Interviews and Observation of Blind Software Developers at Work to Understand Code Navigation Challenges

Khaled Albusays
Computing and Information Sciences Ph.D. Program
Rochester Institute of Technology
Rochester, NY 14623
khaled@mail.rit.edu

Stephanie Ludi
Department of Computer Science & Engineering
University of North Texas
Denton, TX 76207
stephanie.ludi@unt.edu

Matt Huenerfauth
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14623
matt.huenerfauth@rit.edu

ABSTRACT

Integrated Development Environments (IDEs) play an important role in the workflow of many software developers, e.g. providing syntactic highlighting or other navigation aids to support the creation of lengthy codebases. Unfortunately, such complex visual information is difficult to convey with current screen-reader technologies, thereby creating barriers for programmers who are blind, who are nevertheless using IDEs. To better understand their usage strategies and challenges, we conducted an exploratory study to investigate the issue of code navigation by developers who are blind. We observed 28 blind programmers using their preferred coding tool while they performed various programming activities, in particular while they navigated through complex codebases. Participants encountered many navigation difficulties when using their preferred coding software with assistive technologies (e.g., screen readers). During interviews, participants reported dissatisfaction with the accessibility of most IDEs due to the heavy use of visual abstractions. To compensate, participants used multiple input methods and workarounds to navigate through code comfortably and reduce complexity, but these approaches often reduced their speed and introduced mistakes, thereby reducing their efficiency as programmers. Our findings suggest an opportunity for researchers and the software industry to improve the accessibility and usability of code navigation for blind developers in IDEs.

CCS Concepts

•Human-centered computing→ User studies; •Social and professional topics→ Assistive technologies; •Software and its engineering→ Requirements analysis; •Human-centered computing→ Empirical studies in accessibility

Keywords

Accessibility; Blind Programmers; Programming Challenges; Code Navigation Difficulties; User Studies.

1. INTRODUCTION

Integrated Development Environments (IDEs) is software which integrates a text editor, file management, compiler, and other tools to promote an efficient workflow for modern computer programmers. IDEs play an important role in the modern software development process, especially when creating lengthy codebases. The text editors in these systems often include visual aids that use indentation to indicate scope level, different colors for syntax highlighting, and various other features to help programmers understand their code structure and navigate through it more easily.

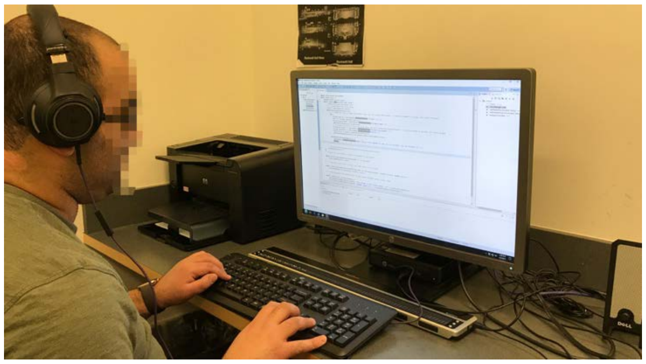

Figure 1. A participant using JAWS with an 80-cell Brilliant Braille Display, while programming in Java using the Eclipse Integrated Development Environment (IDE).

Prior research on blind programmers has found that the information conveyed through visual metaphors in IDEs are often not conveyed by screen readers [2], which creates challenges for blind programmers [9, 18], putting them at a disadvantage when compared to their sighted peers [7, 11]. IDEs provide a particular benefit when programmers are working on large pieces of software, to help (sighted) programmers understand the structure and relationships between lines of code. Blind programmers who are using screen readers or braille displays (which convey information linearly, one line at a time) have difficulty understanding structural relationships quickly [11, 19, 23]. Screen reader users also have access to fewer advanced IDE features for quickly moving through a large codebase, often forcing them to navigate code line-by-line or jump to different locations using "find/search" features [2].

Section 2 discusses how most prior studies on blind programmers have based their findings on case study observations or feedback from a relatively small number of users, e.g. 8 or 12 participants [11, 19]. To build a foundation for future research addressing users' challenges, there is a need for larger empirical studies. Our recent survey of 69 blind programmers explored a broad set of issues [1], and participants listed navigating through code and understanding its structure as key concerns. However, we had not conducted follow-up interviews to analyze these issues more deeply. To address this, we conducted this current observation and interview-based study to specifically investigate how 28 blind programmers navigate through code, using their own preferred development tools while performing common programming activities.

To preview for the reader, our study had three key findings:

1. Programming software (e.g., IDEs) did not meet participants' needs for code navigation; they regularly struggled when performing typical programming activities with these tools. Nevertheless, participants still preferred to use IDEs, even though they encountered these navigation difficulties.
2. Assistive technologies and specific accessibility features of some IDEs did not provide adequate support to enable users to navigate through code comfortably. Although some users were able to customize their assistive technology to better convey the information displayed by the IDE and trigger specific commands, the inefficiency of code navigation made participants feel a loss of control, and they often reported disorientation in the code.
3. Participants felt uncomfortable disclosing their programming needs (e.g., navigation difficulties) and their disability status to colleagues or researchers, which may prevent them from understanding the need to improve the accessibility of IDEs.

In summary, our findings provide greater empirical evidence about the need to improve code navigation features in IDEs for blind programmers, and it identifies opportunities for researchers and industry for addressing these needs. This study contributes to the literature by providing more detailed, firsthand feedback from a relatively large number of participants.

This paper is structured as follows: Section 2 surveys prior work in various tools and techniques to help blind individuals overcome programming barriers. Section 3 outlines our specific research questions to investigate how blind programmers navigate through a lengthy codebase, using their own preferred development tools and while performing common programming activities. Section 4 provides an overview of the methodology used in this paper to investigate the outlined research questions in Section 3. Section 5 explains our interview and observation results, and Section 6 summarizes our conclusions and future research directions.

2. LITERATURE REVIEW

Globally, the number of students entering the Computer Science discipline has increased over the past 10 years [7], however, people with disabilities remain underrepresented in computing [12]. Students who are blind must overcome significant educational and technological barriers, including the heavy use of images and visual abstractions in classrooms; prior researchers have examined how the traditional curriculum in Computer Science has not been designed with assistive technologies in mind [18, 23]. While there has been significant prior research on investigating particular design interventions to benefit blind programmers, e.g. audio cues (Section 2.1) or navigation aids (Section 2.2), there have been relatively few studies that have explored the challenges faced by blind programmers more broadly (Section 2.3).

