



# Accessible Web Development: Opportunities to Improve the Education and Practice of web Development with a Screen Reader

CLAIRE FERRARI and AMY HURST, New York University, New York

There are a growing number of jobs related to web development, yet there is little formal literature about the accessibility of web development with a screen reader. This article describes research to explore (1) web development accessibility issues and their impact on blind learners and programmers; (2) tools and strategies used to address issues; and (3) opportunities for creating inclusive web development curriculum and supportive tools. We conducted a Comprehensive Literature Review (CLR) to formulate accessibility issue categories, then interviewed 12 blind programmers to validate and expand on both issues in education and practice. The CLR yielded five issue categories: (1) visual information without an accessible equivalent, (2) orienting, (3) navigating, (4) lack of support, and (5) knowledge and use of supportive technologies. Our interview findings validated the use of CLR-derived categories and revealed nuances specific to learning and practicing web development. Blind web developers grapple with the inaccessibility of demonstrations and explanations of web design concepts, wireframing software, independent verification of computed Cascading Style Sheets (CSS), and navigating browser-based developer tool interfaces. Tools and strategies include seeking out alternative education materials to learn independently, use of CSS frameworks, collaboration with sighted colleagues, and avoidance of design and front-end development. This work contributes to our understanding of accessibility issues specific to web development and the strategies that blind web developers employ in both educational and applied contexts. We identify areas in which greater awareness and application of accessibility best practices are required in Web education, a need to disseminate existing screen reader strategies and accessible tools, and to develop new tools that support Web design and validation of CSS. Finally, this research signals future directions for the development of accessible web curriculum and supportive tools, including solutions that leverage artificial intelligence, tactile graphics, and supportive-online communities of practice.

CCS Concepts: • **Human-centered computing** → *Accessibility*;

Additional Keywords and Phrases: Accessibility, Human-centered computing, visually impaired/blind programmers, screen reader, accessible web development, accessible design tools, accessible web design

## ACM Reference format:

Claire Ferrari and Amy Hurst. 2021. Accessible Web Development: Opportunities to Improve the Education and Practice of web Development with a Screen Reader. *ACM Trans. Access. Comput.* 14, 2, Article 08 (July 2021), 32 pages.

<http://doi.org/10.1145/3458024>

Authors' addresses: C. Ferrari and A. Hurst, New York University, 82 Washington Square East, Pless Hall, Department of Occupational Therapy, 6th Floor, New York, NY 10003; emails: {ckv214, amyhurst}@nyu.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2021 Association for Computing Machinery.

1936-7228/2021/07-ART08 \$15.00

<http://doi.org/10.1145/3458024>

## 1 INTRODUCTION

As web technologies become more ubiquitous [1, 32] web development subfields such as full-stack, frontend and backend engineering, product and user-experience design, and web accessibility continue to emerge and expand [23]. These roles require general web development knowledge and skills as part of day-to-day job functions [23]. According to the Bureau of Labor Statistics, general knowledge and skills are centered around graphic design and web development programming languages [23]. Projections by the U.S. bureau of labor statistics predict significant growth for Web Development and Web Development-adjacent jobs in the next decade [22]. However, persons with disabilities are still significantly underrepresented [58].

In 2017, the employment rate for persons with a visual disability was 44.2% and the average household income was \$45,500 [30]. These figures seem staggeringly low when compared with persons without disabilities (75.5% employment and average household income of \$68,000) [31]. The field of Computing has a long-standing equity problem—with a lack of representation of women, racial and ethnic minorities, and people with disabilities [54]. Although there are not specific statistics about representation of blind individuals in Computing, people with disabilities make up about 13% of the population in the United States, and less than 7% of Computer Science and Engineering degree graduates have disabilities [55]. A complex array of factors contribute to underrepresentation, including socio-economic, educational, and systemic barriers [33].

Web development involves the creation of new technologies while relying on a wide range of ever-expanding technical tools [24]. Although myriad evolving web publishing platforms, development libraries, and frameworks support web development, three core technologies remain central to the practice: HTML, CSS, and JavaScript [24]. Here, we describe and define relevant web and assistive technologies, how they interact, and digital accessibility best-practices put forth by the World Wide Web Consortium in the form of **Web Content Accessibility Guidelines (WCAG)** [80].

First, we focus our definition on the three main languages of the web: **Hypertext Markup Language (HTML)**, **Cascading Style Sheets (CSS)**, and JavaScript. HTML is a markup language that describes the structure and content of web pages [82]. CSS is a language that defines the visual presentation of web sites. Like HTML, CSS has a defined syntax and can be used to address and change a site's layout, colors, and fonts [81]. JavaScript is a scripting language that generates interactivity on websites and when introduced on the client-side, allows end-users to control multimedia, and experience dynamically updating content [53]. Each of these primary web languages has a unique set of rules and syntax that enables it to manipulate web page content and presentation.

To access web information, screen readers provide a non-visual interface to users that are blind or have low vision by communicating on-screen information via speech synthesis or refreshable Braille output. On the web, screen readers convey the semantic properties (roles, states, and properties) of **document object model (DOM)** that is the basis of the **user interface (UI)**. It is important to note that with few exceptions, the output of CSS has no impact on the semantic information conveyed to screen readers. Additionally, JavaScript does not inherently communicate state and dynamic content changes to screen readers. Many issues that blind screen reader users face when using web content stem from a lack of screen reader-compatible semantic structure and interactivity. WCAG is a series of design and development guidelines intended to support accessible content for screen reader users [80]. WCAG's authoring counterpart ATAG [83] is lesser known but intended to support the accessibility of creation tools. ATAG is a general guide written primarily for authoring tool developers, but it also includes resources for users wanting to request accessibility features in authoring tools, as well as for managers and policy-makers [83].

With web-related courses offered at the college-level, and many more offered free online, or as part of boot camp certification programs—there are numerous pathways to pursue careers in

web development [24]. The accessibility of education and technologically mediated workflows are likely factors in the representation of blind web developers. However, there has been little formal investigation of what navigating web development education and web-related career pathways are like as a blind screen reader user. The literature on computing accessibility has primarily focused on non-web Computer Science education and practice. To address this gap in the literature, we explore accessibility issues that screen reader users face when learning and practicing web development, the workarounds or strategies they employ, and opportunities to design curricula and tools to better support blind web developers.

The remainder of this article is dedicated to Related Work, Methods, Findings, and Discussion sections. The Related Work section provides an overview of relevant technologies and prior research in the area of programming accessibility and inclusive curricula. Our Methods section outlines our process of reviewing both traditional and non-traditional literature sources for web-specific issues, our observational process from pilot work to support the accessibility of web development, and finally, interviews blind web developers. Our Findings contribute to the definition of web development-specific accessibility issues, strategic workarounds, and opportunity areas to support a more inclusive approach to teaching and practicing web development. This section is divided by CLR-identified issue categories and data sources, followed by findings from interviews with blind programmers that have helped to validate and elaborate on web development-specific issues and strategic workarounds. In our Discussion, we frame findings as core contributions and directions to researchers, educators, and blind students interested in pursuing this work.

This research is intended to explore and deepen the discourse about accessibility issues and opportunities to support blind web developers. By encouraging more diverse participation and incorporating the perspectives of people with disabilities in the development of new technologies, support for accessibility and general usability increases [80]. In learning novel ways to support blind students and distributing these learning materials and training, this work will contribute to an increase in blind people's participation in web development and adjacent fields, and ultimately, the accessibility and democratization of the web.

## 2 RELATED WORK

Here, we provide an overview of prior research in the area of programming accessibility and inclusive curricula. Although web development-specific research is scarce, related work has been done to develop supportive software for blind students and programmers. The focus of this research has largely been the development and evaluation of tactile interfaces, audio and speech-based interfaces, and software for navigating the structure of code. There is also a body of work related to the development of born accessible programming languages for teaching and learning and inclusive curricula designed for use by blind programmers and students. Here, we outline related work in the areas of tactile and audio interfaces, as well as navigating code structures and accessible **Computer Science (CS)** education.

### 2.1 Tactile Interfaces

The use of tactile interfaces to support learning has been explored as an effective way to represent visual and spatial information non-visually [26]. Blind students report that tactile graphics are useful and make them feel included in learning environments [85]. Research on the educational benefits of tactiles suggest that students can enhance their symbolic understanding and knowledge when using tactile graphics [79], and that 2.5D tactile graphics, or graphics that have more tactile resolution than one dimension but are not quite three-dimensional, support learning of a variety of subjects [78].

As many programming concepts are taught through visual and spatial means, Kane and Bigham [41] developed a workshop in which they taught students that are blind and low-vision students how to write computer programs for analyzing Twitter data to produce 3D printed visualizations that allowed for a tactile exploration of their program output. The tactile outputs were an accessible mode of communicating data concepts and stimulated student engagement in programming [41]. Additionally, researchers created and studied Torino—a physical programming language created to teach programming constructs and computational thinking to children (7–11 years) [75]. The system is composed of several connectable beads or nodes that when attached in specific ways by node-type produce audio feedback. Their work shows how children make meaning through their system design (e.g., physically following the program when it becomes complex or during debugging) and design of programming tasks and social interaction [75]. Li et al. [47] produced a tool that outputs tactile print-outs of web-page layouts to support blind developers in understanding spatial relationships between elements and modify page layouts. The evidence that supports the use of tactile manipulatives in educational settings has informed our decision to explore its use in the instruction of web development. In particular, we explore the use of 2.5D tactile designs in teaching and learning web design concepts (alignment, font-family, font-size, and the box model).

## 2.2 Audio Interfaces

Like the speech synthesis outputs of screen readers, audio interfaces for programming accessibility act as a non-visual interface but are less robust and verbose. As many graphical user interfaces for programming offer visual cues and features to enhance programmer efficiency, audio interfaces for programming accessibility are designed for programming tasks and use tonal feedback or cues. Sanchez and Aguayo [65] developed an **audio programming language (APL)** and found preliminary support that these interfaces can “fit the needs and mental models of blind learners to motivate and help them to enter the programming world” (p. 1769). Stefik et al.’s [72] empirical study of the differences between auditory cue-types showed that their empirically designed auditory cues (music, tones, and words derived through artifact encoding studies) are significantly better than screen reader + audio cues added on top of Visual Studio’s [51] development environment features. They showed that well-designed audio cues can enhance performance to the level of performance with visual representations [72]. Unfortunately, APL is not a robust language and can only perform simple isolated problem-solving tasks [62]. Other tonally based enhancements are again for users already familiar with Visual Studio’s complex development environment and require intensive study to be useful. As is such, these tonally based interface enhancements are not a robust solution and not suited for use in an introductory web development workshop.

Although audio enhancements have made significant contributions, the work discussed here will leverage core web language accessibility features, WCAG best practices, and tactile tools. In contrast to audio-based enhancements, web language accessibility features and WCAG are widely used, robust, actively maintained, and have ample documentation for use.

## 2.3 Navigating the Structure of Code

As screen readers relay information in a line-by-line, word-by-word, or character-by-character linear fashion, it can be difficult for users to understand complex code structures and navigate them efficiently [13]. Smith et al. developed a series of requirements for a non-sighted tree-structure navigation and an aural code tree navigation tool, which was then tested with blind students [71]. Students found that the tool helped them navigate code easily and efficiently [71]. Similarly, Baker et al.’s study of an accessible code tree-view IDE plug-in tool helped blind students better understand the structures of code [13]. More recently, Schanzer et al. developed a tool that describes program structure and allows for keyboard shortcut navigation [67]. Their research suggests that

the use of this tool helps blind programmers feel better about navigation tasks and improves their accuracy and code orientation without significant changes in task completion time [67]. These findings have informed our own program navigation and orientation subsections in our web development workshop, discussed shortly.

