users. The minimal overlap we found in the two topics of visual accessibility in development and low/no code development, suggested there may be a significant gap in the types of research currently being conducted. The creation of development tools for people who require screen readers is still an afterthought in many circumstances [8].

3 APPROACH

The goal of our study is to allow more people with visual impairments to use software development tools. Currently, those with visual impairments use screen readers to read code aloud [5]. When using these tools on applicable IDEs, visually impaired programmers have been shown to comprehend code to the same degree as their sighted peers [4]. However, there appears no evidence to show that low-code development environments allow screen reader users to comprehend software development. Our overarching research question is:

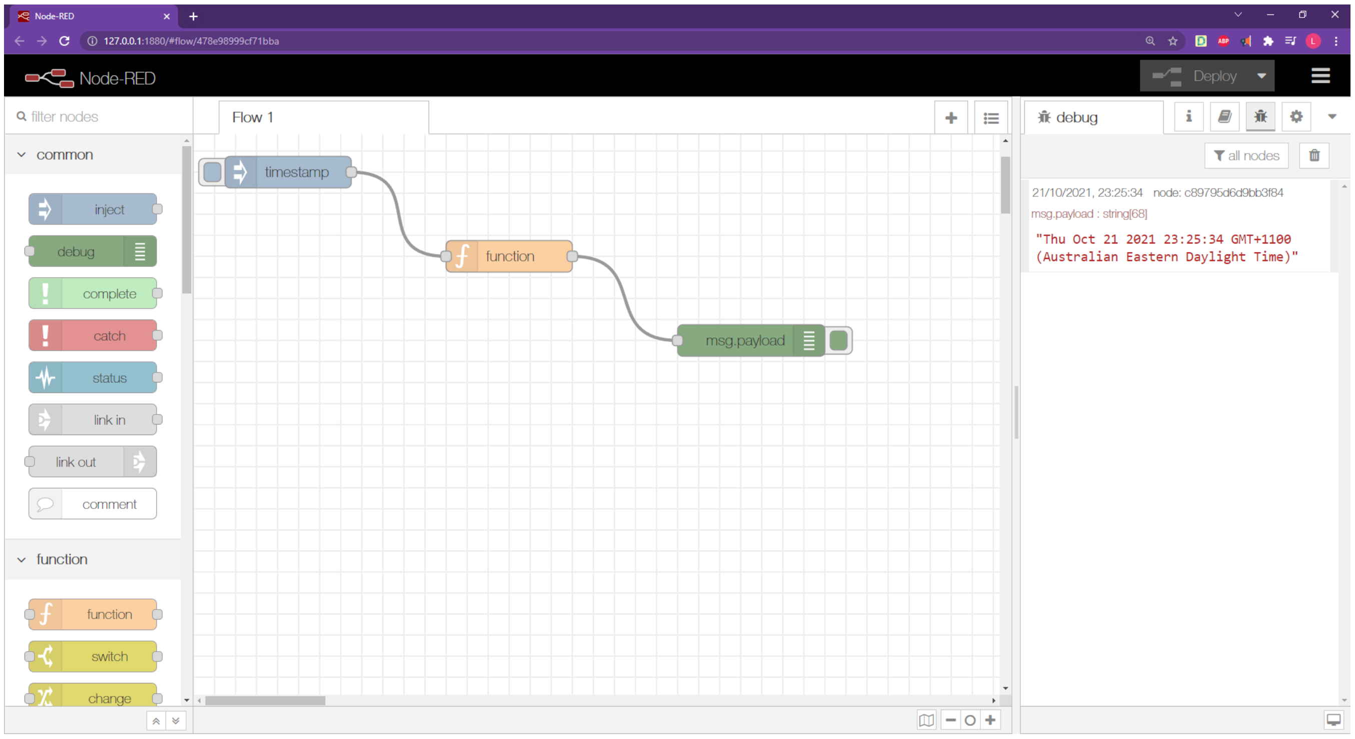
RQ: Can we extend a low-code IDE to meet the accessibility needs of people requiring screen readers?

3.1 Selecting an Extendible IDE

Tools developed through research are at risk of not being maintained once the associated research concludes. For example CodeTalk was listed as pre-release in 2017, and has not yet been published [20], nor has it been updated to be compatible with IDEs following Visual Studio Code 2017. As such, we further investigated the quality of low-code IDEs in the market, evaluating both open source and commercial solutions, to create and test a low-code IDE that was accessible for those who are reliant on screen readers. This presented two options: Find 1) a commercially available low-code IDE and create a system that translates the users screen-reader input into normal navigation actions; 2) an open source low-code IDE with exposed code that we could directly edit to make more compatible with screen readers.