2.1 Design Interventions: Audio Cues

Significant prior research has examined how to create audio-based accessibility tools for computing students with visual disabilities or other professional software developers [16, 21]. For instance, Sanchez and Aguayo 2005 developed a custom programming tool called Auditory Programming Language (APL) aimed to help blind students write software code comfortably [17]. They demonstrated that audio could convey important information to non-visual users. However, their tool provided a limited set of commands (e.g., input, output, cycle, condition, and variable) making it difficult to scale.

Similarly, Stefik et al. [23] created a tool called Sodbeans based on NetBeans IDE for Java programming, to help convey certain information to students who are blind. The tool used audible cues [6] so that blind students can learn programming concepts. These cues were designed to be browsed in a hierarchical tree manner, to support navigation. In addition, blind students have a rich set of programming environments and tools that they can use beyond the use of Java (e.g., Java, PHP, Ruby). The tool was evaluated based on the students' ability to master the programming concepts.

Various researchers, e.g. [20], have examined the potential of auditory cues to benefit programmers, including the potential benefits of non-speech audio for blind programmers: For instance, Vickers and Alty [25] found that such audio helped programmers locate bugs in their code. Specifically, musical cues proved useful for conveying information to programmers during a debugging process. Boardman et al. [4] created a tool called LISTEN to investigate the use of sounds when analyzing various program behaviors; their goal was to instrument computer programs so that different audible sounds were mapped to different behaviors during the program execution. Stefik et al. [22] investigated the use of audio cues to convey lexical scoping relationships in software code; different cues were played when a change in scope was detected.

Some researchers have examined the limits of understandability of audio cues: In [13], researchers conducted an experiment to investigate the usefulness of auditory cues for navigating menus on mobile phones. These authors also compared two techniques, *Earcons* (a hierarchical progression of variable tones) and *Spearcons* (spoken directions, compressed and sped up) as navigation methods [26]; they found that *Spearcons* helped blind users navigate through cell phones quickly and comfortably. They also found that short cues were more effective than more complicated cues, for conveying certain types of information.

In summary, audible cues have been found to be helpful for a variety of programming activities. Further investigation might help indicate whether audio cues can benefit non-visual users (programmers) to locate errors and debug code faster, mainly to reduce the amount of time blind programmers take to navigate through lengthy codebases.

2.2 Design Interventions: Code Navigation

Several prior researchers, e.g. [8], have proposed interventions to help enhance code navigation. By "code navigation," we refer to the ability of blind programmers to understand lengthy codebases better and how each code statement is nested within the code, which results in enabling blind programmers to navigate code quicker.

Smith et al. [19] developed an Eclipse plug-in to help non-sighted users understand code structure, to speed navigation through a codebase. The authors used keyboard inputs and speech/sound outputs of the hierarchical structure of the codebase to convey certain information to non-visual users. In their work, authors performed a usability test using hyperbolic browser method that employs a fisheye technique [10]. The fisheye technique refers to zooming-in on a single node in a hierarchy tree structure, with the details of the ancestors and descendants presented in reduced detail. Such an approach helped researchers identify strategies that sighted developers tend to use while moving through familiar and unfamiliar trees. Based on this, the authors defined a set of user requirements for an accessible tree navigation system.

Similarly, Baker et al. [2] created an Eclipse plug-in called StructJumper that aimed to help screen reader users navigate through a large amount of codebase quickly. The tool was designed to create a hierarchical tree representation based on the codebase, which presents hierarchical tree-based information about the nesting structure of a Java class. In their tool, blind programmers used a *TreeView* feature to get an overview of the code structure. In addition, they could use a *Text Editor* feature to get an idea of where they are within the nested structure of the code. Thus, blind programmers could look up contextual information about their code without having to lose their position. For example, with the use of shortcut keys, blind developers could press a defined key to find which statement of the code he or she is working on. Such a technique allows a blind developer to quickly jump to the node corresponding to the current location. This approach was similar to that used by other researchers to recognize code in order to present a tree-like structure in a hierarchical tree representation [19].

Other researchers have examined technology interventions to improve code understanding for sighted developers: For example, researchers in [5] created a system where code is presented in "bubbles", which are editable views of, e.g., specific methods or collections of variables; each bubble is in a different color. Of course, the heavy use of visual abstractions is not suitable for blind programmers; further study would be needed to determine whether this bubble metaphor could benefit non-visual users.

2.3 Programming Challenges

While a variety of studies have been published focused on the design and evaluation of specific technology interventions to benefit blind programmers, there have been relatively fewer empirical studies to explore and identify programming challenges. For example, Mealin and Murphy-Hill (discussed below) interviewed 8 participants [11], and Smith et al. [19] conducted an experiment with 12 participants to evaluate a code navigation plug-in. We discuss two prior studies, most closely related to our work:

Mealin and Murphy-Hill conducted an interview study with eight experienced blind developers to highlight their programming difficulties [11], and they identified a number of challenges: First, they noticed that developers were not using the tools available within the IDEs. It was unclear from their study whether users were unaware of the tools offered within these IDEs, found the tools to be too complex, or if the tools were not easily accessible. Second, they found that many blind developers were using a temporary text buffer to store programming notes and to work in it. During the interviews, participants also mentioned challenges with debugging, inaccessible UML diagrams, code navigation, complexity of IDEs, and working in teams with sighted programmers. The authors discussed how blind developers use workarounds or other strategies to overcome the above-mentioned challenges.

We recently conducted an online survey of 69 blind programmers to identify challenges [1]. Similar to the findings of [11] regarding the use of temporary text buffers, we found that blind developers felt overwhelmed when using existing IDEs (e.g., Eclipse, NetBeans, etc.), and therefore they preferred to use simpler editors to write software code comfortably (e.g., Notepad, Notepad++, etc.). In addition, participants indicated that navigating through code and understanding its structure were key concerns. However, the focus of our survey had been on programming challenges broadly, and the survey methodology did not permit us to conduct follow-up discussions with our participants to probe for additional details about code navigation or conduct an in-depth analysis.

3. RESEARCH QUESTIONS

To better understand blind programmers' challenges and build a foundation for future research addressing these problems, we have therefore conducted an observation and interview-based study to specifically investigate how blind programmers navigate through code, using their own preferred development tools and while performing common programming activities. This new study also allowed us to more specifically investigate this issue of code navigation that had been raised in our prior survey [1]. Specifically, we wanted to investigate the following questions:

- **RQ1:** What difficulties do blind developers encounter when navigating through a codebase?
- **RQ2:** What tools do they use in their development work?
- **RQ3:** What workarounds or strategies do they use to overcome any code navigation barriers?