## 2.4 Accessible CS Education

Within CS education, there is a body of work related to the research and development of inclusive programming curricula designed for use by blind students. Most notably, Stefik et al. [73] developed and studied the use of an accessible development environment, accessible language, and a multi-sensory curriculum for blind students. Their empirical study of a workshop with blind participants produced evidence of a significant increase in programming self-efficacy. Through observation and interviews Stefik et al.'s work yielded three suggestions for programming curriculum development with blind students: (1) Use manipulative objects that students can touch and hold to introduce concepts, where possible; (2) Favor hands-on activities and projects over lecturing; and (3) Provide an array of self-paced projects and activities to engage students [73]. Baker et al. also found that by programming in a supportive environment, blind students gain confidence to persevere in computer science [14]. They also found that creating course material in electronic format is best, because it can be converted to alternative formats (large-print, electronic speech, Braille, etc.) and that testing materials early and extensively with potential end-users and assistive technologies is beneficial to programs of this nature [14]. Stefik and Ladner [74], have also been active in developing the evidence-based programming language Quorum, which was designed as a teaching tool for basic to advanced concepts in computation [74].

This prior research on inclusive programming curricula for blind students has informed our decision to explore the use of manipulatives, hands-on collaborative lessons, and scaffolding of supportive tools in our accessible web development workshop. However, none of this work has focused on web-development specifically, and authors have not taken a practical approach to teaching web development for vocational training, including instruction in markup (HTML), Styling (CSS), and JavaScript. Our research and development work focus primarily on web development as a practical creative tool and vocational skill.

## 3 METHODS

We employed a variety of methods to explore web development accessibility issues, strategies and workarounds, as well as opportunities to develop inclusive tools and curricula. Our methodology can be divided into two main methods: (1) a **Comprehensive Literature Review (CLR)** of existing data from traditional literature review sources, blog posts, email list threads, and observations from our previously published research, and (2) interviews with blind Web developers that were shaped in part by findings of the CLR.

### 3.1 Comprehensive Literature Review

To address the gap in literature about web development-specific accessibility issues for people that are blind, we utilized CLR methods as outlined by Reference [59]. CLR is a systematic approach to establishing a literature review from multiple sources as a methodology. Here, we have employed a CLR to bring together multiple viewpoints that may not be present in a traditional literature review. The process includes traditional literature searches (books, articles, proceedings, etc.) and involves expanding the search to include one or more **MODES (Media, Observation, Documents, Experts)**, before final analysis and synthesis of information [59]. To better understand accessibility issues that screen reader users face when learning and practicing web development, this literature review included the evaluation of blog articles authored by blind programmers, blind



programmer email list exchanges, as well as observations from pilot workshops to identify web-specific accessibility issues.

**3.1.1 Traditional Literature Review.** The traditional literature review consisted of various accessibility, HCI, and computer science books, journal articles, and conference proceedings. As part of our literature review and throughout this article, we use the term “blind” not only because it is the preferred term by some people in the disability community, but because we are primarily interested in blind users and not a range of acuity that is captured by the term visual impairment. We acknowledge that terms like “visually impaired” and “visually impaired person” are widely used, however, we feel that this language emphasizes functional damage that many people that are blind do not identify with and also captures a variety of technologies and interaction modalities that are not the focus of this research (low-vision tech such as magnification software). We began our literature review by searching the terms “blind programming,” “blind programmer,” “blind coding,” “blind computer science,” “blind computer science education,” and “accessible computer science” to search Google Scholar. We then used citation information to branch out and add to a list of related articles that inevitably led us to a wide range scholarship related to blind programming education and practice. This process resulted in a compiled list of 30 articles that mention challenges or issues faced by blind programmers.

**3.1.2 Blog Posts by Blind Programmers.** By searching terms “blind programmer” and “blind web developer” using Google as an Internet search engine, we were able to find five blog articles authored by three blind web developers (Florian Beijers, Parham Doustdar, Tuukka Ojala) that contained content related to issues faced when doing web development. These posts include Florian Beijers’ blog posts titled “*A Vision of Coding, without Opening your Eyes*” on Freecodecamp’s Medium.com account [16] and “*Stories from the Trenches: What I’ve learned from Working as a Blind Developer for a Sighted Dev Team*” on the website 24a11y.com [17]; Parham Doustdar’s personal posts titled “*An Autobiography of a Blind Programmer*” [28] and “*The Tools of a Blind Programmer*” [27]; and Tuukka Ojala’s post titled “*Software development 450 words per minute*” posted on the website Vincit.fi [60].

**3.1.3 Email List Threads.** We joined [program-l@freelists.org](mailto:program-l@freelists.org)—an active mailing list on the topic of blind programming for a period of six months. This free community list is used by blind programmers to ask questions and share resources among members. We monitored email list exchanges with 277 unique subjects and noted those that related to web development specifically. In the six-month period, we identified 12 threads related to web development and noted the issues, respondent workarounds, and resources shared.

**3.1.4 Observations from Accessible Web Development Workshops.** Our fourth dataset in the CLR consisted of observations from our pilot work teaching web development workshops to students that are blind. In a CLR, observations function as examples that help strengthen understanding regarding the topic and concepts [59]. Over the past several years, we have taught a variety of web development workshops and classes geared towards blind adults. Observations of web development student accessibility issues are derived from two prior multiday web development workshops. The first was taught at a holiday technology camp in Gulu, Uganda, in January of 2018 [43], and the second was taught at a New York Library in August of 2018 [45]. The following sections are dedicated to describing the workshops from which observations are derived.

**3.1.4.1 Oysters and Pearls Technology Camp Workshop.** This project included the development and evaluation of a six-day web development curriculum taught to 13 blind students aged 18–35 [43]. Students had all received **Job Access with Speech (JAWS)** [33] certification training prior to

attending but had no previous web development experience (one student had some prior non-web programming experience). All students in the workshop used JAWS as their primary screen reader and were provided Windows laptops with the JAWS installed. Our inclusion criteria were blind adults, proficient with screen readers, and interested in web development. As the organization that we partnered with serves individuals that are blind, we did not collect data on student vision, color perception, or whether students had experience with other screen readers. The curriculum covered an introduction to the web, HTML, CSS, and JavaScript. The curriculum included the use of tactile diagrams and an accessible web-based **Red Green Blue (RGB)** color mixing and naming tool [43]. The main methods for gathering data were observations of students in the classroom as well as post-course focus group and survey feedback [43]. Time and resource restrictions prevented us from conducting onr-on-one interviews. Observations of students in the classroom included their reactions to the classroom environment and curriculum, questions they had, as well as the direct feedback they gave at the end of each day. These observations were noted by instructors and reviewed at the end of each day.

*3.1.4.2 New York Library Workshop.* This project included the development and evaluation of an eight-day course in web development taught at a New York Library [45]. A total of 14 blind students began and completed the workshop. Our inclusion criteria were blind adults proficient with screen readers and interested in web development. We did not collect additional data on student vision or color perception. Students had little to no experience building websites prior to attending (one student had taken an HTML class over a decade ago, and another had edited HTML while using a CMS). Primary screen reader use was mixed (9 students reported that JAWS [33] was their primary, 3 said NVDA [58], and 2 said Voiceover [7]), and 4 students reported using more than one screen reader on laptop or desktop. Students in the workshop had access to high-speed Internet and laptops with software installed (JAWS, the text editor Notepad++ [40], and File Transfer manager Cyberduck) were provided, if needed. The curriculum spanned units on an introduction to the web, HTML, CSS, and JavaScript. Students built a website over the course of the workshop, and the final day was dedicated to project development. Units were created in HTML, uploaded to the students' computer desktops, and updated daily with any changes and additions. Time and resource restrictions prevented us from conducting one-on-one interviews. However, we collected data using an adapted self-efficacy and interest scale administered pre- and post-workshop and quizzes administered at the end of each workshop unit. Our CLR will draw solely from observations of students in the classroom. This observational data included reactions to the curriculum, questions they had, as well as the direct feedback they gave at the beginning of each lesson.

## 3.2 Interviews with Blind Web Developers

Interviews were conducted with blind web developers to further explore web development-specific accessibility issues, workarounds, and areas of future work. Participants were recruited from open email lists and Twitter, with the inclusion criteria of using a screen reader, having some web development experience, and being over the age of 18. Remote interviews lasted approximately one-hour each and included questions about participant disability status, assistive tech use, web development education, and languages used. Following demographic and background questions, we included open-ended questions about recurring issues encountered when learning or practicing web development. After discussing the issues that participants had, we asked a series of questions aimed at validating issue categories derived from the literature. Participants were asked whether they had experienced issues related to each broad category, then the interviewer probed with specific issues in each category (Table 3). If participants had experienced the issue, then they

Table 1. Issue Groups and Specific Issues as Identified in the CLR. L=Literature, B=Blogs, E=email list, and O=Observations

Accessibility Issue Category	Specific Issue	Source
1. Visual information without an accessible equivalent	1.1 Integrated Development Environment Visual Accessibility Issues	L, O
1. Visual information without an accessible equivalent	1.2 Whitespace for formatting	L, B, E, O
1. Visual information without an accessible equivalent	1.3 Supportive Graphs/Diagrams	L, B, E, O
1. Visual information without an accessible equivalent	1.4 CSS and Visual Design	L, B, E, O
2. Orienting	2.1 Orientation (comprehension of program structure, and debugging)	L, E, O
3. Navigation	3.1 Keyboard navigation	L, E, O
3. Navigation	3.2 Navigating Interfaces	L, O
4. Lack of Support	4.1 Lack of support (e.g., from instructors, employers, colleagues)	L, B, E, O
5. Use of Supportive Tech	5.1 Issues related to use of or proficiency with assistive technologies or productivity software	L, B, E, O

were asked to describe it in detail. We then asked if participants could think of any additional issue categories outside of those described. Participants were then asked to rate the severity of the impact of these issues. To evaluate the impact of accessibility issues during interviews with blind programmers, we used a rating system based on Dumas and Reddish's [29] levels of severity in human-computer interaction. This rating scale is commonly used in the study of usability and found in HCI textbooks [33, 36, 47, 61]. Severity ranges from 1–3 (1 = minor effect on usability, 2 = creates significant delay and frustration, and 3 = blocker to task completion). This was used to assess the severity of usability issues in the context of non-visual, screen reader-assisted web development. Finally, we asked participants what they would create or have created for them to make the practice of web development easier/better and what advice they would give to new students. Interviews were conducted remotely using Zoom conferencing software [86], and each session was recorded and auto-transcribed. Once complete, recordings and transcriptions were given multiple readings, and classical content analysis strategies [15] were used to form initial codes. Then a line-by-line coding process, followed by memo-writing and axial-coding processes, were applied to develop a narrative/propositional statement about Blind web developer experiences of learning and practicing web development [24].

#### 4 CLR FINDINGS

The traditional literature review, combined with themes from blogs, email list threads, and observations, informed initial findings. Together, these data sources supported the comprehensive literature review and revealed various accessibility issues related to learning and practicing web development. We aggregated issues from sources forming an initial coding scheme to organize accessibility issues (Table 1). Using qualitative content analysis, issues underwent systematic abstraction and categorization [68]. This process yielded five categories that are potentially relevant to the practice of web development. Broadly, these issues can be organized into (a) visual information without an accessible equivalent, (b) orientation, (c) navigating, (d) lack of support, and



(e) use of supportive technology. Web development-specific issues identified are related to inaccessible demonstrations of CSS/web design concepts, the inaccessibility of wireframing software, the inability to independently verify CSS outputs, and trouble navigating browser-based developer tool interfaces. Traditional literature review source findings skewed towards more non-web programming accessibility issues. Non-traditional sources (blog, email lists, and workshop observations) provide a more nuanced picture of practicing web development with a screen reader—including valuable strategies and workarounds. Interviews helped us to elaborate on CLR findings, understand issue categories in the context of web development, and hone in on additional web development-specific issues.