While some commercial products seemed promising, finding APIs that would allow the kind of interactivity we were looking for required much more information than we could find. Additionally the basic system sat behind a paywall and there would be minimal, if any, support to build a compatible extension. Thus, we decided to opt for the open-source approach for the straightforwardness and lower bar of entry. In order to find the best suited tool from a limited pool of selection, the open source repositories were evaluated based on their usage, maintainability, ability for accessibility improvement and how active the development community is.

Many platforms were evaluated, most notably: App inventor [12] (Git Stars = N/A), a visual programming environment for building apps for Android and iOS devices, includes a simplified GUI, primarily targeting young students. Rintagi [26] (Git Stars = 213), a low-code enterprise application builder focuses on mobile apps which utilises complex tools to allow users to quickly build applications. Node-RED [24] (Git Stars = 13,200), an open-source low-code browser-based programming platform for event-driven applications, uses a GUI consisting of inter-connectable nodes. Node-RED



```
21/10/2021, 23:25:34   node: c89795d6d9bb3f84
msg.payload : string[68]
"Thu Oct 21 2021 23:25:34 GMT+1100
(Australian Eastern Daylight Time)"
```

Figure 1: An example of Node-RED in use

was a stand out for several reasons, including known accessibility issues that could be improved through the implementation of WAI-ARIA guidelines. Figure 1 shows an example of Node-RED in use. Additionally, while complex platforms such as Rintagi and App inventor can be exceptionally useful, they are used by significantly fewer people, and have a higher technical barrier to entry, which reduces the potential impact which research in this area could have. Node-RED has a large body of both users and contributors and if merged into the main repository, the extension can be actively maintained after the research concludes, potentially benefiting future visually impaired users.

3.2 Design

Through evaluating Node-RED against the WCAG [30], as well as testing the program using a screen reader, we identified areas which may be difficult for screen reader users. Key functional issues highlighted in the analysis included:

Keyboard Shortcuts: We found that keyboard shortcuts were being intercepted by the screen reader, making them have no effect on the Node-RED work space. Through evaluation and testing, we mapped out alternatives for some of the keyboard shortcuts which were being intercepted. The scope of the changes was limited to only include shortcuts which could be used within the 'First Flow' tutorial [32].

Labels: Descriptive labels for some of the buttons appeared when the cursor hovered over the button. However, they do not persist when the user tries to hover over the label itself in order to get the screen reader to read it. Therefore, the value those labels provides is lost due to inaccessibility.

Wiring Nodes: The process of wiring nodes together could only be completed through the use of a mouse/touch-pad. Additionally, there were no indications as to whether the user had begun creating a wire and whether or not the wire creation was successful. There was also no auditory indication or confirmation of which nodes were just connected.

Workspace navigation: The workspace on Node-RED is expansive, and changes size dynamically as the user scrolled around. If the user was not keeping track of the relative position of the nodes they had already placed in the workspace, a few accidental scrolls could remove the nodes from the screen view, thus making it difficult to continue to add nodes to the current process or make edits.

Node information: When hovering over added nodes in the workspace, only the text on top of each node is provided to the screen reader. This is not always helpful in quickly identifying the nodes. For example, the default text on an 'inject node' reads 'timestamp', which, to the unfamiliar, would not indicate that this was an inject node. Additionally, nodes have input and/or output terminals and there is no way for the screen reader to identify which type of terminal a node has and whether or not it is connected to anything.

3.3 Feasibility Analysis

We forked the Node-RED GitHub repository [24] to test the feasibility of an extension and gain an understanding of the code base.

By identifying where and how keyboard shortcuts were being managed, we discovered that the in-built web-browser functionalities of the screen reader were blocking several Node-RED keyboard shortcuts as they overlapped with web standard keys. This was a result of web-standard key combinations that remained reserved for higher level navigation. We also discovered that not all keyboard shortcuts listed within the code were easily discoverable for users on the Node-RED website, suggesting that some existing shortcut combinations are not being utilised to their full capacity. As such, we built Node-Read [23], an extension that focuses predominantly on improving keyboard shortcut accessibility. The key focus of the extension was to surface existing shortcuts to the user that would work with the NVDA screen reader. As the screen reader blocked some common keyboard combinations, we found there were unique key combinations that would not be intercepted. Therefore, some existing shortcut functionalities were translated into different shortcut combinations in order to bypass the interception issue. Some additional shortcuts were also added to assist in navigating nodes. Shortcuts were not added for wiring nodes together or opening their property panes.