4. METHODOLOGY

As methodological inspiration, we have drawn upon the recent work of Szpiro et al., who conducted a study using contextual inquiry and qualitative data analysis to understand the challenges faced by people with low vision when accessing computing devices [24]. Their goal was to uncover challenges and identify opportunities for researchers and industry to improve low vision accessibility tools, and we have similar aims in regard to code navigation for blind programmers. In this work, we used observations and semi-structured interviews with blind programmers to identify code navigation difficulties, the tools they used, and any workarounds they employ. This methodology will help us gain deeper insight, relative to our prior survey-based study in [1]. In addition to identifying future research opportunities, another goal was to involve blind programmers in the research and to gather firsthand comments and suggestions from these users.

4.1 Interview Design

Prior to the main study, we conducted pilot tests (mock interviews with five sighted programmers) to ensure that our semi-structured question plan, interview technique, and procedure were well-formed. As a result of these pilot tests, the interview questions and procedure were modified, e.g. the wording of some of the questions were changed to use terminology more familiar to this user group.

The planned questions included five multiple-choice and 16 open-ended questions, which were grouped into several topics:

- **Demographics:** user characteristics such as age, gender, country, visual acuity, and level of expertise in software development.
- **Languages and Tools:** to identify a list of programming languages and programming tools (e.g., Eclipse, etc.).
- **Assistive Technologies:** to identify participants' preferences of assistive technologies such as screen readers or braille displays.
- **Development Style:** to capture and observe blind programmers' strategies when developing software, mainly to navigate code.
- **Navigation Difficulties:** to uncover navigation difficulties and how it impacted blind developers' performance.
- **Navigation Tools:** to investigate existing code navigation tools and how it helped overcome navigation difficulties.

- **Working in Teams:** to understand how blind programmers work in teams, mainly with sighted programmers.

4.2 Participants

We recruited participants using a private mailing list from previous studies (individuals had agreed to join this mailing list and had previously indicated an interest in participating in studies) and by posting advertisements on private groups (Google, LinkedIn, and AppleVis) for people who are blind. A total of 36 people responded. We conducted an initial screening interview over *Skype* and *Google Hangouts* to first determine the eligibility of the participants. To participate, individuals had to be an experienced developer (5+ years in programming), 18 years or older, self-identified as fully blind, actively engaged in programming either as a job or a hobby, and a user of assistive technologies (e.g., screen reader or braille display). Eight respondents were excluded from the study due to the use of magnifiers and corrective lenses. (The focus of this study was on users of screen readers or braille displays.)

Afterward, we conducted the interview sessions with the remaining 28 blind programmers. Participants (all male) varied in age from 22 to 52 (mean =29.68, SD=6.59). Our sample showed variation in programming experience (lowest = 5 years, highest = 24 years) and employment status (e.g., retired, employed, unemployed, freelancer). Few unemployed participants are searching for job opportunities. All participants use screen readers, and 8 participants used braille displays (see **Table 2**). Participants were from five different countries: United States (n = 22), United Kingdom (n = 3), Australia (n = 1), India (n = 1), and the Netherlands (n = 1).

4.3 Procedure

The interview took place online via *Skype* and *Google Hangouts* per the participants' preference. Prior to the interview, participants were provided with informed consent documents and the interview questions (so they could familiarize themselves with the interview topic in advance). Each interview lasted approximately one hour and occurred during January to December 2016. Enrollment had not been fixed; rather, recruitment was discontinued (at 28) after the researcher observed that no new issues were raised during the sessions conducted with participants 27 and 28. All participants were compensated for their time with $50 Amazon gift cards.

The session began with the brief semi-structured interview (questions in section 4.1). This was followed by an observation period when we asked participants to engage in common programming activities. The participant transmitted their voice and their computer's audio output, and in addition, they transmitted the video image of what was displayed on their computer screen. The interviews and observations were recorded with the participants' prior approval, using *Screencast-omatic* software.

Since our goal was to understand code navigation difficulties and observe how participants deal with these navigation problems, we identified in advance a set of programming activities, which we requested the participant to perform during the observation:

- Conducting a programming walk through using any language or tool: We asked the participant to open some code that they had been editing recently as part of their professional work and to explain the code, giving a demonstration of its structure.
- Demonstrating for the researcher some code navigation difficulties they encounter frequently.
- Navigation walk through of some other programmers' codebase with which the participant had no prior knowledge.
- Demonstrating any strategies or workarounds that the participant uses to overcome navigation difficulties.
- Demonstrating any solutions or tools and how they helped.

We observed participants perform the above-mentioned programming tasks, and we occasionally interrupted them with questions. For each question, we encouraged participants to speak freely and openly (explaining that their feedback was very valuable and this research might benefit other programmers in the future) so that we can elicit more detailed answers. We did not insist that participants use specific programming languages or development tools, mainly because participants owned various platforms and had their own preferences. In a few cases, some participants did not wish to perform one of the asks or were unable to do so, and we did not insist in those cases. Our priority during the session was to elicit comments and impressions from the participants about code navigation difficulties that they encountered when performing these tasks, to capture information about: what assistive technologies that they use and why, how they used them, how they completed these activities, and how they felt when performing it.

4.4 Data Analysis

During the session, we captured the following data:

- Responses to closed-ended questions were recorded.
- Additional written notes were taken for open-ended responses, with particular focus on capturing direct quotations.
- Timestamps were noted when an important issue was raised, to facilitate the researcher reviewing key portions of recordings.
- All notes were stored and duplicated for further analysis.

We followed a qualitative methodology for our data analysis. The data was managed and annotated using NVivo qualitative data analysis software. Following an open-coding method [15], we analyzed open-ended questions based on their content using a set of codes that we developed to represent recurring ideas or problems raised by participants. We assigned codes to segments of text transcription or experimenter notes in our dataset.

Two researchers performed coding independently, reading and organizing the participants' transcripts. Afterwards, they met periodically to discuss code categories (e.g., navigation challenges, assistive technologies, programming tools, workarounds, and user needs). In rare cases when coders disagreed (inter-rater reliability = 67%), they held a meeting to reach an agreement and form a consensus coding. We generated a set of themes based on the number of times each issue was raised. For example, high occurrences indicate higher demand or importance. Themes were developed using affinity diagramming [3], which is a useful technique for organizing and analyzing large-scale qualitative data.