In this section, we first describe the five Issue Categories in detail as they are found in the traditional literature sources, blogs, email lists, as well as workshop observations. We then describe findings from the interviews, which were used to validate CLR issue findings and gain a better understanding of their impact, as well as provide descriptions of issues related to education, workarounds, and opportunities for development.

## 4.1 Visual Information without an Accessible Equivalent

**4.1.1 IDE Visual Accessibility Issues.** Visual information without an accessible equivalent is an issue that is pronounced when using the graphical interfaces of **Integrated Development Environments (IDEs)**, interpreting whitespace, graphs, and diagrams, and outputs of visual design.

**4.1.1.1 Literature Related to IDE Visual Accessibility Issues.** Accessibility issues related to IDEs is mentioned consistently in the literature [71, 13, 5, 3, 14, 42, 61]. These authors often cite a lack of communication between visual information on the screen and what a screen reader relays to users.

### Synthesis of Multiple Visual Cues

Potluri's conception of glanceability [61] refers to IDE **graphical user interface (GUI)** visually dense arrangement of elements and sighted user's ability to quickly glean information from it. Efficient programming requires the ability to glance at and synthesize a number of visual stimuli. Particularly, the glanceability of the structure of code/programs as a whole (Figure 1), relevant context surrounding the current line of code, and glanceability of cues from multiple regions (like the console and line-specific syntax highlighting) that taken together give information and clues about the functioning of code, are all important aspects of efficiency in using IDE GUIs [61].

### IDE Error Cueing

A key feature of IDEs is that they provide users visual information to identify coding errors in real-time while typing. Most IDEs do not support screen reader accessibility of real-time error messages as they are conveyed through visual cues (Figure 2) and do not possess text alternative information or utilize additional auditory cueing [13, 61].

### Syntax Highlighting

Another valuable feature of many IDEs is their capacity to differentiate and communicate parts of programming language syntax. IDEs do this by first breaking down the text of code into a list of "tokens," match them with parts of a predefined grammar/syntax rule-set, and then assign them different colors and styles based on a theme [66]. The appearance of code drastically changes when syntax highlighting is applied (Figure 3), with patterns of syntax and structure emerging clearly. Although there is evidence that syntax highlighting can improve program comprehension, particularly in novice programmers [66], the visual syntax highlighting feature of IDEs has no coherent screen reader equivalent in mainstream IDEs, and this presents an accessibility issue for screen reader users seeking equivalence in functionality [13].





Fig. 3. Sublime Text’s syntax highlighting. The image on the left is without syntax highlighting, and the image on the right is with syntax highlighting.



Fig. 4. Sublime Text IDE’S visual cueing for saved (“x” icon) versus unsaved (circle icon) files.



Fig. 5. Current line in focus is indicated to users with a light grey highlight of line number (line 3 is in focus).

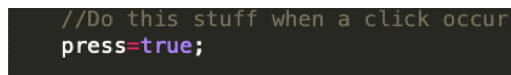
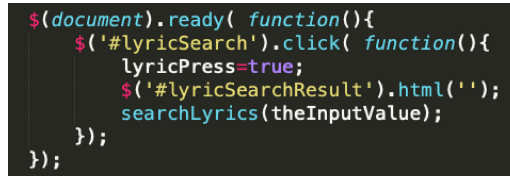


Fig. 6. Embedded comments are rendered as a faded grey to allow sighted programmers to glance over them, but focus on the code.

Typographic accessibility issues include difficulty perceiving characters, operators, and parentheses, and there is also a lack of accessibility in using autocomplete features [5]. Although all of this information is discoverable by screen readers in several steps (e.g., navigating to a specific line of code and moving through the line character-by-character), there are not accessible equivalents to many of these subtle visual cues.

**4.1.1.2 Observations Related to IDE Accessibility Issues.** In our observations of workshop students that are blind or low-vision, IDE-related issues were generally not as prevalent as they are in the literature [43, 45]. This is due, in part, to the fact that we are aware of the accessibility that many IDEs have, and because their complexity does not make them ideal teaching/learning tools in short introductory web development courses. As a result of known IDE accessibility issues and complexity, we offered training using simplified text editors (Notepad [50], TextEdit [8]), or the accessible IDE for Windows, NotePad++ [40], at each student’s discretion. One notable exception was an ambitious student from the Library workshop. This student installed and intended to use XCode [9] for Apple OS prior to the course but found it very cumbersome to learn in such a short period that he abandoned it in favor of a TextEdit. To compete with the efficiency that IDEs afford sighted developers, students and practitioners of web development need access to IDEs and their features. Although the majority of our students did not encounter this issue, if they continue to pursue development work, and their code increases in length and complexity, then the necessity for IDEs may become more apparent.



```
$(document).ready( function(){
    $( '#lyricSearch' ).click( function(){
        lyricPress=true;
        $( '#lyricSearchResult' ).html('');
        searchLyrics(theInputValue);
    });
});
```

Fig. 7. “Whitespace” indentation used to organize code and make it easy to identify openings and closings of statement.

#### 4.1.2 Whitespace.

**4.1.2.1 Literature Related to Whitespace Issues.** Whitespace is a by-product of using the tab and space keys to create indentation. Whitespace is used strategically to format code in a way that makes it easier to identify nested relationships and structure, including to identify where statements begin and where statements end (Figure 7). In the language Python, levels of indentation are interpreted meaningfully and define scope and group statement [84]. In this case, a user’s ability to identify and interpret whitespace is vital to producing and understanding code. Whitespace has been identified as an issue when using a screen reader, as it can be missed or harder to interpret—unless the user navigates character-by-character and counts to keep track of spaces. In Albusays et al.’s observations of blind programmers, these researchers reported that 22 of 28 blind programmers were observed struggling with indentation and whitespace [5].

**4.1.2.2 Blogposts about Whitespace Issues.** The blogs that we reviewed did not detail issues related to whitespace. In fact, we found an author describing how he used whitespace to organize and structure code. Tuukka Ojala writes, “Proper indentation helps me just as much as it does a sighted programmer. Whenever I’m reading code in Braille (which, by the way, is a lot more efficient than with speech) it gives me a good visual clue of where I am, just like it does for a sighted programmer. I also get verbal announcements whenever I enter an indented or un-indented block of text” [60].

**4.1.2.3 Email List Threads about Whitespace Issues.** Among the email list threads that we reviewed, three dealt with questions and resources related to whitespace. Two of these were related to inquiries about how to begin programming in Python with a screen reader. Members shared that there are ways to read whitespace including using a plain text file with the IndentNav add-on for NVDA—which reads a hierarchically structured list of ideas marked by indentation levels, using an IDE with relatively good accessibility support—including Visual Studio Code [51] and IDLE [62], as well as Python optimized IDE add-ons (from *program-l@freelists.org*).

**4.1.2.4 Observations of Whitespace Issues.** In our prior web development workshops [43, 45], we discussed conventions for using whitespace to organize code, primarily for sharing with sighted collaborators. We created tactile representations of the shape of HTML (Figure 8), CSS and JavaScript with formatting/whitespace, and shared resources for settings that announce indentation to screen readers. Because this was taught as a best practice for sharing with sighted programmers, and we were not using a language that interprets whitespace, whitespace did not present an immediate usability issue to our web development students.

#### 4.1.3 Graphs and Diagrams.

**4.1.3.1 Literature Related to Graphs and Diagrams.** In Computer Science, students and software developers are often required to use graphs and diagrams during the development process [12, 5, 42]. In web and non-web CS education, instructors include graphs and diagrams of many

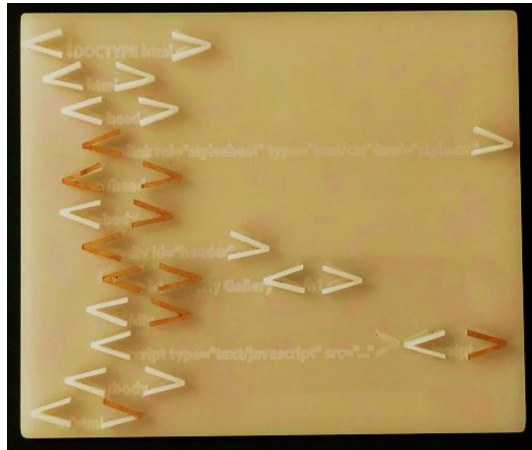


Fig. 8. Tactile representation of whitespace/indentation in HTML used to organize code. Made with laser-cut acrylic.

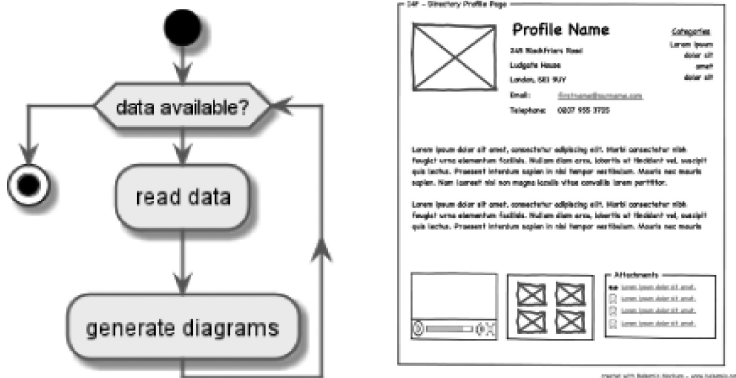


Fig. 9. UML diagram of a while loop (source: <https://plantuml.com/activity-diagram-beta>) and web profile page wireframe (source: <https://commons.wikimedia.org/wiki/File:Profilewireframe.png>).

things: database administration processes, site maps, and wireframes (see Figure 9), automata, decision trees, and resource-allocation diagrams that are not accessible without the addition of screen reader support or provision of alternative text descriptions [12]. In their survey of 69 blind programmers, Albusays and Ludi found that several participants mentioned this problem and the need for assistive tools to interpret diagrams [3]. Tools, including Swell Form and other tactile diagramming techniques to support the accessibility of CS diagrams and graphs, have been developed as separate systems for blind users and not integrated into the same interface that sighted users use [12]. Additionally, although diagrams can be made accessible by providing alternative text descriptions, acquiring good descriptions in a timely manner is often impossible [3].

**4.1.3.2 Blog Posts Related to Graphs and Diagrams.** We found one blog post that described accessibility issues related to graphs and diagrams. Specifically, the author wrote about the requirement of using UML charts in university as part of lessons and activities related to the structure of databases. The lack of accessibility support and lack of support from professors in addressing this issue was a turning point for this author. He writes, “This constant pressure, coupled with



unwilling professors that would rather get their job done and get home, made me realize that it is not worth it to go through an education system that I had to fight with all the time. That was when I quit” [27].

**4.1.3.3 Email List Threads Related to Graphs and Diagrams.** We found two mentions of accessibility issues related to graphs and diagrams in blind programmer list emails. Specifically, two members wrote to the list seeking to find a solution to translate UML diagrams into navigable and meaningful XML structures. In response, a member suggested that although options are limited, if a UML diagram was created using the UML drawing tool PlantUML,<sup>1</sup> then they could potentially access the markup text that was used to create the diagram. Another member suggested exploring tools to parse UML, JSON, and XML into tree structures.

**4.1.3.4 Observations Related to Graphs and Diagrams.** Graphs and diagrams used to convey CS concepts were not used in our previously taught web development workshops. However, in web development wireframing is a common practice and helpful in planning information architecture and designing user experience. The tools commonly used to create static wireframes are not accessible, and to address wireframing, we have had workshop students write ordered, nested lists of intended elements and user interactions [43, 45]. This method lacks the dynamic, non-linear affordances of complex visual wireframes, but has been helpful to get them thinking about page layout and user experience.