3.4 'First Flow' Tutorial Expansion

The Node-RED 'first flow' tutorial [32] accurately represents the first process a new user would undertake while learning the software. We duplicated and amended the tutorial to include references to the new keyboard shortcuts. As the initially available tutorial does not indicate which keyboard shortcuts can be used throughout the process, the tutorial was re-written to include these options. This change included both the addition of existing keyboard shortcuts to the tutorial, as well as the addition of keyboard shortcuts added in Node-Read.

4 EVALUATION

As we wanted to understand how visually impaired users new to Node-RED would interact with the system, we analysed the 'first flow' tutorial. A user's first interaction with a product generally determines how likely they are to use it again [22]. A negative first experience would be a potential deterrent from using the software in the future. Hence, studying the accessibility of this tutorial is critical to understanding the barriers in place which prevent visually impaired developers from learning to use this technology. The tutorial relies heavily on visual interaction, using a combination of clicking, dragging and pasting to complete tasks. It walks users through seven key interactions: 1) Adding a node to the workspace; 2) Wiring two nodes together; 3) Deploying a process; 4) Deleting a wire; 5) Navigating between nodes; 6) Adding code to a node; 7) Finding output in the debug panel. The instructions for the Node Read tutorial were revised to use a variety of keyboard shortcuts, both pre-existing and newly added combinations, to navigate the tutorial flow.

Our study involved walking participants through both the original and amended versions of the tutorial on Node-RED and Node-Read respectively. Participants filled in a questionnaire after each run-through and their feedback was collated and analysed. We randomised participants (1:1) to group A and group B, with group A using the Node-Read software first, while group B used Node-RED first. Sighted participants completed the study with a visual overlay to obscure the screen, as shown in Figure 2. The participants were given an opportunity to familiarise themselves with the screen reader interactivity if needed and were then guided through the tutorials, moving at their own pace. If participants struggled to navigate to the correct area of the screen, the researcher suggested areas to move towards so as to ensure that participants were not hindered by an inability to operate the screen reader effectively.

Participants were instructed on how to perform the tasks, using the instructions from the existing and amended 'first flow' tutorial. Once the participants had completed the full set of tasks, they filled out a questionnaire about their experience using the software and then repeated this process for the other software version, completing the same set of tasks using different methods.The questionnaire was designed to gather an understanding of the usability and overall experience of using Node-RED and Node-Read while completing the tutorial. The participants were asked: (1) Whether they found each functional step of the tutorial easy; (2) Whether they would be able to do the tutorial unguided afterwards; (3) How helpful, intuitive and memorable were the shortcuts used in Node-Read; (4) What (if any) roadblocks did they face while completing the tutorial; (5) How functionally complete the system felt; (6) Were there any parts of the application they found difficult to navigate to in particular; and (7) Whether there were any surprisingly helpful elements of the system. The gathered data was de-identified by transcribing any recordings before deletion, and removing any identifiable details. Data was then analysed to identify key findings and areas of improvement.

5 RESULTS

Participants: Twelve participants with varying technical backgrounds participated in our study. Participants' technical skills ranged from completely non-technical to software developer. One participant had a visual impairment which required the use of a screen reader for their development work. The sighted participants had never used a screen reader before and were not visually impaired. Small number of participants is typical in studies involving individuals with specific impairments [14, 16, 19]. None of the participants had used Node-RED prior to the study. Half the participants were randomly selected to complete the first tutorial on Node-Read and the other half began with Node-RED.

Simulation of Visual Impairments: Sighted participants completed the study with an translucent white visual overlay, calibrated to make reading text not possible, yet still provide the visual input to recognise shapes and distinctive colours. This effect was created using Screen Dragons 2, which allows this visual overlay to be calibrated quickly and clicked through. When configured in this way, the Node-Read or Node-RED instance was viewed through the overlay, while the mouse was not affected. This replicates the NVDA screen readers ability to provide audio feedback of the mouse icon when the icon changes.