5. RESULTS

In this section, we describe key findings, illustrated with examples of our participants' behavior or comments from the interview sessions. Quotations are labeled with code numbers preceded by the letter P that represent individual participants (e.g., P1, P2, etc.). This section is organized based on the major themes that arose during our data analysis: code navigation challenges (Section 5.1), tools (e.g., assistive technologies, programming languages and tools) (Section 5.2), and strategies to overcome navigation difficulties (Section 5.3).

5.1 RQ1: Code Navigation Challenges

In software development, programmers regularly use their sight to obtain information about their software codebase, which allows them to formulate an understanding of their code structure and navigate throughout the code. Blind programmers rely on other senses (e.g., hearing and touch) to acquire contextual and structural information about their software codebase. We observed our participants encountering several code navigation difficulties when performing various programming activities, and participants discussed this issue in their interview responses. We summarize a taxonomy of sub-types of navigation difficulties in **Table 1**; next to each description, we provide the number of participants who mentioned each issue. Due to length constraints, the remainder of this section will summarize some key points, along with illustrative examples and quotations from participants.

Navigation Difficulties	# Participants
Debugging: difficulty navigating through the code in the process of understanding a wrong output.	24
Line by Line: difficulty navigating through code to locate specific information without having to go through the entire codebase linearly, line-by-line.	23
Indentation: unable to distinguish the level of whitespace using a screen reader in indentation-based languages, e.g. Python.	22
Nesting: difficulty navigating through nested methods, loops, functions, or classes.	20
Back Track: difficulty returning quickly to a specific line (in a lengthy codebase) when reviewing other code statements in various files.	18
Errors: difficulty quickly locating code errors while navigating through lengthy codebases.	14
Scope: difficulty understanding the scope level, e.g. while navigating deeply nested methods or loops.	14
Characters: difficulty perceiving certain characters, operators, and parentheses, e.g. missing some characters while coding.	10
Autocomplete: difficulty accessing the autocomplete feature due to incompatibility with the screen reader.	9
Relationship: unable to distinguish the relationship between code entities within a codebase, e.g. the relationship between a class and its subclasses.	9
Line Numbers: difficulty accessing line numbers in the code editor as they were not designed to be readable by a screen reader, e.g. using PyCharm with VoiceOver.	7
Elements: unable to quickly locate a specific element within a given array, class, function or loop, e.g. locating values or variables.	5

Table 1. List of navigation difficulties and number of participants who mentioned each during interviews; the difficulties are sorted based on this number.

Debugging: When a failure occurs in software, programmers must perform three main activities to correct the failure. First, they need to perform fault localization to identify the code statement responsible for the software failure. Second, they need to complete a fault understanding activity that involves understanding the origin of the software failure. Third, they must perform a fault correction activity, to determine the best way to remove the cause of the software failure. All three of these activities are commonly referred to as "debugging," which is an essential skill in software development [14]. Our participants indicated that they understood the importance of debugging and how it helps to correct unwanted software behaviors. However, participants indicated that they tend to rely on simple debugging techniques, mostly because of the accessibility issues in current debugging tools (e.g., *FindBugs*, *Firebug*, etc.). For example, P4 examined several available debugging tools to find one that is compatible with their screen reader. He found that most debugging tools were not accessible as they were designed with vision in mind. Therefore, P4 and many other participants (n = 19) decided to rely on simple debugging techniques such as inserting print commands in the code or tracing:

"I rely on printf *to fix code defects. I also tried to test different tools like* FindBugs *or* Firebug*, but they were not fully accessible to [my] screen reader." (P4)*

P26, on the other hand, discussed the difficulty of navigating through a lengthy codebase to find logic errors. He explained that debugging techniques, such as *printf*, may take longer as there is no clear indication where to find the problem that caused the software to behave incorrectly. While most participants relied on simple debugging techniques, some (n = 9) used advanced debugging tools:

"I was trained to use advanced debugging tools by my sighted colleagues even with the accessibility issues. I think the training helped me use them better." (P8)

Although participants vary in their own debugging experiences, most participants mentioned that debugging is a significant barrier to blind programmers, mostly because it is difficult to interpret software control flow while debugging.

Line by Line: We have discussed previously how vision helps software developers get an overview of the entire codebase. To get an overview of code, most of our participants (n = 18) indicated that they tend to go through a codebase line by line, mainly because screen readers encourage users to move through text in a linear fashion. P5, for example, explained a difficulty that they encountered when working with complex codebase:

"How to accomplish things in my complex code [is] frustrating. I need more time to understand each line and more time to remember what each code block is doing." (P5)

While several participants discussed the difficulty of navigating linearly with a screen reader, some (n = 8) used other techniques, e.g. searching through the codebase using keywords, to avoid scrolling through the entire codebase line by line. While P13 and others enjoy using keywords, another participant (P20) indicated that keyword searching is time-consuming and often frustrating, because the same keyword might appear in several locations within the same codebase. P6 and a few others, on the other hand, agreed that keywords are very popular among blind developers to find a specific code statement. But considering that some cases where the same keyword is used twice or even more, blind programmers often need to review a few code statements before and after the keyword location to ensure that they have found the right line:

"Keywords [are] useful when you deal with the small code, but not a large one, especially when you try to find a variable that [has] been used several times in different locations. Which code block I am reviewing is hard to distinguish with keywords." (P6)

Indentation: Indentation-based languages (e.g., Python, Occam, etc.) use whitespace indentation to delimit code blocks, instead of using keywords or curly braces. In these languages, an increase in indentation may indicate a new, deeper code block, and a decrease in indentation indicates the end of the code block. Python was the most commonly used programming language among our participants (n = 18), mainly because of their job requirements. To navigate through an indentation-based language, most participants indicated that they tend to go through it block by block instead of line by line, mainly to avoid the verbalization of whitespaces (indents) using a screen reader. By "block by block," we refer to instances when blind programmers wish to skip-over code blocks (e.g., in a loop, a function definition) to avoid reading one code statement at a time while browsing the entire codebase. For example, P10 explained that a screen reader will verbalize an indentation as a sequence of individual "space" characters, rather than a single indent of a particular length. When a screen reader user navigates through indentation based languages, the blind programmer will hear his or her screen reader verbalizing whitespaces as a single space (e.g., "space, space, space") rather than a count ("three spaces").