#### 4.1.4 CSS and Visual Design.

**4.1.4.1 Literature Related to CSS and Visual Design Issues.** Mealin and Murphy-Hill [49] conducted interviews with eight blind software developers in which nearly all participants described having difficulty with user interface design because of the visual nature of concepts and tasks. The aspects of design that these software developers found challenging were interpreting interface layouts, inspecting their own design work, and writing CSS with efficiency and confidence [49]. When the authors inquired specifically about working with CSS, participants reported that they have \* PlantUML is a used to draw UML diagrams and can be found at [plantuml.com/](http://plantuml.com/) trouble with it compared to sighted developers [49]. More recently, Norman et al. found that nearly all of their blind participants created websites collaboratively or kept CSS stylesheets created by sighted developers on-hand as a reference [57].

**4.1.4.2 Blog Posts Related to CSS and Visual Design Issues.** The blogs that we reviewed did not present specific information about CSS or Visual Design issues. However, we found a description of specific strategies and tools used to approach design. In the blog post titled “*Software development 450 words per minute*,” Tuukka Ojala shares the following about visual design for the web: “I’ve found that libraries like Bootstrap are a godsend for people like me. Because of the grid system I can lay out a rough version of the user interface on my own. Despite this all the interface-related changes I’m doing are going through a pair of eyes before shipping to the customer. So, to sum up: I can do frontend development up to a point, at least while not touching the presentation layer too much” [60]. This blog entry points to the use of frameworks like Bootstrap [76] and sharing with sighted colleagues to review as potential strategies for doing front-end interface design.

In a separate blog about tooling for blind programmers, Parham Doustar describes interface creation as a barrier to getting freelance developer jobs. When considering freelance roles, the question, “*will I need to create a user interface on my own?*” inevitably arises, and according to Doustar, “*The answer is, at this point in time, a resounding ‘yes.’ That is why I’m not a freelancer*” [27].

<sup>1</sup>PlantUML is a used to draw UML diagrams and can be found at [plantuml.com/](http://plantuml.com/).

*4.1.4.3 Email List Threads Related to CSS and Visual Design Issues.* We found 13 distinct threads about the CSS and visual design issues. Issues range from specific CSS questions to more general conceptual issues with visual design.

Subscribers that had a specific CSS issue, wrote about differences in the way they wrote CSS and the way it was rendered (e.g., having overlapping elements when the CSS does not seem to be calling for elements to overlap) and wanting resources that would help them validate their CSS. Most of these related threads contained issues related to general conceptual issues. Subscribers sought advice on how to conceptualize CSS in an application or determine if something is visually appealing after the CSS is rendered within the browser. These subscribers sought CSS and accessible visual design tutorials and learning resources that are geared toward people that are blind.

*4.1.4.4 Observations Related to CSS and Visual Design Issues.* During the Oysters and Pearls technology camp and library workshops, we observed students having issues with visual design concepts and validation of CSS work [43]. In both workshops, we dedicated a day to an introduction to design and CSS [43, 45]. As part of this introduction, we covered a variety of design concepts (the box model, layout, typography, color theory, and color contrast accessibility), CSS syntax, and properties (font family, font size, alignment, padding, margins, etc.).

When learning how to apply CSS to their sites, students had challenges verifying their CSS outputs and required frequent visual interpretation to help confirm whether their CSS worked as intended. As they gained more experience, students tried more complex changes but still required validation from a sighted interpreter.

Identified issues with visual web design concepts are related to the method and tools used to teach them. More specifically, issues related to using tactile diagrams, RGB color creator tool, and the accessibility of a web-based color contrast analyzer. To teach concepts of serif vs. sans serif, font size, the box model, and alignment, we incorporated tactile and Braille diagrams. To teach alignment, we created a diagram of the visual interface of Google with and without styling. The diagram was meant to convey the concept of element spacing and alignment. Although many students found this helpful, some students in the Oysters and Pearls workshop struggled to grasp the meaning of design concepts. These students did not have experience with tactile diagrams, and their issues call into question both the quality of this type of diagram (created with a Braille embosser, no solid lines) and utility among students with no experience with tactile diagrams.

To teach color use in web design, we have taught color theory and described the color palette choices of branded sites. We asked students to experiment with using both foreground and background colors. Some students expressed that they had difficulty selecting colors. In the Oysters and Pearls Workshop, one student remarked, “I don’t like mixed colors. I want to use names.” Another student with some usable vision delighted in the activity and created a number of color combinations [43]. We did not gather information about how many of the students have some usable vision, color perception, are congenitally blind, or at one time lost their sight.

At the time of our 10-day library workshop, WebAIM’s popular color contrast analyzer tool was not screen-reader-accessible, and this made teaching students about color contrast difficult. The tool has since had accessibility improvements made to it, however, this prompted us to test the accessibility of accessibility validation tools more regularly [43, 45].

## 4.2 Orienting

*4.2.1 Literature Related to Orienting.* Several authors [5, 13, 71] have noted that when programming using a screen reader, there is a tendency to lose track of one’s location—in a code structure and in a development environment’s interface—which can slow the process of development and

cause frustration. Through their interviews and observations of blind programmers, Albusays et al. [5] found that participants felt out of control and “often reported disorientation” (p. 92). Notably, Smith et al.’s [70] research and development of an accessible code tree navigation strategy and IDE plug-in for blind programmers included a popular and effective “where am I?” key command. This command was used to describe which node a user is focused on in their code’s tree structure, and in their evaluation of the tool, this key was used most frequently [70]. This is due to an issue that Stefik et al. [73] have since referred to as the “where am I” problem. This problem occurs when a blind programmer loses their orientation in a program

**4.2.1 Email List Threads Related to Orienting.** We found several threads that contain information about issues related to orienting themselves in data structures such as JSON and complex interfaces and during tasks like comparing changes in code. For example, one author asked list subscribers about a new utility that they anticipate issues related to keeping track of where they are when comparing two files. Aside from software interface orientation challenges, some subscribers and authors mentioned using classes and naming conventions as a strategy to help with orienting.

**4.2.2 Observations Related to Orienting.** Although students in the workshops that we have previously taught created fairly simple programs, we observed the “where am I” problem in students that are blind and learning web development [43, 45]. This was most pronounced in comprehending nesting and parent-child relationships—instead of a glance at visually formatted code to glean this information, students often relied on memory to keep track of relative positioning in code structures or used notation to remember aspects of program structure.

### 4.3 Navigation

#### 4.3.1 Keyboard Navigation

**4.3.1.1 Literature Related to Keyboard Navigation.** Albusays and Ludi [3] observed that linear movement through code using arrow keys as screen readers users do is not efficient because of the layout and the structure of code. Albusays et al. [5] have observed that in addition to not being able to visually scan code, screen reader users cannot access many advanced IDE features for quickly moving through a large codebase, which forces them to navigate line-by-line or move past whole sections using “find/search” features. Returning to a specific line of code when reviewing code in other files is extremely important and also a barrier for screen reader users [5]. In their 2012 interviews with blind programmers, Mealin and McMurphy-Hill relay a participant’s sentiment that they felt at a disadvantage compared to their sighted teammates that can access content with a mouse, and that using basic “find” commands to navigate code is insufficient, because “finding keywords to jump to can be difficult” (p. 72) [49].

**4.3.1.2 Blog Posts Related Keyboard Navigation.** We did not identify blog posts that contained information about navigation issues. One, however, made specific mention about the use of keyboard navigation. Florian Beijers writes, “I barely ever use a mouse for anything. I tend to stick with hotkeys and the command line instead” [16].

**4.3.1.3 Email List Threads Related to Navigation.** We found 22 email threads specific to navigation. The majority of these threads are related to issues and strategies to address navigation granularity of code (reading code by line, by word, by character, or by changes). Several authors write seeking advice for general navigation strategies, and in response, list subscribers offer their strategies for navigating by line, word, or character depending on the level of detail required for a task. Some authors reported accessibility issues with navigating IDEs and seek advice about issues and

changes in navigation because of assistive tech and IDE feature compatibility (e.g., JAWS 2020 + Eclipse [30], VoiceOver + Emacspeak [63]). Another author reported issues with navigating and testing JavaScript components that their colleagues create.

*4.3.1.4 Observations of Issues Related to Navigation.* Our observations of screen reader navigation issues in IDEs, many text editors, and other development tools are congruent with the literature. The degree to which keyboard-navigation support and efficiency acts as a barrier greatly depends on the complexity of the interface, file structure, and screen reader user proficiency [43, 45].

#### 4.3.2 User Interface Navigation

*4.3.1.5 Literature Related to Navigating Interfaces.* The discoverability of existing and new features in a development environment's interface is challenging when using a screen reader, as features may be hidden within complex navigational hierarchies [61]. Similarly, Albusays and Ludi [3] found users described IDEs as overly complex, with some preferring to use a simplified text-editor to code and or keep notes about the structure of their code in IDEs.

*4.3.1.6 Observations of Issues Related to Navigating Interfaces.* Our workshop observations are limited in findings about complex IDEs in teaching web development to blind students that use screen readers [43, 45]. As the learning curve is so steep to understanding these complex interfaces, their use has not been appropriate for short form introductory workshops (one–two weeks). However, we have observed that browser-based web developer tool interfaces are very difficult to navigate, as they do not have a clear hierarchy of information or meaningful labelling of interface elements. For example, in using Chrome's Developer Tools, JAWS users must use arrow keys to bypass several elements to get to an "Elements" tab, then they need to change modes to navigate a tree view of the page DOM, and once they get to a target element, they must hit "enter" to interact with its properties, although these elements do not indicate these needed interactions by their properties or labels.

## 4.4 Lack of Support

*4.4.1 Literature Related to Lack of Support.* Several literature sources detail issues related to a lack of support. Specifically, authors describe issues that blind programmers face related to a lack of mentorship, issues related to the need to seek assistance, and cumulative effects on motivation.

Mentoring is a key factor in educational and career attainment, as well as community integration [77]. A striking issue faced by blind youth in science and technology is a lack of mentors knowledgeable about screen readers and accessible opportunities in STEM [18]. If instructors do not have a good understanding of how assistive technologies work or how to make materials accessible, then they will not incorporate consideration for their use into course design and as a result create inaccessible projects and materials [69]. Shinohara et al.'s 2019 survey of instructors revealed that the most critical barrier to teaching accessibility is the lack of instructor knowledge and resources about accessibility and the need to have course-specific accessibility resources [69]. In our own work on teaching accessibility in post-secondary education, we found that instructors who have invested in learning about accessibility were frustrated by the number of resources and their technical density [44].

Due to a lack of screen reader accessibility in the technologies used to code, screen reader users are often put in the position of needing to ask sighted individuals for help. In their survey of 69 blind programmers, Albusays et al. reported "seeking assistance" as a theme in responses or the need to seek help for certain programming tasks [3]. Respondents reported feeling embarrassed when asking for help from sighted teammates. Participants in Albusays et al.'s interviews

[5] expressed that they “felt uncomfortable disclosing their programming needs (e.g., navigation difficulties) and their disability status to colleagues or researchers.” These authors note that it is possible that a lack of disclosure may hinder others from fully understanding inaccessibility issues and work to improve the accessibility of IDEs [5]. This lack of support for independent programming can also decrease motivation [14].

According to Baker et al. [14], encountering inaccessible technology and learning materials can contribute to decreases in the motivation of blind students. Additionally, these authors write that, “When assignments are inaccessible, students may be less motivated to complete them to the best of their abilities and may therefore not learn the concepts in the assignments as well” (p. 2) [14]. These authors also describe a phenomenon in which blind people’s experiences with inaccessibility can hinder their exploration of new technology [14]. Each of these outcomes as a result of experienced inaccessibility has potential impact on student engagement and pursuit of programming education and careers.