Set Up: The study was conducted remotely, with the researcher hosting local versions of both Node-RED and Node-Read for the participants to interact with using Zoom's 'screen sharing' and 'remote control' functionalities. The research environment was set up to use the NVDA screen reader for all experiments to ensure

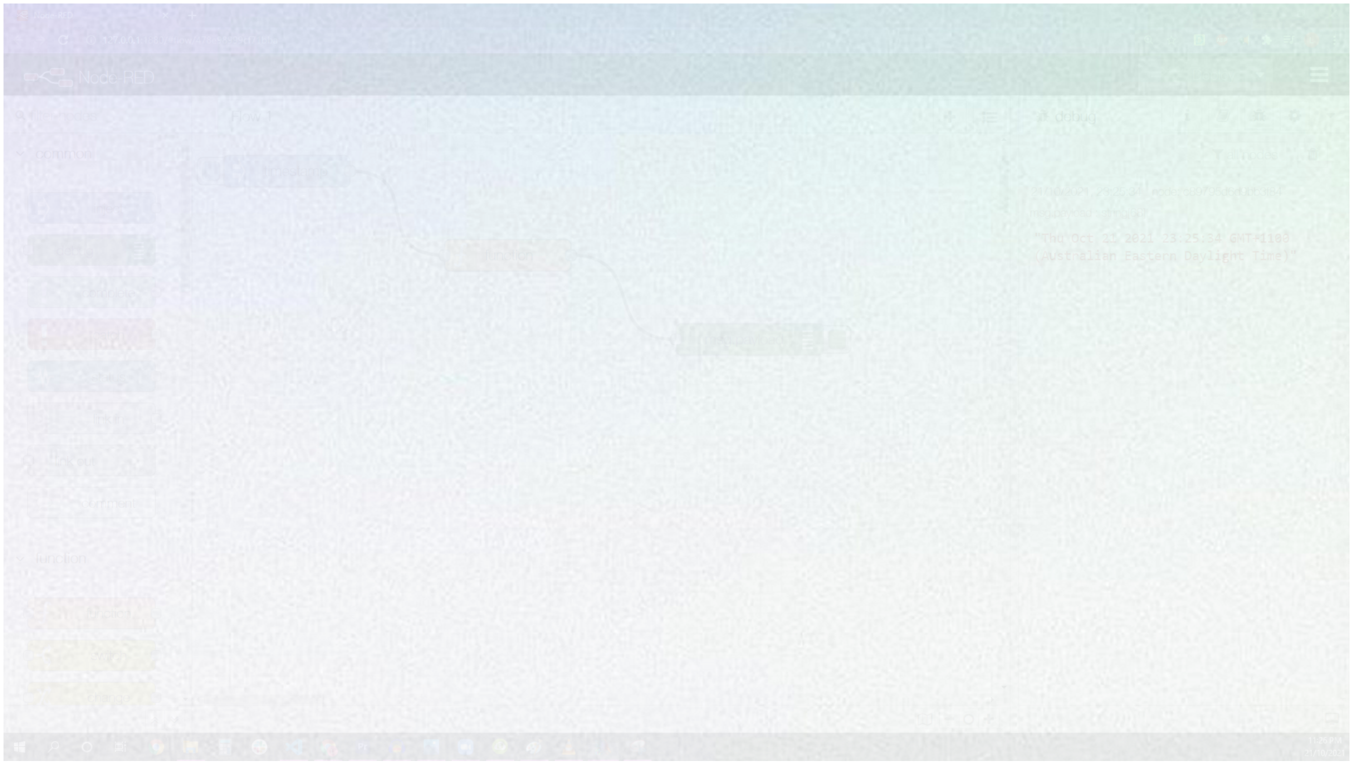


Figure 2: Node-RED with a visual overlay

that the research environment was consistent and replicable. Researchers watched and recorded participants through the tasks. Participants were not given access to Node-read prior to the study. One participant requested to use their own machine and installation of NVDA, as they used it regularly for their vision impairment. They ran both Node-RED and Node-Read directly from their own environment, and were not instructed to utilise any simulators. As with other participants, they were observed using 'screen sharing' and were verbally guided through the tasks.

Questionnaires: After each completed attempt of the introductory Node-RED tutorial 'first flow' with Node-Read and once with Node-RED, they were asked to complete a questionnaire about their experience. Participants rated each version of the application immediately after they had used it.

Observations: We observed that the participants who were unable to complete a section of the tutorial in a given amount of time were offered hints, or additional information that they would not normally be able to access while completing the tutorial in a practical environment. Overall, only one task was not able to be completed without manual researcher intervention. This task was to open the debug pane, for which the button was represented as an unlabeled link, thus entirely incompatible with the screen reader.