P21 explained how to overcome the whitespace problem using a screen reader. The solution involves writing a custom script (a modification of the typical functionality of a screen reader for a particular application) that forces the screen reader to calculate whitespaces and verbalize it as a complete list of whitespaces:

"I found it useful to write script that forces my screen reader to calculate the whitespaces and then present it [to me]. I designed the script to say, for example, 'four spaces' instead of saying 'space' four different times." (P21)

P3 explained a similar approach:

"Instead of listening to my own screen reader telling me all the spaces separately. I wrote [a] script to give me the level of indentation in my code." (P3)

Although calculating whitespaces and verbalizing it helped several participants (n = 5), others (n = 4) found a braille display much more helpful in determining the level of whitespaces. For instance, P16 reported that a braille display provides valuable assistance in determining the level of whitespaces, through touch. Section 5.2 discusses how users mitigate this whitespace issue by using a braille display in conjunction with their screen reader:

"I use a screen reader and [also] braille display with Python, it helps [me] feel the indentation in my code." (P16)

5.2 RQ2: Tools in Software Development

In this section, we discuss the participants' behavior or experiences towards assistive technologies, programming languages, as well as development tools. We also describe each method and technique used by participants to perform various development activities. We presented each category with the actual number of users based on the participants' use of each language or tool.

5.2.1 Assistive Technologies

Assistive technologies refer to any specialty hardware or software add-ons that were designed to increase the functional capabilities of people with disabilities. These assistive tools, whether developed by the industry or privately customized by the end users, provide freedom and independence to people with special needs to accomplish tasks that are difficult without getting help from those who are sighted. In this paper, participants used two different forms of assistive technologies; screen readers and braille displays. A screen reader enables blind users to access the computer display by linearizing the presentation of information from the graphical user interface and verbalizing this information using a speech synthesizer (or transmitting this information to a braille display).

Participants described a variety of experiences performing common programming activities using their screen reader. For example, P2 prefers to use the Non-Visual Desktop Access (NVDA) screen reader when working with a Python codebase:

"I use NVDA because its free, made by a blind user, and helps me convert text into [a] Braille Display." (P2)

P10 uses NVDA for programming activities, mainly because it allows for personal customization. He uses *PyCharm* to write Python applications, despite challenges in using this tool with his screen reader. P10 indicated that *PyCharm* is very complex platform, and it poses many programming problems:

"I like to use PyCharm to write python application, I modified NVDA script to ignore unwanted features and to help [me] reduce its complexity." (P10)

Although many participants (n = 12) decided to use NVDA for personal reasons or financial constraints, others (n = 16) preferred to use a different type of screen reader (see **Table 2**). For instance, P13 uses JAWS with development software, mainly because it allows users to load specific scripts (customized modifications of its behavior) for each platform:

"JAWS provide me with great functionality. You can assign specific script to each application, it helps reduce the time I take to navigate through the entire application." (P13)

A refreshable braille display is an electro-mechanical device to translate information from the computer display into braille characters. It uses round-tipped pins in a flat surface that are raised through holes to convey information to blind users. These devices are available in different sizes (different number of characters that can be displayed in a line simultaneously, e.g., 18, 40, 80) based on the user's needs. In this paper, several participants (n = 8) indicated that they use a refreshable braille display with a screen reader to perform various programming activities (see **Figure 1**). For example, P24 preferred to use a braille display when working with Python codebases, mainly to understand the level of indentation as its difficult to understand when using a screen reader alone:

"Braille display is much better than screen reader when it comes to detecting indentation level. [The] screen reader will say 'space', 'space', 'space', etc. Which is too much to handle with complex code." (P24)

Some participants (n = 6) explained that they preferred to use a refreshable braille display to navigate through a codebase because it was quicker than a screen reader. Others (n = 2) tend to use braille displays because it reduced their "hearing load," i.e. the stress they experience from attending too much information conveyed on the audio channel in an interface. For example, P28 discussed how a screen reader creates significant hearing load when performing programming activities at work:

"I read texts and software code using braille display, [it] helps reduce [the] hearing load and makes me aware of the surrounding, especially in work settings." (P28)

Other participants (n = 2) explained that they used a multi-line braille display. (Most braille displays present a single line of characters, but some are capable of presenting multiple rows of characters simultaneously.) Participants indicated that this device helped them to read several lines of code to get a better overview of the code structure, rather than using a screen reader or a single-line braille display, which presents information linearly:

"Navigating code [is] difficult with screen reader, you feel isolated to one line at a time, I use multi-line braille display which helps me read more than one line at a time." (P16)

5.2.2 Development Languages & Tools

Our participants' knowledge and experience in programming languages and development tools varied. Some (n = 15) were proficient in more than one programming language, and others (n = 13) were experienced in a single language only. This variation was mainly due to their specific job requirements or constraints that are presented by the structure of the programming language.

Participants were asked to the list the programming languages and tools that they use to develop software (see **Table 2**). Our results showed that Python was the most used language among all participants. In fact, 18 participants (n = 64%) indicated that they use Python to write software code for several reasons, including: its simplicity, its rising popularity, and the fact that can be used as an interpreted language – thereby providing users with the ability to dynamically inspect and change their programming code. Although Python was the most used language among all participants, other participants (n = 10) preferred to use Java, again, mostly for job requirements. For example, P27 developed several applications that run on computers, smart cards, and cell phones for the company:

"I developed the company clients support application with other colleagues that was written in Java. [We] choose Java because [of] its well-written libraries. [We] use other languages as well, but mostly Java." (P27)

Operating Systems	#
Windows	23
Linux	8
Mac OS	4

Assistive Technology	#
NVDA	12
JAWS	10
ORCA	5
VoiceOver	4
LSR	3
Windows-Eyes	1
Braille Display	8

Programming Languages	#
Python	18
Java	10
C++	10
SQL	7
C	6
Swift	4
Ruby	3
C#	2
Objective-C	2
PHP	2
Perl	1

Programming Editors	#
Notepad++	18
PyCharm	16
Visual Studio	12
NetBeans	8
Notepad	6
Notepadqq	5
Eclipse	4
Xcode	4
CODA	4
Atom	2
IDLE	1
TextMate	1
Padre	1

Table 2. Number of participants in our study using various operating systems, assistive technologies, programming languages, and programming editors.