**4.4.2 Blog Posts Related to Lack of Support.** In Parham Doustdar’s blog titled, “*The Tools of a Blind Programmer*,” Parham describes a lack of support from college instructors that drove him to become self-taught in computer science. Parham writes that professors would rather “get their job done and get home...I had to either rely on my own knowledge, or look for ways to get someone to read my books for me” [27]. Florian Beijers writes about a lack of support in job application and hiring processes in his blog post titled “How to Get a Developer Job When You’re Blind: Advice from a Blind Developer Who Works Alongside a Sighted Team” [17]. He describes the careful consideration he has given to the timing of mentioning his disability to the jobs he is applying for. He notes that although discrimination on the basis of ability is illegal, “it is at times rather easy to disguise as something else” [17]. He also describes experiences with inaccessible assessments and screening tools used in the hiring process—writing that in the past, he has not pushed for accessible processes, because he has found that the more he emphasizes his status as a blind person, “the more you make an issue of it, (and) the more importance it gains for the person considering your job application” [17].

**4.4.3 Observations of Issues Related to Lack of Support.** In each of the accessible web development workshops we taught, lessons and materials were created with accessibility in mind, and as part of efforts to foster a judgement-free learning environment, students were encouraged to ask questions [43, 45]. We have observed that some students are reluctant to ask for help. However, we have not observed this as a barrier to student learning, as gaining independence in programming requires independent debugging and problem-solving.

## 4.5 Use of Supportive Technology

**4.5.1 Literature Related to Use of Supportive Technology.** Screen readers are complex software controlled via a wide number of key combinations, requiring practice to become proficient. Armally and McMillan [10] note that screen reader users who have not yet mastered their screen readers are greatly challenged while programming. Regardless of proficiency, it can take screen reader users longer to perform tasks on a computer compared with sighted users. In a study of screen reader experiences [46], blind participants reported losing 30% of their time to frustrating experiences such as confusing page layout, accessibility issues with applications, poorly designed/unlabeled inputs, and screen reader software crashing. In the context of programming, Baker et al. [14] write that coding with a screen reader is more time-consuming when encountering new tasks. Albusays [4] reports that learning additional key commands and workarounds are needed to increase speed/efficiency and reduce errors while programming.



**4.5.2 Blog Posts Related to Use of Supportive Technology.** Tuuka Ojala writes about issues he has had with screen readers while designing web applications—not because of his own proficiency, but because of their design and use given the task of programming. He writes, “Unfortunately VoiceOver, the screen reader built into Mac OS, suffers from long release cycles and general neglect, and its navigation models aren’t really compatible with my particular way of working. There’s also a screen reader for the Gnome desktop and, while excellently maintained for such a minor user base, there are still rough edges that make it unsuitable for my daily use” [60].

**4.5.3 Email Threads Related to Use of Supportive Technology.** We found 29 unique threads that contained information about screen reader commands. These threads were often exchanges of information about commands to efficiently access information or perform tasks in development environments with more efficiency. Several others related to screen reader performance and compatibility issues (e.g., freezing or crashing screen readers, or compatibility in combination with development software).

**4.5.4 Observations Related to Use of Supportive Technology.** In our prior accessible web development workshops for students that are blind, we have found that screen reader proficiency can vary greatly in a class of beginners—even among people that have received the same training and certification (as was the case at the Oysters and Pearls workshop) [43]. On day one and throughout the Oysters and Pearls workshop, we observed different levels of student proficiency in browsing the web, accessing applications (including already open applications), and navigating file structures [43]. Although each student was adept in basic JAWS commands, just over half ( $n=7$ ) did not take advantage of existing key commands in the browser that make these basic tasks much easier [43].

## 4.6 Summary of CLR Findings

The CLR helped to identify a wide range of screen reader accessibility issues that exist in processes related to learning and the practice of computer programming. We used traditional literature review sources, as well as blog posts, email list threads, and observations from pilot workshops to better understand the breadth of issues and to help identify web development-specific accessibility issues and workarounds. The broader issue categories identified include presence of visual information without an accessible equivalent, orienting, navigation, and lack of support. Although we believe there is overlap between general programming and web development accessibility, web development-specific issues that were identified through the CLR can be summarized as follows: (1) inaccessible wireframing tools, (2) a lack of accessible explanations and demonstrations of CSS and Web Design concepts, (3) no way to independently verify computed CSS outputs, and (4) trouble navigating browser-based developer tool interfaces. We also identified some web development workarounds/strategies in the CLR, including the use of CSS frameworks like Bootstrap to more confidently create web layouts and collaborating with sighted developers for feedback on visual design. These specific issues are a helpful starting point in understanding what opportunities exist to improve the education and practice of web development with a screen reader. As a next step, we pursued confirmation and elaboration about their impact, as well as more web-specific issues, strategies, and workarounds from blind web developers.

## 5 INTERVIEW FINDINGS

With the aim of validating the categories and issues derived from the CLR, we interviewed 12 participants in semi-structured sessions that lasted approximately one hour. First, we collected participant demographic information including level of vision, level of vision at the time of learning web development, mode of web development education, and years of experience practicing web development. Then, we included open-ended questions about recurring accessibility issues related

Table 2. Interview Participant Table

Participants	Level of Vision	Primary Screen Reader	Output Type	Years of Experience	Web Dev. Education
P1	Blind	NVDA	Braille & speech	21	1, 2, 3, 4
P2	Blind	Voiceover	Speech synthesis	8	4
P3	Blind	JAWS	Speech synthesis	8	1, 4
P4	Blind	JAWS	Braille & speech	14	2, 4
P5	Blind	JAWS	Braille & speech	12	4
P6	Blind	JAWS	Braille & speech	2	1
P7	Blind	NVDA	Speech synthesis	6	1, 3
P8	BLV	JAWS	Braille & speech	15	1, 4
P9	Blind	JAWS	Speech synthesis	27	2, 4
P10	Blind	JAWS	Braille & speech	23	4
P11	Blind	NVDA	Braille & speech	5	4
P12	Blind	JAWS	Braille & speech	2	1, 3

Level of Vision: Blind and Low Vision=BLV. Web development education: College-level course =1, College-level degree=2, workshop=3, and self-taught=4. Accessibility Opportunities.

to learning and practicing web development and questions about the creation of supportive tools. Participants described issues with inaccessible learning materials, visual interface design, support from colleagues and employers, and use of supportive software, as well as the workarounds that they employ to address these issues. Participants also gave suggestions and ideas for the creation of accessible tools and technologies to assist in learning and practicing web development.

### 5.1 Participant Demographics

Table 2 summarizes interview participant information. All 12 participants identify as blind (1 participant identifies as both blind and low vision). The majority of participants use JAWS (n=8) as their primary screen reader while programming, some reported using NVDA (n=3) as their primary screen reader, and 1 participant reported using VoiceOver. Of note, 2 participants reported that they use JAWS as their primary and NVDA as an alternative while troubleshooting.

The majority of participants (n=9) reported using a combination of speech synthesis and Braille while coding, while others use just speech (n=4). Of note, those participants who use a combination of Braille and speech synthesis described the utility of each in a developer workflow—speech is good for a fast overview, whereas Braille helps them zero-in on the details of the code, as well as, the overall shape and structure of it. For example, P8 reported that Braille is good as “a visual spatial thing and for double checking what I just typed. (It gives) a zoomed-in, detailed view. Speech is quicker.” Further, P5 reported that speech synthesis provides a “simple analysis of what’s happening, or review of large quantities of code...Braille reading speed is slow, but good for debugging or close analysis of typographical errors.”

### 5.2 Validation and Expansion of CLR Issue Categories

Interviews with blind programmers provided insight into the validity of the five previously defined issue categories (visual information without an accessible equivalent, orienting/getting lost, navigation, lack of support, and knowledge and use of supportive technology) (See Table 1). We found that general categories derived from the CLR are sufficient in capturing web development-specific accessibility issues that participants described in open-ended responses. When asked if they had experienced recurring issues related to the five categories, participants responded “yes” to an average of four of the five issue categories (visual information n=12, orientation n=8,

Table 3. Interview Participant Experience with CLR Issues and Rating of Severity

CLR Issue Categories and Clarifying Statements	# of Participants That Reported Having the Issue	Median Severity Rating (1–3)
Visual information without an accessible equivalent	12	2
Orienting yourself in code or coding software (knowing where you are)	8	1
Navigating code and software (moving through sections of code, or within and between applications and files)	11	2
Lack of Support (help and support from teachers, colleagues, employers, peers)	10	2
Use of supportive tech (e.g., your screen reader, Braille outputs, or tactile diagrams)	7	1

navigation n=11, lack of support n=10, and supportive tech use=7). The majority of participants (n=8) agreed that the categories were sufficient in capturing any recurring accessibility issues that they had encountered. When asked if there were any additional types of issues they faced, participants did not provide examples that fell outside the main categories. In summary, we found overlap with web-specific issues identified in the CLR (lack of accessible explanations and demonstrations of CSS and Web Design concepts, inaccessible wireframing tools, no way to independently verify computed CSS outputs, and trouble navigating browser-based developer tool interfaces). However, interviews provided expanded explanations of these experiences in the context of learning and practicing web development.

5.3 Accessibility Issue Severity

To evaluate the impact of accessibility issues during interviews with blind programmers, we used a rating scale based on Dumas and Reddish’s [29] levels of severity in human-computer interaction. Assigning severity to the issues proved difficult for participants because of a variety of contextual factors, including the type of task and project, importance of task completion, availability of external help, and how a programmer relates to the term “blocker.” The general sentiment is that workarounds can be applied, and even severe issues do not necessarily prevent the completion of a task. Regardless of the issue, participants feel they can persist in learning and practicing web development.

5.4 Findings Related to Learning Web Development

Interview data suggests that blind web development students face recurring issues related to the accessibility of learning materials. Although the CLR helped to identify issues with wireframing, web design concepts, CSS verification, and developer tools—interviews helped to expand on and elaborate on these issues in the context of learning. Specifically, interview participants reported that they encountered inaccessible course-required software (e.g., prototyping and wireframing software, as well as IDEs and related software packages), video tutorials lacking description or supplementary code snippets, insufficient access to accessible print or Braille materials, PDFs, and

images of code without alt-text. P6 described his experience with inaccessible tutorial videos: “Instructor will say ‘let’s do this or that’ without describing what (it is)...and ‘click here or type this.’”

Interview participants reported that as students, they developed strategies to work around inaccessible learning materials, including self-advocating for usable materials from instructors and authors, seeking out alternative learning materials, and taking it upon themselves to learn web development independently. In seeking out alternative learning materials, participants reported requesting accessible materials from instructors, searching for alternatives themselves, and tapping blind programmer communities online.

Participants reported that in response to inaccessible learning materials, they had to self-advocate in higher-ed settings. For example, P10 told us, “I had to spend a lot of time explaining why I couldn’t do it.”

In addition to requesting accessible materials from instructors and their institutions, participants took it upon themselves to find supplementary information in accessible formats. They performed internet searches, read blogs and official documentation (e.g., W3C), identified and followed good content creators, and reached out to content creators and software developers about the accessibility of their work directly. P9 reported, “*I learned much better on my own than I did in any classes.*”

This strategy is reflected in student reporting about where and how they learned web development—the majority of students reported that they were self-taught in addition to acquiring a Computer Science degree with some web development or taking college-level web development courses. P7 broke down what he believed was the distribution of learning across course work and teaching himself: “*I’d say it was about 30% the course. And that was a lot of like wrestling with accessibility issues as well. And then 70% looking stuff up online and just finding other like resources...like books and courses to just supplement what the school is doing.*”

Participants also noted that they relied on online communities for sharing resources—two mentioned the now defunct [blindprogramming.net](#), as well as the programming-I and National Federation of the Blind computer science email lists.