5.1 Analysis of Results

More participants stated that they could easily interpret system output, including audio feedback or text labels in Node-Read (50%) than Node-RED (33%). Similarly, participants found Node-Read (58.3%) to be more functionally complete than Node-RED (41.7%). However, participants indicated that Node-RED (50%) remained easier to navigate than Node-Read (41.7%). Each application appeared to be equally pleasant to use overall, however participants felt more strongly about Node-Read, and strongly agreed or disagreed in higher numbers, as shown in Figure 3. Fewer participants indicated that they could repeat the tutorial without guidance in future attempts when using Node-Read (41%) comparing to Node-RED (58.3%). Participants found that the deployment process was more straightforward using Node-RED (75%) than Node-Read (66.7%), as shown in Figure 4. The addition of keyboard shortcuts in Node-Read resulted in more information to memorise thus lowering participants confidence that they could repeat the process without assistance. However, participants consistently stated that Node-Read keyboard shortcuts were helpful to the process (91.7%), as shown in Figure 5.

5.2 Discussions

The results appear to in some way indicate that the Node-Read extension provided more visually accessible approaches to the tasks within the first flow tutorial. However, there is no clear indication that the participants had a more positive experience with Node-Read over Node-RED, and in some cases had a worse experience. Future research is required to understand why this may have occurred.

Unchanged Tasks: While wiring nodes together was unchanged between the two versions, four of twelve participants reported that

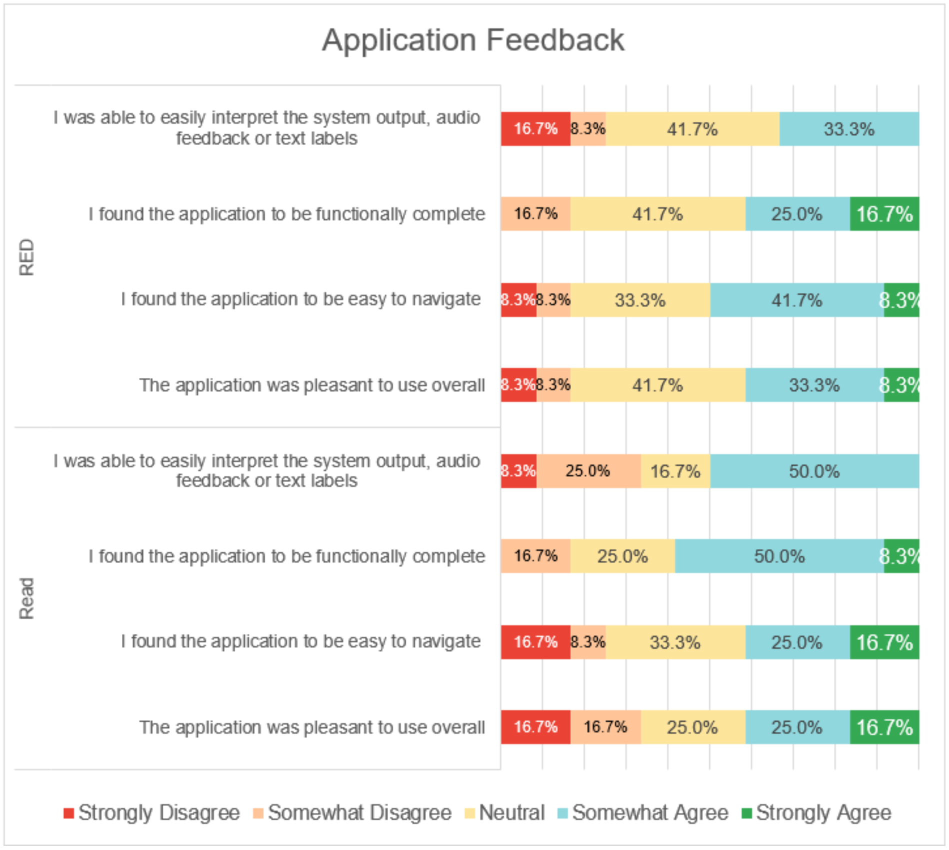


Figure 3: Frequency of feedback scores regarding the overall app experience, separated by app version

the task was more difficult in the Node-Read version, while only two reported that the task was less difficult. The remaining six participants reported no change to the difficulty of the task. We noted that this may be a result of participants being unaware of their previous questionnaire answer for this question, and that different collection methods may produce different results in this area.

Trends: Participants who completed Node-RED first typically found the shortcuts easier to remember. We observed the incremental learning, first of the task, followed by the addition of the keyboard shortcut on the following run-through, often aided participants understanding of the process. Asking participants if they felt that they could repeat the process without guidance revealed that keyboard shortcuts created more actions to memorise, making repeating the process more difficult. The presence of a visual button provides users with a set of actions to take, however there is no audio equivalent system which provides a quick overview of potential actions for keyboard shortcuts. This leads to a higher cognitive load on users when learning these shortcuts, and reduced discoverability of potentially helpful actions. As the task which required the participant to open the debug pane was entirely inaccessible when using a screen reader, the tutorial, as a whole, was therefore very difficult to complete without assistance. This highlights the importance of ensuring that each section within the system is accessible, as many common tasks require interaction with multiple components. If a single component is inaccessible in this manner, we observed that participants quickly became disengaged. This resulted in an increased likelihood that they would review the software negatively.