In regard to development tools, all participants preferred to use simpler editors rather than current IDEs. Participants explained that simpler editors (e.g., *Notepad*, *Notepad++*, *Notepadqq*, etc.) were popular due to their simplicity and flexibility with assistive technologies and programming languages. *Notepad++*, for example, was especially popular among users of the Windows operating system as it available for free. P4 explained that his reason for using *Notepad++* was due to its wide range of plug-ins, that helped facilitate writing software code. While some participants (n = 7) favored plug-in features to install tools that had previously been developed by the blind programming community, others (n = 8) find it useful to write their own plug-ins. For example, P15 worked with several blind programmers to develop a plug-in that allows screen reader users to navigate through autocomplete functionality, mainly to make it more accessible. Autocomplete is a common feature in most IDEs in which the system displays a pop-up menu of predictions of what the programmer is about to type next, based on the first few characters of the word they have typed. But this feature is not fully accessible to screen reader users, mainly because it appears on the screen as a pop-up which the screen reader does not recognize. Although most participants preferred to use simpler editors when performing various programming activities, all participants agreed that IDEs are necessary at times, despite accessibility problems.

5.3 RQ3: Programming Strategies

In the midst of a discussion about navigation difficulties with our participants, it would have been easy for participants to forget to mention positive information, such as navigation workarounds or strategies. For this reason, we specifically asked participants to demonstrate or explain some examples of these. Our participants discussed a myriad of strategies to overcome various programming challenges, mainly code navigation difficulties. Due to length constraints, this section will summarize some key points, along with illustrative examples and quotations from participants.

Simple Editors: As discussed above most participants (n = 26) indicated that they rely on simple editors to write software code; we highlight here how several of our participants reported using simple editors in concert with IDEs – to overcome inaccessible features in existing IDEs. For example, several participants (n = 8) explained how they use simple editors to record code errors, bugs status, and where variables located to enhance navigation. Other participants (n = 7) use them to avoid losing their current spot while reviewing other code statements. For example, P18 demonstrated how to use *Notepad* to navigate through a complex codebase that was written by other programmers:

"The code I am showing is large and long. I work with other programmers to maintain it and mostly to modify it. I use Notepad to record code errors while reviewing other statements for reference." (P18)

Custom Scripts: Our participants expressed mixed feeling about the use of assistive technologies, mostly screen readers. In this work, most participants (n = 19) modified screen reader settings to match their own personal needs. Others (n = 9) wrote custom scripts to overcome many issues including programming difficulties. Participants explained that creating a custom script is not a perfect solution, yet it still provides an alternative method to solve some of the problems they experience when interacting with current IDEs. For example, P11 showed a script that was designed to force the screen reader to locate elements on the *PyCharm* (IDE) which was not fully accessible. P9, on the other hand, reported that his screen reader will not read line numbers on some of the IDEs, mainly because line numbers was not designed to be readable by a screen reader. Therefore, he wrote a custom script to force the screen reader to read line numbers:

"I wrote many custom scripts to help do my job faster. My screen reader will not catch line numbers on some of the IDEs, so I coded [a] script to force my screen reader catch line numbers. (P9)

Shortcut Keys: as researchers, we were interested to know how blind programmers get a high-level overview of the entire codebase for navigation purposes. Screen readers navigate through codebase linearly, forcing the user to read the entire codebase one line at a time. To overcome this problem, several participants (n = 9) indicated that they use shortcut keys as a navigation strategy. For example, P4 relies on shortcut keys to locate specific code statements without scrolling through the entire codebase. Other participants (n = 12) use them to get structural information about their codebase. However, P1 argues that shortcut tools like *find* comment (to search for text strings) can help programmers find content in the codebase using keywords, but often a single keyword is not enough to jump through all the associated content (for e.g. in programming languages like Java and C++, jumping through all the functions in a code using a single search keyword can be ineffective as all related functions might not use those specific keywords).

However, P12 said that the use of shortcut keys was inefficient since it forces users to jump between code blocks, which is difficult for someone who is blind, especially for unfamiliar codebases:

"Depending on the language, the start of the block may not be easy to follow without reading through all lines. In cases like that, shortcut keys may not be [a] helpful strategy at all." (P12)

P19, on the other hand, was annoyed that various IDEs make use of specific shortcut key combinations that are also used by his screen reader, leading to conflicting functions:

"I rely on shortcut keys to navigate through code, but there are overlapped keys between several applications. I had to write a custom script to control overlapped shortcut keys for me." (P19)

Code Comments: In software development, commenting involves placing different readable descriptions inside code blocks to detail the purpose of each block. Most blind developers rely on them to make code maintainable and debugging easier. Commenting is an important technique, especially when a project involves other programmers. In this work, most participants (n = 16) used commenting, not in the traditional manner (to make source code readable or document how a certain function works), but rather to overcome navigation barriers. For example, P3 used commenting to locate software bugs that need to be addressed immediately with other software programmers. Although some participants (n = 6) used commenting to locate code errors or bugs, others (n = 9) use it to highlight code statements that require further review:

"When modifying some of my code function, I use comments to locate them fast, especially while checking other statements so I can get back to them fast." (P22)

Sighted Help: Our participants indicated that seeking help from others, especially from those who are sighted, is avoided by many blind individuals in workplace settings, often due to embarrassment about the amount of time they take to accomplish certain tasks. Additionally, many participants indicated that they wanted to demonstrate that their visual loss had no impact on their ability to fulfill their job requirements. Although most participants (n = 16) tended to avoid seeking help from sighted co-workers, others (n = 10) found it necessary. For example, P25 seeks sighted help to get an overview of the entire codebase when a new implementation takes place. This helps reduce the amount of time a blind programmer takes to get an overview of the entire implementation. P7 agreed that requesting sighted help is understandable since blind programmers are unable to simply glance at codebase due to the linear nature of the screen reader:

"Reviewing another programmer's code with a screen reader takes longer than someone who is not blind, I seek help sometimes to get [a] quick overview of the new implementation." (P7)

P13 shares a similar opinion about the importance of requesting sighted help whenever needed:

"I enjoy working with sighted programmers, you always learn many tips." (P13)

5.4 The Need for Better Navigation

As part of our interview, we also discussed with participants some possible future features that could be added to IDEs to improve their accessibility. In some cases, the participants requested features prior to being prompted. For all participants, we included a section in the interview in which we briefly described several possible future enhancements to IDEs – to gauge the interest our participants had in each option. Overall, 82% of our participants (n = 23) showed interest in using these various features (listed below), while 18% said that they might be willing to try them. The set of possible future enhancements to IDEs discussed during our interviews included the following:

Tree View: Most participants expressed the need to have an alternative feature to navigate through codebase, mostly to avoid going through it line by line. For example, several participants suggested a hierarchical navigation feature in which codebase could be presented as a tree, mainly to hide code complexity. (This is in agreement with prior findings of Baker et al. [2].) In fact, 18 participants (64%) showed interest in using such a feature. Tree view (or tree list) is already available feature in some of the IDEs but is not fully accessible to screen readers:

"Going through code line by line is very difficult with [a] screen reader, especially when you deal with complex software code. As blind programmers, we discuss many ideas about accessibility in programming. In fact, we thought to program [a] tool that presents the software code as tree instead of navigating through line code, which takes forever." (P14)

"I would love to see a tool that shows code in a different way, not line by line." (P27)

Auditory Feedback: Several participants (n = 7) suggested that sounds should become a core integration component when interacting with programming activities, especially for blind programmers. For example, some participants (n = 3) indicated that sounds would help them monitor background processes in development tools while attending other tasks. One advantage is that auditory cues can help blind programmers split their attention between an immediate task and waiting for the result of some background process. Participants also suggested that sounds could be used to help provide additional information regarding syntax errors, invalid statements, and current location in code in order to reduce programming difficulties. (This is in agreement with findings of Vickers and Alty [25].) In fact, 19 participants were interested in using auditory feedback (68%), while 9 (32%) participants said that they would be willing to try it:

"It would be nice to have audio feedback when we make code mistakes. It will help locate errors while navigating through code or maybe highlight any syntax error." (P8)

"The way how programming relies on visual representation is the major impact in almost all difficulties that we face as blind individuals. We need another way of programming, maybe with audio or something else as I can't think of different way that could help us." (P24)

Bookmarks or Tags: Our participants described how they used comments to leave keywords at particular locations in their code, which they could then jump to more easily by using a search feature. Participants also reported that they tend to remove all of these comments before sharing their code with others, especially sighted people, mainly because they feel embarrassed. Several participants expressed the need to have a bookmark feature in which they could tag specific line of code and return to it later for further modification (without making use of comments and searching to accomplish this task). Participants cautioned that the bookmark feature should be designed to jump to a specific code statement, rather than to a specific line number (which may shift when additional code is inserted or deleted). Bookmark or Tag features are already available in Visual Studio and other IDEs, further investigation may indicate whether such tools are fully accessible and beneficial for non-visual users. In fact, 24 participants (86%) showed interest in using bookmark feature, while 4 (14%) participants said that they might try it:

"I always wanted to build [a] tool that tags code for personal use. You could build it in [a] way that any line can be tagged either for private or public comments. You could also use shortcut keys to locate each tag to quickly find them. I guess I did not find the right time to develop it." (P22)

Nesting & Scope Level: Nested code is commonly used in software development where various programming logic structures are combined to one another (e.g., embedded within one another). Deeply nested code can pose challenges for blind users because it is harder to read. When nested code goes beyond three levels of indention, it can be difficult to understand and navigate. To handle nested code, sighted programmers tend to use *code folding* in software editors. This feature allows them to collapse an entire code block (visually hide the full text of the code and replace it with a small visual placeholder instead), which allows programmers to have a better view of the surrounding code statements. Several of our participants also suggested that it would be valuable to have a scope and nesting level indicator feature. This would read aloud the current cursor location when a special shortcut key combination is pressed. We are not aware of any similar study or tool in this regard. In fact, 19 (68%) of our participants were interested in having nesting and scope level indicators, while 9 (32%) participants said that they might try it:

"I find it difficult to know my location when working with nested code block. You can't tell with a screen reader unless you read the entire block. I think a good solution is to have a tool that gives [me] the location and how deep I am within the nested code." (P5)

Class Relationships: In object-oriented programming, a class is used to describe one or more objects, mainly to serve as a template for creating various objects within a program. Each object is created from a single class – this one class could be used many times, mostly to instantiate multiple objects. It can be also used by software developers to isolate specific objects so that their internal variables or methods are not accessible from all parts of the program. This prevents the programmer from changing internal implementation details of some code, which might break other parts of the codebase. Programmers tend to use classes to help create more structured programs that can be easily modified. The inheritance relationships for classes can become complex, especially when there are multiple subclasses that inherit all or some of the characteristics of the main class. To understand class relationships, sighted programmers often rely on diagrams (e.g., how components are interrelated). Diagrams can be difficult to understand by blind programmers. Our participants expressed the need to have some method of conveying class relationship features, e.g. audio cues as they navigate through classes or subclasses in order to provide an overview of classes in a codebase. We are not aware of any similar study or tool in this regard. In fact, 17 (61%) of our participants expressed interest in using a class relationship feature, while 11 (39%) participants said that they might use it:

"It would be interesting to have class relationship tools where you get instant feedback through audio. Maybe [by] pressing shortcut keys to get audio feedback whenever I need to know all the subclasses of a class." (P18)

6. CONCLUSION AND FUTURE WORK

In this paper, we presented our exploratory study aimed at understanding code navigation challenges encountered by blind programmers when using various development tools. We illustrated and discussed our methodology for learning about code navigation difficulties from our participants: blind software developers. Our study offers a new perspective into the use of common development tools (e.g., Eclipse, NetBeans, etc.) alongside assistive technologies by developers who are blind. Most previous studies have based their findings on a small number of participants [2, 11, 19]. Our results arose from observing and interviewing a much larger sample, and our findings highlight various code navigation difficulties based on different programming languages and tools.

Our findings indicated that participants struggled to navigate through codebases using existing development software alongside assistive technologies (e.g., screen reader). Although accessibility tools provided benefits, they failed to give enough support for blind programmers to navigate through codebases quickly and comfortably. Since navigation options in IDEs are restricted to sighted users, blind programmers prefer simpler editors (e.g., *Notepad*, *Notepad*++). Participants explained and demonstrated how diverse programming environments, in combination with assistive technologies, lead to various challenges, often because these IDEs were designed without accessibility in mind.

Most of our participants preferred to use a screen reader (despite its limitations) to write software code. Others found this difficult, and therefore, favored using a braille display instead. However, several of our participants indicated that they could not afford to purchase a braille display. While most IDEs were not fully accessible, blind programmers still rely on them to accomplish their work. Moreover, some blind programmers may seek sighted help for various reasons, mostly to access content that is not accessible with assistive technologies. Although some blind programmers seek sighted help, others prefer writing custom scripts to overcome many programming challenges. For example, several blind programmers wrote custom scripts to enhance navigation in indentation-based languages. Others wrote scripts for each IDEs, mainly to access features difficult to use with a screen reader.