## 5.5 Findings Related to Practicing Web Development

When asked whether they experienced recurring issues when practicing web development, several participants noted that there is so much overlap between learning and practicing web development that the distinction did not naturally occur to them. These participants feel that the learning process is ongoing, and the issues faced while in school or just starting out carry over into their continued and in some cases professional practice. Outside of issues with inaccessible learning materials, three issues emerged during interviews that related to the practice of web development, including interface design, support from colleagues and employers, and the accessibility of supportive technologies.

**5.5.1 Visual Interface Design.** In both learning and practicing web development, interview participants reported that they have grappled with issues related to web interface design. In line with CLR findings, the issues that participants describe relate to both conceptual, and applied aspects (default element styles and verifying CSS outputs). Here, we describe interview findings that elaborate on previously defined issues.

Conceptual issues that participants reported relate to knowing what good visual design is. For example, participants described issues with knowing what colors look good together, what typeface to select for different projects, and how to position elements to make a visually appealing interface layout. P4 stated, “I don’t really know what a good website looks like, and I don’t know

how to produce it.” P3 reported having trouble developing color schemes and palettes during the interface design process and told us, “you can find color wheels and contrast checkers, but it’s still hard to know what they look like.” Some interview participants reported having issues with reconciling the relationship and potential differences between the **document-object model (DOM)** and the way DOM elements are rendered visually. P2 reported that he was able to grasp the associated terms, but not how they work together to create a “nice looking website.”

Applied aspects of visual interface design include working with default HTML visual style and applying CSS. Participants reported issues with getting feedback about default element styles, the way element styles compute in relation to each other, and confirming that their CSS work has impacted visual styling, as intended. As P12 noted, when she was first learning HTML, she did not know whether headings automatically created place around them: “I just never learned you know, making headings...does it actually include a blank line...does it automatically, if you finish a heading put you on a new line?” Similarly, the default behavior of list elements was never communicated, and it was unclear which way they are oriented: “I can’t really even tell that those are going across the top versus going down” (P12).

Most participants (n=10) reported that they had issues getting feedback about their computed CSS and the impact it has on the relationship between elements. For example, P7 describes an issue with alignment of elements, “Elements on the front-end (are) hit or miss...you might have an element that is position(ed) over another fully or partially, and there’s no way to catch it...and the screen is looking bad.”

P8 reported that CSS output “is by far the biggest accessibility hurdle that you have to deal with, which is why I don’t deal with CSS.” Specifically, he described issues in knowing what is rendered on screen, “it’s hard to see (if) something is overlapping...If things are not like breaking when you minimize the screen or make (it) responsive...the closest you can get is check the CSS classes on an element, but that doesn’t always tell you if you’re actually being applied or something’s going wrong somewhere.” The issues related to visual design impact the work that participants do. Workarounds and strategies to address these issues include focusing on back-end development (avoidance of design), asking for help or collaborating with sighted programmers, using sighted interpreters, and using accessible developer tools.

Several (n=7) noted that visual design issues have informed their decisions not to engage in design processes and their options related to work. More specifically, these participants have worked around design by primarily focusing on honing their skills in backend web development and working at jobs that either have a clear division of labor between back and frontend, or that foster collaboration between sighted and blind programmers. P3 reported issues with getting some jobs, but that it is common for there to be a separation between front and back-end role responsibilities: “I didn’t get jobs, because I couldn’t do the full stack...but if you work in a shop that has two or more people, the division of labor happens naturally.” Speaking more directly to the division of labor, P3 described that “it just makes it a little bit easier to focus more on the back-end and not worry so much about visually what’s happening.”

Participants also described scenarios in which they collaborated with sighted programmers, asked for help from friends or colleagues, or used the visual interpreter services, AIRA [2]. P2 described a web development process with both divisions of primary duties and collaboration: “I worked with a team of people. So, I was building a lot of back-end stuff and they were kind of doing the front-end stuff, or we built the front-end stuff together.” Regarding visual interpretation, P3 remarked that they use the service AIRA, but it comes at a cost and this is not always a reliable solution, “It definitely does cost money and that also requires you get an agent that’s okay describing things (referring to web interfaces) which seems like it would be required for AIRA, but is not always the case.”



Some participants noted that they use CSS validation tools (P1, P5, P10) and color contrast analyzers (P3). Screen readers also offer advanced features that convey some positional and hierarchical information via object navigation in NVDA and JAWS. P3 described that object navigation provides information about whether, “your data is visible and showing...in kind of where it’s where you think it is... It gives you like a rectangle, and coordinates and size of that rectangle.” However, participants confirmed that this still requires they calculate relative position. As P3 noted, “you can at least make sure it’s not overlapping.” Some participants (n=3) use the NVDA **developer toolkit (DTK)** [19] to better understand visual interfaces and the output of their CSS. NVDA DTK allows users to move through elements in the DOM using arrow keys and provides some information about the visual interface (such as their color, font, height, position of top, right, bottom, and left edges, and distance between parent elements). However, “DTK cannot access information such as CSS rules, padding, borders, or z-index” [58].

**5.5.2 Support from Colleagues and Employers.** Some participants reported issues with a lack of support from colleagues. P8 described issues with colleagues not knowing how to work with him and needing to demonstrate how, “You have to show them ‘monkey see, monkey do.’ If you don’t have the confidence you won’t make it in this industry.” P12 described an ongoing collaborative project in which her partner sometimes “doesn’t understand why I need help.”

Several participants (P2, P4, P7, P8, P11) told us that they have mostly shied away from front-end web development and in their professional lives have gravitated to companies/organizations that structure work around a division of labor between front-end and back-end web development. For example, P3 described that “when you’re blind you are seen as a liability... you can’t get support to do (it) another way... they give (the work) to someone else, or pair you and that takes 2-hours.”

Participants described their efforts to self-advocate and educate individuals in the workplace. P8 told us about a number of strategies employed to self-advocate in the workplace, but ultimately that it was the responsibility of everyone. P8 said, “You have to have meetings, presentations... this is what I can do and what I can’t... this is how you do accessibility and this is how you break it... I can take a hit and educate people when I don’t have a million things going on... (it’s) not up to one person... it’s up to everyone.”

**5.5.3 Issues Related to Supportive Technologies.** Although participants reported that they were proficient with screen readers, some (P3, P4, and P6) described performance issues with their assistive tech that made web development tasks like debugging difficult. P4 described a situation in which it was unclear whether his markup was invalid or his JAWS software was freezing up, “Sometimes JAWS just freezes up... (it’s) hard to figure out what is the problem when JAWS is messing up... you don’t know which is which.” Some described the advantage of knowing advanced scripting techniques—as an intro to programming and a way to improve the functionality of their assistive technologies (P3, P9, P11). P6 told us that not knowing how to create scripts that enhance the functionality of screen readers is a disadvantage among blind web developers.

Some participants mentioned that browser-based web developer tools, such as the network tab and console, are not very accessible. For example, P4 told us that in the browser it is “hard to find the network tab in the developer tools.” Both P4 and P6 said that these browser tools are difficult to find, unless you are using NVDA and Firefox in combination.

## 5.6 Findings Related to New Tools and Technologies

As part of the interviews with blind programmers, we discussed the tools that would make web development more accessible, easier, and more enjoyable. The solutions and ideas that participants generated fall into three main categories: **Artificial intelligence (AI)** systems, tactile interfaces, and community supports.

Several participants (P1, P5, P9, P10, P12) noted the potential to leverage AI and specifically, machine learning and image recognition technologies to aid in the process of visual design and CSS output verification. Participants felt these tools could allow them to set visual design criteria and themes, then use criteria to generate color palettes or evaluate chosen color combinations and position of elements. P1 described their AI-powered idea as essentially the visual interpretation service AIRA, but (a) without a human agent and (b) with web development and design expertise. Similarly, P12 said he wanted a tool that “would automatically tell me how something actually looks in words.”

Some (P3, P6, P8) described tactile interface solutions to aid in learning and design verification by giving users a sense of the overall spatial layout of their website or even the structure of their code. The tactile representations that participants described ranged from static 2.5D–3D representations to dynamically updating tactile displays that change with code or content and state changes as a result of user interactions. P8 described a dynamic tactile interface that “shows you your content... as you are changing it.”

P2 and P4 described community support tools to establish a collaborative community of learning and practice. P2 suggested the development of an open educational resource for accessible materials on design and CSS concepts and component library. P4 suggested the formation of working groups that consist of blind and sighted individuals collaborating in real-time and sharing resources. He told us, “The biggest thing would be a community support. Not just for blind people, but some place we can go and feel comfortable disclosing... Most people get their start on their own... I want to do this (too), but how do I make it visually work?”

## 6 DISCUSSION

Prior to embarking on this work, we hypothesized that the accessibility of technologically mediated workflows and education are key factors in the lack of representation of blind web developers. We learned of accessibility issues in the technologies that support web development, the material and social factors that contribute to a perceived lack of support, as well as tooling combinations and strategies used to overcome accessibility issues.

From our findings, we are able to summarize web development accessibility issues into the following five categories: visual information without an accessible equivalent, orienting, navigating, lack of support, and use of supportive technology. Interviews supported the recurring issues in education and practice documented in programming accessibility literature. The web development-specific issues that we found are related to the inaccessibility of wireframing tools, a lack of accessible explanations and demonstrations of CSS and web design concepts, not being able to independently verify computed CSS outputs, and having trouble navigating browser-based developer tool interfaces. Elaborating on literature that documents a lack of support in educational contexts, we spoke with blind web developers that got little from formal education and had to supplement materials with those they sourced themselves. In examining web development-specific practice issues, we found they are consistently centered around the web design and the output of CSS.

When prompted to reflect on the severity of these issues, the vast majority of interview participants did not rate them as blockers, as they have persisted in learning and practicing web development. The following discussion is dedicated to the problem with issue severity ratings, accessibility of web development education, web development-specific practice issues, and ideas for tools to make web development more accessible.

Strategies and workarounds include avoidance of front-end development, the use of CSS, use of CSS frameworks, and collaborating with sighted developers for feedback on visual design.

Participants also provided us suggestions for the development of new accessible tools and technologies to support learning and practice, including AI systems, tactile interfaces, and community supports.

### 6.1 Interpreting Severity Ratings

Interview findings related to the issue categories identified in the CLR helped to validate these categories. Issue categories are relevant to our interview participants and their practice of web development. However, it became clear that gaining an understanding of the severity of their impact on the practice of web development was fraught with issues related to context and interpretation of the meaning of the word “blocker.”

Usability issue severity ratings are helpful in identifying and prioritizing design and engineering solutions [56]. In the context of this research, severity ratings are intended to identify the most serious usability and accessibility issues blind developers face when doing web development and point to areas of opportunity to develop solutions. Our interviews with blind web developers demonstrated that the issue categories are multifaceted and context dependent. In certain contexts, and related to specific project types, participants feel that the same issue could have very different impacts on their practice. For example, not being able to interpret the outputs of CSS is not considered a blocker if that task can be shared with a sighted coworker/collaborator. The levels of severity seemed to take on additional meaning with blind developers, because they did not want to be seen as less capable than sighted developers. Because participants have adapted and found ways to learn and practice web development, the word “blocker” read as too final and too severe. Similarly, participants questioned the meaning of the word—does an issue become a blocker when you have to ask for help? For many, the answer is no. This means they ask for help and move on with their practice. Participants described a variety of adaptive and creative strategies to address the issue categories.

We selected the Dumas and Reddish usability scale [29] because of its prevalence in usability research [33, 37, 47, 62]. However, our findings signal the need to re-evaluate the word “blocker” and use a different rating instrument in future. A multi-dimensional rating instrument that captures different aspects of usability would better inform our understanding of the issues we identified. For example, Nielsen describes usability issues as a combination of frequency, impact, and severity of problems [56]. We should not view usability problems as full stop barriers, but take into account larger contexts of use. Screen reader users are dynamic and adaptive problem solvers, and we need to inquire about their next steps and strategies when faced with usability problems.