6 THREATS TO THE VALIDITY

The user study largely drew from a pool of participants who had limited experience using screen readers. Restrictions on travel due to COVID 19 further limited the methods which could be used to gather data from participants.

External Threats: Due to the existing under-representation of visually impaired people within development communities, few participants with visual impairments could be found to participate in the study. Having small number of participants in studies needing participants with specific impairments is common [14, 16, 19]. In place of genuine visual impairments, artificially simulated visually impairments were applied to the software for sighted participants. While this may replicate some of the visual impairments faced by the visually impaired community, it is not a representative sample of all visual impairments. Further, sighted participants had less prior exposure to screen readers, making them not aware of commonly known screen reader shortcuts and best practices. Additionally, the limited sample size creates a higher margin of error within study results. As participants had little time to adjust to the simulated

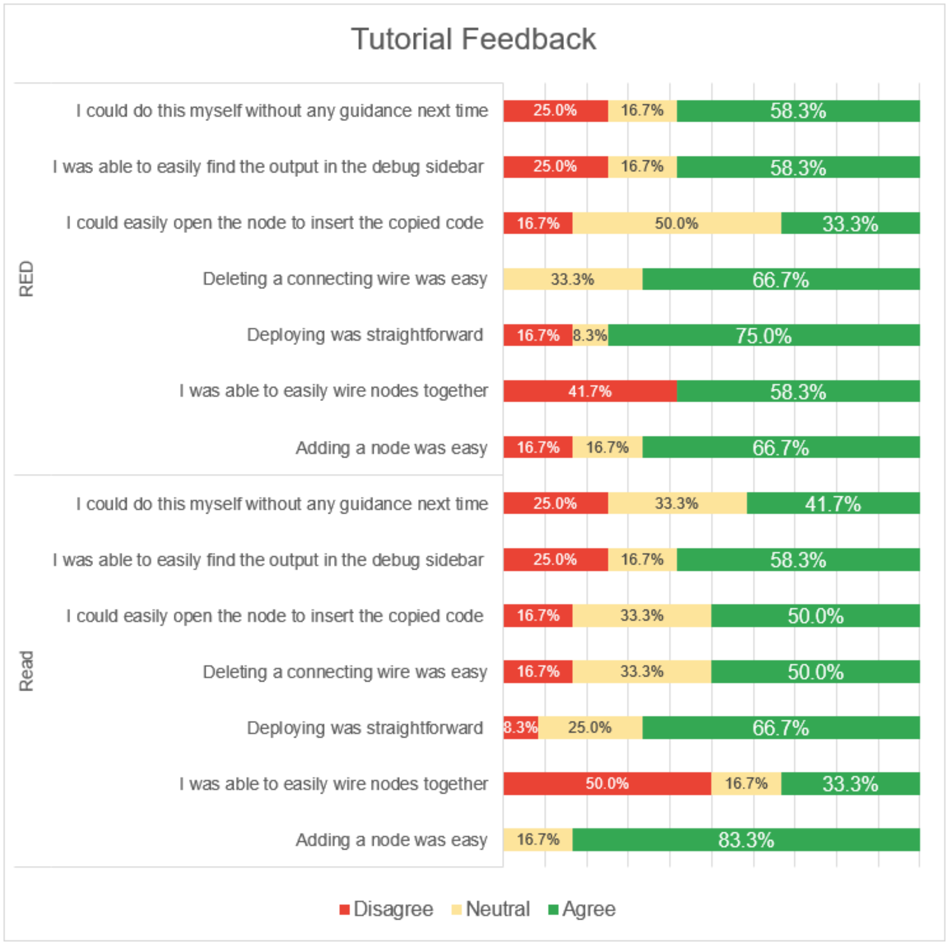


Figure 4: Frequency of feedback scores regarding the specific tutorial steps taken, separated by app version

visual impairments they would use to complete the tasks with, they in turn did not have the time to develop accommodating strategies and learn how to use screen readers. Participants with more prior experience or training in using screen readers may have been able to complete the tutorials without any visual input, as opposed to the reduced visual input used for the study.