There were some limitations of our study: First, we only explored navigation difficulties encountered by experienced developers, who were totally blind, actively engaged in programming either as a job or hobby, and used assistive technologies to access the computer display (e.g., screen reader, braille display, or both). It was beyond our scope to study novice programmers or individuals with greater diversity in their visual acuity. A further investigation into such an important user group may reveal different findings. Secondly, while the qualitative design of this study allowed us to gather firsthand comments and experiences from our user group, and to discover new issues that arose, in future work, it may be important to follow up this study with a survey administered to a larger group of participants, to verify some of our findings.

The navigation challenges identified in this study illustrate the need for further research on improving the usability and accessibility of current IDEs. For example, participants showed interest in using a new forms of code navigation, e.g. using hierarchical navigation approaches. Participants also indicated a desire for bookmarks (or tags) features that would allow blind programmers to tag specific line of code and return to it later for further modification. They also expressed interest in scope and nesting level indicator, auditory additional feedback, and methods for conveying class relationships, which could make programming more accessible for these users.

Finally, while the participants in our study expressed interest in various technology interventions to address their needs, it would be necessary in future work to conduct formal evaluations of the efficacy of such technology in studies with blind developers. In fact, we are specifically planning, in our future work, to explore some form of auditory feedback which could help convey important information while users are navigating through lengthy codebases. Several participants expressed interest in this technology. Participants also suggested that audio cues could be used in various other programming activities. We plan to conduct participatory design research to understand how to best use auditory cues in a code navigation system.

In summary, the results of this study provide future accessibility researchers a foundation for understanding the needs of blind programmers, which may support their work in creating and evaluating new technologies to address those needs.

7. ACKNOWLEDGMENTS

We acknowledge scholarship support from the King Salman Scholarship Program. We would like to thank our participants and our anonymous reviewers for their valuable feedback and insights.

8. REFERENCES

[1] K. Albusays and S. Ludi. 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*, pages 82–85. ACM, 2016.

[2] C. M. Baker, L. R. Milne, and R. E. Ladner. 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*, pages 3043–3052. ACM, 2015.

[3] H. Beyer and K. Holtzblatt. *Contextual design: defining customer-centered systems*. Elsevier, 1997.

[4] D. B. Boardman, G. Greene, V. Khandelwal, and A. P. Mathur. Listen: A tool to investigate the use of sound for the analysis of program behavior. In *Computer Software and Applications Conference, 1995. COMPSAC 95. Proceedings., Nineteenth Annual International*, pages 184–189. IEEE, 1995.

[5] A. Bragdon, R. Zeleznik, S. P. Reiss, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola Jr. Code bubbles: a working set-based interface for code understanding and maintenance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2503–2512. ACM, 2010.

[6] S. A. Brewster. Using nonspeech sounds to provide navigation cues. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(3):224–259, 1998.

[7] J. M. Francioni and A. C. Smith. Computer science accessibility for students with visual disabilities. In *ACM SIGCSE Bulletin*, volume 34, pages 91–95. ACM, 2002.

[8] A. Z. Henley and S. D. Fleming. The patchworks code editor: toward faster navigation with less code arranging and fewer navigation mistakes. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pages 2511–2520. ACM, 2014.

[9] M. Konecki, A. Lovrenčić, and R. Kudelić. Making programming accessible to the blinds. In *MIPRO, 2011 Proceedings of the 34th International Convention*, pages 820–824. IEEE, 2011.

[10] J. Lamping, R. Rao, and P. Pirolli. A focus+ context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408. ACM Press/Addison-Wesley Publishing Co., 1995.

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

[12] National Science Foundation, National Center for Science and Engineering Statistics. 2017. *Women, Minorities, and Persons with Disabilities in Science and Engineering: 2017*. Special Report NSF 17-310. Arlington, VA. Available at www.nsf.gov/statistics/wmpd/.

[13] D. K. Palladino and B. N. Walker. Learning rates for auditory menus enhanced with spearcons versus earcons. Georgia Institute of Technology, 2007.

[14] C. Parnin and A. Orso. Are automated debugging techniques actually helping programmers? In *Proceedings of the 2011 international symposium on software testing and analysis*, pages 199–209. ACM, 2011.

[15] J. Saldaña. *The coding manual for qualitative researchers*. Sage, 2015.

[16] J. Sánchez and F. Aguayo. Blind learners programming through audio. In *CHI'05 extended abstracts on Human factors in computing systems*, pages 1769–1772. ACM, 2005.

[17] J. Sánchez and F. Aguayo. Apl: audio programming language for blind learners. *Computers Helping People with Special Needs*, pages 1334–1341, 2006.

[18] R. M. Siegfried. Visual programming and the blind: the challenge and the opportunity. *ACM SIGCSE Bulletin*, 38(1):275–278, 2006.

[19] A. C. Smith, J. S. Cook, J. M. Francioni, A. Hossain, M. Anwar, and M. F. Rahman. Nonvisual tool for navigating hierarchical structures. In *ACM SIGACCESS Accessibility and Computing*, number 77-78, pages 133–139. ACM, 2004.

[20] A. C. Smith, J. M. Francioni, and S. D. Matzek. A java programming tool for students with visual disabilities. In *Proceedings of the fourth international ACM conference on Assistive technologies*, pages 142–148. ACM, 2000.

[21] A. Stefik, R. Alexander, R. Patterson, and J. Brown. Wad: A feasibility study using the wicked audio debugger. In *Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on*, pages 69–80. IEEE, 2007.

[22] A. Stefik, C. Hundhausen, and R. Patterson. An empirical investigation into the design of auditory cues to enhance computer program comprehension. *International Journal of Human-Computer Studies*, 69(12):820–838, 2011.

[23] A. M. Stefik, C. Hundhausen, and D. Smith. 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*, pages 571–576. ACM, 2011.

[24] S. F. A. Szpiro, S. Hashash, Y. Zhao, and S. Azenkot. 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*, pages 171–180. ACM, 2016.

[25] P. Vickers and J. L. Alty. When bugs sing. *Interacting with Computers*, 14(6):793–819, 2002.

[26] B. N. Walker, A. Nance, and J. Lindsay. Spearcons: Speech-based earcons improve navigation performance in auditory menus. Georgia Institute of Technology, 2006

9. ONLINE APPENDIX

Supplemental materials have been uploaded to the ACM Digital Library to accompany this paper, including two comma-separated value (CSV) files: (1) a table of additional quotations from participants and (2) a table listing the operating system, assistive technologies, programming languages, and programming editor uses by each participant (complementing the summarized information found in Table 2 in this paper). In addition, these materials are also available online at our laboratory website at the following URL: http://latlab.ist.rit.edu/assets2017code