### 6.2 Accessibility of Web Development Education

Based on our review of traditional literature sources, first-hand accounts in blog posts, and our interviews with blind web developers, college-level programs and courses in web development are failing blind students. The development and standardization of inaccessible course materials and tools alienate blind students and force them to take their education into their own hands. Interview participants attributed this to a lack of awareness and understanding about accessibility on the part of individual instructors and institutions. These findings point to a lack of accessibility best practices applied in course materials and curriculum. In particular, the instruction of Web design and CSS has not been adapted or made accessible to screen reader users.

To address this problem, a greater awareness and application of accessibility best practices is needed. Increased awareness among faculty would not only bode well for blind students, but for sighted graduates of these programs entering into the workforce and applying their own use of best practices in the technologies they help to develop. There is accessibility rigor needed in these programs, but especially in courses and units related to web design and CSS. It may be assumed

that this is an area fundamentally inaccessible or “off limits” to blind web developers, but we know that this is not the case. There is a need and desire among blind web developers to be able to understand design concepts and work with CSS, and therefore a need to make this curriculum more accessible.

### 6.3 Web Development-specific Practice Issues

In computer programming, there is a breadth of accessibility issues related to the visual elements in IDEs, whitespace, supportive graphs and diagrams, interface design, orientation, navigation, lack of support, and issues related to use of supportive software. It is through our CLR and Interviews that we were able to gain a better understanding of the accessibility issues unique to the practice of web development. These are primarily related to the inaccessibility of wireframing software, the inability to independently verify CSS outputs, and trouble navigating browser-based developer tool interfaces.

The separation of document structure from style is intentional [20], and benefits (1) web developers by offering flexibility when applying style, as well as (2) assistive tech users through the prioritization of semantics. However, this means that automatically obtaining useful information about the visual rendering of content can be a challenge as part of a blind web developer’s workflow. Participants reported a number of tools and strategies including the NVDA developer toolkit [19], browsers/assistive technology combinations and key commands that facilitate access to browser-based developer tools, web design frameworks like Bootstrap [76], working in jobs with a clear division of labor between frontend and backend web development, and collaborating with sighted designers and developers.

These findings point to a need to develop and refine tools that help with web design and CSS, and they teach strategies used to new developers. To address the issues with existing tools, the creators of supportive software for development (wireframing software, IDEs, browser-based developer tools) need to do better to prioritize the accessibility of their interfaces and workflows. With these issues in mind, ideas and opportunities to develop new tools are described below.

### 6.4 Participant Suggestions for Accessible Tools

The suggested solutions provided by participants (AI systems, tactile interfaces, and community supports) are all in some way related to addressing issues with inaccessible visual information, and in particular web interface design and CSS outputs. The community support ideas proposed also address the lack of support and meaningful collaboration in education and workplace settings.

The proposed AI solutions address a desire to leverage machine learning and image recognition technologies and to be more independent in the visual design process. Desired feature sets of these AI-powered systems include the ability to set thematic criteria and generate color palettes to evaluate chosen color combinations based on color theory and trends and to describe the position of elements and overall look of websites.

Like AI solutions, proposed tactile interface solutions reflect and address a desire to do visual design work more effectively and independently. They also address the issue of item-by-item linear presentation of information and can give screen reader users a sense of the overall layout of their website or even the structure of their code.

The community support ideas proposed include the development of a novel open educational resource for accessible materials on design and CSS concepts, as well as working groups that consist of blind and sighted individuals collaborating in real-time and sharing resources. These solutions

have the potential to address the gap in accessible learning materials and reflect a desire to create a collaborative community of practice.

## 7 LIMITATIONS AND FUTURE WORK

Our CLR was limited by sources that were skewed to non-web-focused programming accessibility. This work is also limited by our almost exclusive focus on blind web developers that have been practicing for multiple years (participant experience ranged between 2–27 years with an average of 12 years). In addition, because participant experience was high, this could have impacted the perception of the word barrier and potentially have influenced severity ratings. To better understand the current landscape and accessibility challenges in education, future research should focus on new learners. Although findings related to issue severity ratings helped us to better understand the complexity of the web development process with a screen reader, our instrument was limited. In future work, we would develop a variety of questions that tap each issue category but are more specific to different contexts and web development tasks. There is also opportunity to further develop and research educational initiatives that include findings about accessible tools and strategies for doing web development with a screen reader, and a focus on visual design and CSS components. Based on the desired tools discussed, there are several opportunities to develop and test workflow tools that leverage AI, tactile interfaces, and community support, as well. Long-term, we hope that this work stimulates thinking and development of (1) an inclusive web development curriculum and (2) tools to support the accessibility of web development. We believe that an accessible curriculum and supportive tools have the potential to stimulate confidence and increase workforce skills among blind students in the web development workshop.

## 8 CONCLUSION

In this article, we described the accessibility issues faced by blind screen reader users when learning and practicing web development, the workarounds/strategies they use, and ideas for the design of accessible curricula and tools to better support blind web developers. There is much work to be done to improve the accessibility of web-development education and opportunities to leverage existing technologies in the creation of tools to make the practice more accessible. Here, we have laid the groundwork by identifying the current state of education and practice of web development with a screen reader—including accessibility issues, strategies, and opportunities to develop new tools. Continued work to ensure that web development education and practice are accessible will help advance the democratization and accessibility of the web, as well as embolden new web developers.

## ACKNOWLEDGMENTS

We thank our participants, Drs Chris Hoadley, Kristie Koenig, and Kenneth Aigen (NYU), the NYU Ability Project, and the Andrew Heiskell Braille & Talking Book Library.

## REFERENCES

- [1] Gregory Abowd and Elizabeth Mynatt. 2000. Charting past, present, and future research in ubiquitous computing. *ACM Trans. Comput.-Hum. Interact.* 7, 1 (2000), 29–58. DOI: <https://doi.org/10.1145/344949.344988>
- [2] ARIA Tech Corp. 2020. ARIA. Retrieved from <https://aria.io/>.
- [3] 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*. ACM, 82–85.
- [4] Khaled Albusays. 2018. Exploring auditory cues to locate code errors and enhance navigation for developers who are blind. *SIGACCESS Access. Comput.* 120 (2018), 11–15. DOI: <http://doi.acm.org/10.1145/3178412.3178414>



- [5] 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.
- [6] Michael. Allen. 2003. Eight questions on teacher preparation: What does the research say? *A Summary of the Findings*. Retrieved from <https://eric.ed.gov/?id=ED479051>.
- [7] Apple. 2021. VoiceOver User Guide. Retrieved from [https://www.apple.com/voiceover/info/guide/\\_1121.html](https://www.apple.com/voiceover/info/guide/_1121.html).
- [8] Apple. 2021. TextEdit User Guide. Retrieved from <https://support.apple.com/guide/textedit/welcome/mac>.
- [9] Apple. 2021. Xcode. Retrieved from <https://developer.apple.com/xcode/>.
- [10] Ameer Armaly and Collin McMillan. 2016. An empirical study of blindness and program comprehension. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE'16)*. Association for Computing Machinery, New York, NY, 683–685. DOI: <https://doi.org/10.1145/2889160.2891041>
- [11] Cheryl Aasheim, Lixin Li, and Susan Williams. 2009. Knowledge and skill requirements for entry-level information technology workers: A comparison of industry and academia. *J. Inf. Syst. Educ.* 20, 3 (2009), 349–356. Retrieved from <https://aisel.aisnet.org/jise/vol20/iss3/10>.
- [12] Suzanne P. Balik, Sean P. Mealin, Matthias F. Stallmann, Robert D. Rodman, Michelle L. Glatz, and Veronica J. Sigler. 2014. Including blind people in computing through access to graphs. In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility (ASSETS'14)*. Association for Computing Machinery, New York, NY, 91–98. DOI: <https://doi.org/10.1145/2661334.2661364>
- [13] 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 (CHI'15)*. Association for Computing Machinery, New York, NY, 3043–3052. DOI: <https://doi.org/10.1145/2702123.2702589>
- [14] Catherine M. Baker, Cynthia L. Bennett, and Richard E. Ladner. 2019. Educational experiences of blind programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE'19)*. Association for Computing Machinery, New York, NY, 759–765. DOI: <https://doi.org/10.1145/3287324.3287410>
- [15] Martin Bauer. 2000. Classical content analysis: a review. In *Qualitative Researching with Text, Image and Sound*, Martin Bauer and George Gaskell (Eds.). SAGE Publications Ltd., 132–151. Retrieved from <https://www.doi.org/10.4135/9781849209731>.
- [16] Florian Beijers, A Vision of Coding, without Opening your Eyes. Retrieved from <https://medium.com/free-code-camp/looking-back-to-what-started-it-all-731ef5424aec>.
- [17] Florian Beijers. 2018. Stories from the trenches: What I've learned from working as a blind developer for a sighted dev team. Retrieved from <https://www.24a11y.com/2018/stories-from-the-trenches/>.
- [18] E. Bell and A. Silverman. 2018. The impact of attitudes and access to mentors on the interest in STEM for teens and adults who are blind. *J. Blind Innov. Res.* 8, 2 (2018) Retrieved from <https://nfb.org/images/nfb/publications/jbir/jbir18/jbir080201.html>.
- [19] Andy Borka. 2020. Developer toolkit. Retrieved from [https://addons.nvda-project.org/addons/developerToolkit.en.html#:~:text=Developer%20toolkit%20\(DTK\)%20is%20an,size%2C%20position%2C%20and%20characteristics](https://addons.nvda-project.org/addons/developerToolkit.en.html#:~:text=Developer%20toolkit%20(DTK)%20is%20an,size%2C%20position%2C%20and%20characteristics).
- [20] Bert Boss. 2016. CSS 20. Retrieved from <https://www.w3.org/Style/CSS20/history.html>.
- [21] Bureau of Labor Statistics, U.S. Department of Labor, Occupational Outlook Handbook, Computer Programmers. Retrieved from <https://www.bls.gov/ooh/computer-and-information-technology/computer-programmers.htm>.
- [22] Bureau of Labor Statistics, U.S. Department of Labor. 2020. Occupational Outlook Handbook, Web Developer. Retrieved from <https://www.bls.gov/ooh/computer-and-information-technology/web-developers.htm>.
- [23] Bureau of Labor Statistics. 2020. Occupational Outlook Handbook: Web Developers. Retrieved from <https://www.bls.gov/ooh/computer-and-information-technology/web-developers.htm#tab-8>.
- [24] Randy Connolly. 2019. Facing backwards while stumbling forwards: The future of teaching web development. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE'19)*. Association for Computing Machinery, New York, NY, 518–523. DOI: <https://doi.org/10.1145/3287324.3287433>
- [25] John Creswell. 2012. *Educational Research: Planning, Conducting and Evaluating Qualitative and Quantitative Research (4th ed.)*. Pearson Education Inc., Boston.
- [26] Diagram Center. 2015. Image Description Guidelines. Retrieved from <http://diagramcenter.org/table-of-contents-2.html>.
- [27] Parham Doustdar. 2016. The Tools of a Blind Programmer. Retrieved from <https://www.parhamdoustdar.com/2016/04/03/tools-of-blind-programmer/>.
- [28] Parham Doustdar. 2016. An Autobiography of a Blind Programmer. Retrieved from <https://www.parhamdoustdar.com/2016/03/27/autobiography-blind-programmer/>.
- [29] Joseph Dumas and J. Janice Redish. 1999. *A Practical Guide to Usability Testing*. Intellect Books.