Internal Threats: During instances in which the opaque screen filter was not sufficient to obscure participant vision and required re-calibration, participants may have been able to read text on the page prior to the opacity adjustments. Therefore, these participants may have been able to identify nodes in the pallet at a higher rate than counterparts without this initial glimpse. Thus, participants in this category are less representative persons with genuine visual impairments. Some participants reported visual lag as a result of screen sharing through Zoom. This effect potentially impacted the time taken and perceived difficulty of tasks performed throughout the affected period. Replication of these sessions in person could alleviate this source of error. In some instances, participants reported forgetting what they had reported during their first run-through, making comparisons between individual participants software versions and run-through orders less accurate.

7 CONCLUSION AND FUTURE DIRECTIONS

We created Node-Read, a Node-RED extension for developers who require screen readers. We ran a user study with both visually impaired developers, and developers with artificially simulated visual impairments. The user study determined that Node-Read's keyboard shortcuts were helpful, however the system was less pleasant to use, as a result of the additional complexity. Features which reduce this complexity either by increasing action discoverability or by reducing cognitive load provide avenues for future research, and the potential to make these changes will be strictly beneficial for both visually impaired and sighted persons. Providing users with an audio overview, or suggestions of the potential actions at a given time may alleviate some of the reduced action discoverability for screen reader users. Further research is required to validate both impact and potential resolutions in this space.

Many participants reported difficulty wiring nodes together, which cannot be completed with a keyboard shortcut in either software version. Further research into the visual accessibility of Node-RED may benefit from the introduction of a keyboard based method of connecting nodes. Additionally, our study has identified the need for implementation of accessibility labels that provide

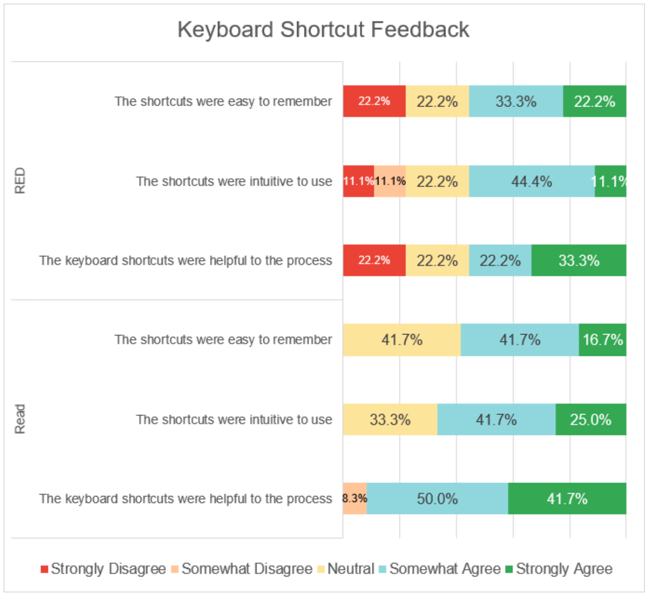


Figure 5: Frequency of feedback scores on the keyboard shortcuts, separated by app version

context, descriptions and state information for buttons and nodes that are currently referred to in generic terms. In addition to the technical changes, it is apparent that the tutorials provided are not sufficiently accessible for the visually impaired. Further research into this space may uncover changes which are not specific to Node-RED, and instead can be generalised to provide a framework into accessible supporting documentation.

ACKNOWLEDGMENTS

Support for this research from ARC Laureate Program FL190100035 is gratefully acknowledged.

REFERENCES

[1] Benjamin Adrian, Sven Hinrichsen, and Alexander Nikolenko. 2020. App Development via Low-Code Programming as Part of Modern Industrial Engineering Education. In *International Conference on Applied Human Factors and Ergonomics*. Springer, 45–51.

[2] 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*. 82–85.

[3] 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*. 91–100.

[4] Ameer Armaly, Paige Rodeghero, and Collin McMillan. 2017. A comparison of program comprehension strategies by blind and sighted programmers. *IEEE Transactions on Software Engineering* 44, 8 (2017), 712–724.

[5] Ameer Armaly, Paige Rodeghero, and Collin McMillan. 2018. AudioHighlight: Code skimming for blind programmers. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 206–216.

[6] Catherine M Baker, Lauren R Milne, and Richard E Ladner. 2015. Structjumper: A tool to help blind programmers navigate and understand the structure of code. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. 3043–3052.

[7] Andrew Begel. 2004. Spoken language support for software development. In *2004 IEEE Symposium on Visual Languages-Human Centric Computing*. IEEE, 271–272.

[8] Sheryl Burgstahler and Terrill Thompson. 2019. Accessible cyberlearning: A community report of the current state and recommendations for the future. *Seattle: University of Washington. Retrieved January* 1 (2019), 2021.

[9] Samer Faraj and Lee Sproull. 2000. Coordinating expertise in software development teams. *Management science* 46, 12 (2000), 1554–1568.

[10] Anne Fisher. 2021. How Companies Are Developing More Apps With Fewer Developers. https://fortune.com/2016/08/30/quickbase-coding-apps-developers/

[11] Filipe Del Nero Grillo, Renata Pontin de Mattos Fortes, and Daniel Lucrédio. 2012. Towards collaboration between sighted and visually impaired developers in the context of Model-Driven Engineering. In *Joint Proceedings of Co-located Events at the 8th European Conference on Modelling Foundations and Applications (ECMFA 2012), Lyngby*. 241–251.

[12] App Inventor. [n.d.]. App Inventor. https://appinventor.mit.edu/

[13] Mukta Kulkarni. 2019. Digital accessibility: Challenges and opportunities. *IIMB Management Review* 31, 1 (2019), 91–98.

[14] Andreas Kunz, Klaus Miesenberger, Max Mühlhäuser, Ali Alavi, Stephan Pölzer, Daniel Pöll, Peter Heumader, and Dirk Schnelle-Walka. 2014. Accessibility of brainstorming sessions for blind people. In *International Conference on Computers for Handicapped Persons*. Springer, 237–244.

[15] Stephen W Liddle. 2011. Model-driven software development. In *Handbook of Conceptual Modeling*. Springer, 17–54.

[16] Leandro Luque, Leônidas de Oliveira Brandão, Elisabeti Kira, Anarosa Alves, and Franco Brandão. 2017. Inclusion in computing and engineering education: Perceptions and learning in diagram-based e-learning classes with blind and sighted learners. In *2017 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–8.

[17] Sean Mealin and Emerson Murphy-Hill. 2012. An exploratory study of blind software developers. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 71–74.

[18] Lauren R Milne and Richard E Ladner. 2018. Blocks4All: overcoming accessibility barriers to blocks programming for children with visual impairments. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–10.

[19] Francisco Oliveira, Heidi Cowan, Bing Fang, and Francis Quek. 2010. Enabling multimodal discourse for the blind. In *International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction*. 1–8.

[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*. 1–11.

[21] Christopher Power, André Freire, Helen Petrie, and David Swallow. 2012. Guidelines are only half of the story: accessibility problems encountered by blind users on the web. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 433–442.

[22] Matthew Rabin and Joel L Schrag. 1999. First impressions matter: A model of confirmatory bias. *The quarterly journal of economics* 114, 1 (1999), 37–82.

[23] Node-Read GitHub Repository. 2021. Node-Read GitHub Repository. Removed-for-blind-review

[24] Node-Red GitHub Repository. [n.d.]. Node-Red GitHub Repository. https://github.com/node-red/node-red

[25] Christoph Rieger and Herbert Kuchen. 2019. Towards Pluri-Platform Development: Evaluating a Graphical Model-Driven Approach to App Development Across Device Classes. In *Towards Integrated Web, Mobile, and IoT Technology*. Springer, 36–66.

[26] Rintagi/Low-Code-Development-Platform. [n.d.]. Rintagi/Low-Code-Development-Platform. https://github.com/Rintagi/Low-Code-Development-Platform

[27] Jaime Sánchez and Fernando Aguayo. 2005. Blind learners programming through audio. In *CHI'05 extended abstracts on Human factors in computing systems*. 1769–1772.

[28] Andreas Stefik, Roger Alexander, Robert Patterson, and Jonathan Brown. 2007. WAD: A feasibility study using the wicked audio debugger. In *15th IEEE International Conference on Program Comprehension (ICPC'07)*. IEEE, 69–80.

[29] Andreas M Stefik, Christopher Hundhausen, and Derrick Smith. 2011. On the design of an educational infrastructure for the blind and visually impaired in computer science. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. 571–576.

[30] Web Accessibility Initiative (WAI). [n.d.]. WCAG 2.1 at a Glance. https://www.w3.org/WAI/standards-guidelines/wcag/glance/

[31] Annalyn Welp, R Brian Woodbury, Margaret A McCoy, Steven M Teutsch, Engineering National Academies of Sciences, Medicine, et al. 2016. The impact of vision loss. In *Making eye health a population health imperative: vision for tomorrow*. National Academies Press (US).

[32] First Flow Tutorial: Creating your first flow. [n.d.]. First Flow Tutorial: Creating your first flow. https://nodered.org/docs/tutorials/first-flow