- [30] Foundation Inc. 2020. Eclipse. Retrieved from <https://www.eclipse.org/ide/>.
- [31] W. Erickson, C. Lee, and S. von Schrader, S. 2017. *Disability Statistics from the American Community Survey (ACS)*. Cornell University Yang-Tan Institute (YTI), Ithaca, NY. Retrieved from Cornell University Disability Statistics website: [www.disabilitystatistics.org](http://www.disabilitystatistics.org).
- [32] A. C. Finkelstein, G. Kappel, and W. Retschitzegger. 2002. Ubiquitous web application development—A framework for understanding. Retrieved from <http://www.cs.ucl.ac.uk/staff/A.Finkelstein/papers/uwa.pdf>.
- [33] Freedom Scientific. 2021. JAWS. Retrieved from <https://www.freedomscientific.com/products/software/jaws/>.
- [34] Free Software Foundation. 2015. GNU Emacs. Retrieved from <https://www.gnu.org/software/emacs/Eclipse>
- [35] W. O. Galitz. 2007. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. John Wiley & Sons.
- [36] Ronald J. Glotzbach and Nishant Kothary. 2003. Usability & human behavior analysis through real-time performance data. In *ACM SIGGRAPH 2003 Web Graphics (SIGGRAPH'03)*. Association for Computing Machinery, New York, NY. DOI: <https://doi.org/10.1145/965333.965354>
- [37] Ezekiel Kimball, Rachel Friedensen, and Elton Silva. 2017. Engaging disability: Trajectories of involvement for college students with disabilities. In *Disability as Diversity in Higher Education: Policies and Practices to Enhance Student Success*. Retrieved from [https://scholarworks.umass.edu/cfssr\\_publishedwork/7](https://scholarworks.umass.edu/cfssr_publishedwork/7).
- [38] Google Inc. and Gallup Inc. 2016. Trends in the state of computer science in U.S. K–12 Schools. Retrieved from <https://services.google.com/fh/files/misc/trends-in-the-state-of-computer-science-report.pdf>.
- [39] Google Inc. & Gallup Inc. 2016. Diversity gaps in computer science: Exploring the underrepresentation of girls, blacks and Hispanics. Retrieved from <http://goo.gl/PG34aH>.
- [40] Don Ho. 2011. What is Notepad++. Retrieved from <https://notepad-plus-plus.org/>.
- [41] Shaun K. Kane and Jeffrey P. Bigham. 2014. Tracking @stemxcomet: teaching programming to blind students via 3D printing, crisis management, and Twitter. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE'14)*. Association for Computing Machinery, New York, NY, 247–252. DOI: <https://doi-org.proxy.library.nyu.edu/10.1145/2538862.2538975>.
- [42] Shaun K. Kane, Varsha Koushik, and Annika Muehlbradt. 2018. Bonk: Accessible programming for accessible audio games. In *Proceedings of the 17th ACM Conference on Interaction Design and Children (IDC'18)*. Association for Computing Machinery, New York, NY, 132–142. DOI: <https://doi-org.proxy.library.nyu.edu/10.1145/3202185.3202754>.
- [43] Claire Ferrari, Amy Hurst, and Scott Fitzgerald. 2019. Blind web development training at Oysters and Pearls Technology Camp in Uganda. In *Proceedings of the 16th Web For All 2019 Personalization Conference – Personalizing the Web (W4A'19)*. Association for Computing Machinery, New York, NY, 1–10. DOI: <https://doi.org/10.1145/3315002.3317562>.
- [44] Claire Ferrari, Devorah Kletenik, Kate Sonka, Deborah Sturm, and Amy Hurst. 2019. Evaluating instructor strategy and student learning through digital accessibility course enhancements. In *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS'19)*. Association for Computing Machinery, New York, NY, 377–388. DOI: <https://doi.org/10.1145/3308561.3353795>.
- [45] Claire Ferrari. 2019. Web development training for students that are blind. In *Proceedings of the 16th Web For All 2019 Personalization Conference – Personalizing the Web (W4A'19)*. Association for Computing Machinery, New York, NY, 1–2. DOI: <https://doi.org/10.1145/3315002.3332434>.
- [46] Jonathan Lazar, Aaron Allen, Jason Kleinman, and Chris Malarkey. 2007. What frustrates screen reader users on the web: A study of 100 blind users. *Int. J. Hum.-comput. Interact.* 22, 3 (2007), 247–269.
- [47] Jingyi Li, Son Kim, Joshua A. Miele, Maneesh Agrawala, and Sean Follmer. 2019. Editing spatial layouts through tactile templates for people with visual impairments. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI'19)*. Association for Computing Machinery, New York, NY, 1–11. DOI: <https://doi.org/10.1145/3290605.3300436>.
- [49] V. McKinney, K. Yoon, and F. M. Zahedi. 2002. The measurement of web-customer satisfaction: An expectation and disconfirmation approach. *Inf. Syst. Res.* 13, 3 (2002), 296–315.
- [48] Sean Mealin and E. Murphy-Hill. 2012. An exploratory study of blind software developers. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'12)*. Retrieved from [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=6344485](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6344485).
- [50] Microsoft Corporation. 2019. Microsoft Notepad. Retrieved from <https://www.microsoft.com/en-us/p/windows-notepad/9msmlrh6lf3?activetab=pivot:overviewtab>.
- [51] Microsoft Corporation. 2021. Visual Studio Code. Retrieved from <https://code.visualstudio.com/>.
- [52] Alana Morris, Anthony Onwuegbuzie, and Hannah Gerber. 2018. Using expert interviews within MODES in online and offline spaces to extend comprehensive literature review processes. *Qualitat. Rep.* 23, 8 (2018), 1777–1798. Retrieved from <https://nsuworks.nova.edu/tqr/vol23/iss8/1>.

- [53] Mozilla Developer Network. 2019. Retrieved from [developer.mozilla.org](https://developer.mozilla.org).
- [54] National Science Foundation. 2017. Women minorities, and persons with disabilities in science and engineering. Retrieved from <https://www.nsf.gov/statistics/2017/nsf17310/static/downloads/nsf17310-digest.pdf>.
- [55] National Science Foundation. 2017. Retrieved from <https://www.nsf.gov/statistics/2017/nsf17310/static/data/tab7-5.pdf>.
- [56] Jacob Nielson. 1994. Severity ratings for usability problems. Retrieved from <https://www.nngroup.com/articles/how-to-rate-the-severity-of-usability-problems/>.
- [57] Kirk Norman, Yevgeniy Arber, and Ravi Kuber. 2013. How accessible is the process of web interface design? In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS'13)*. Association for Computing Machinery, New York, NY, 1–2. DOI: <https://doi.org/10.1145/2513383.2513385>.
- [58] NV Access. 2021. NVDA. Retrieved from <https://www.nvaccess.org/>.
- [59] Anthony Onwuegbuzie and Rebecca Frels. 2016. *Seven Steps to a Comprehensive Literature Review: A Multimodal and Cultural Approach*. Sage.
- [60] Tuukka Ojala. 2017. Software development 450 words per minute. Retrieved from <https://www.vincit.fi/en/software-development-450-words-per-minute/>.
- [61] 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 CHI Conference on Human Factors in Computing Systems (CHI'18)*. Association for Computing Machinery, New York, NY, 1–11. DOI: <https://doi.org/10.1145/3173574.3174192>.
- [62] Python Software Foundation. 2021. IDLE. Retrieved from <https://docs.python.org/3/library/idle.html>.
- [63] T. V. Raman. 1996. Emacspeak—a speech interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'96)*. Association for Computing Machinery, New York, NY, 66–71. DOI: <https://doi.org/10.1145/238386.238405>.
- [64] Janice (Ginny) Redish, Randolph G. Bias, Robert Bailey, Rolf Molich, Joe Dumas, and Jared M. Spool. 2002. Usability in practice: Formative usability evaluations—Evolution and revolution. In *CHI'02 Extended Abstracts on Human Factors in Computing Systems (CHI EA'02)*. Association for Computing Machinery, New York, NY, 885–890. DOI: <https://doi.org/10.1145/506443.506647>.
- [65] Jaime Sánchez and Fernando Aguayo. 2005. Blind learners programming through audio. In *CHI'05 Extended Abstracts on Human Factors in Computing Systems (CHI EA'05)*. Association for Computing Machinery, New York, NY, 1769–1772. DOI: <https://doi.org/10.1145/1056808.1057018>.
- [66] Advait Sarkar. 2015. *The Impact of Syntax Colouring on Program Comprehension*. PPIG.
- [67] Emmanuel Schanzer, Sina Bahram, and Shriram Krishnamurthi. 2019. Accessible AST-based programming for visually impaired programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE'19)*. Association for Computing Machinery, New York, NY, 773–779. DOI: <https://doi.org/10.1145/3287324.3287499>.
- [68] Margrit Schreier. 2012. *Qualitative Content Analysis in Practice*. Sage, London.
- [69] Kristen Shinohara, Saba Kawas, Andrew J. Ko, and Richard E. Ladner. 2018. Who teaches accessibility? A survey of U.S. computing faculty. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE'18)*. Association for Computing Machinery, New York, NY, 197–202. DOI: <https://doi.org/10.1145/3159450.3159484>.
- [70] Ann C. Smith, Joan M. Francioni, and Sam D. Matzek. 2000. A Java programming tool for students with visual disabilities. In *Proceedings of the 4th International ACM Conference on Assistive Technologies (Assets'00)*. Association for Computing Machinery, New York, NY, 142–148. DOI: <https://doi.org/10.1145/354324.354356>.
- [71] Ann C. Smith, Justin S. Cook, Joan M. Francioni, Asif Hossain, Mohd Anwar, and M. Fayezer Rahman. 2003. Nonvisual tool for navigating hierarchical structures. In *Proceedings of the 6th International ACM SIGACCESS Conference on Computers and Accessibility (Assets'04)*. Association for Computing Machinery, New York, NY, 133–139. DOI: <https://doi.org/10.1145/1028630.1028654>.
- [72] Andreas Stefik, Christopher Hundhausen, and Robert Patterson. 2011. An empirical investigation into the design of auditory cues to enhance computer program comprehension. *Int. J. Hum.-comput. Stud.* 69, 12 (2011), 820–838. DOI: <https://doi.org/10.1016/j.ijhcs.2011.07.002>.
- [73] Andreas 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*. ACM, New York, NY, 571–576.
- [74] Andreas Stefik and Richard Ladner. 2017. The Quorum programming language (abstract only). In *Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, New York, NY, 641–641.
- [75] Anja Thieme, Cecily Morrison, Nicolas Villar, Martin Grayson, and Siân Lindley. 2017. Enabling collaboration in learning computer programing inclusive of children with vision impairments. In *Proceedings of the Conference on*

- Designing Interactive Systems (DIS'17)*. Association for Computing Machinery, New York, NY, 739–752. DOI: <https://doi.org/10.1145/3064663.3064689>.
- [76] Twitter. 2021. Bootstrap. Retrieved from <https://getbootstrap.com/docs/3.4/>.
  - [77] Teresa Whelley, Richard Radtke, Sheryl Burgstahler, and Theodore Christ. 2003. Mentors, advisors, role models and peer supporters: Career development relationships and individuals with disabilities. *Amer. Rehabil.* 27, 1 (2003), 42–48.
  - [78] Jaroslaw Wiazowski. 2013. Creating tactile images—Decision making process. In *Proceedings of the 7th International Convention on Rehabilitation Engineering and Assistive Technology (i-CREATE'13)*.
  - [79] Tessa Wright, Beth Harris, and Eric Sticklen. 2010. A best-evidence synthesis of research on orientation and mobility involving tactile maps and models. *J. Vi. Impair. Blind.* 104, 2 (2010), 95–106.
  - [80] W3C. 2016. Accessibility, Usability, and Inclusion. Retrieved from <https://www.w3.org/WAI/fundamentals/accessibility-usability-inclusion/>.
  - [81] W3C. 2016. HTML & CSS. Retrieved from <https://www.w3.org/standards/webdesign/htmlcss#whatcss>.
  - [82] W3C. 2019. HTML 5.2. Retrieved from <https://www.w3.org/TR/html52/introduction.html#a-quick-introduction-to-html>.
  - [83] W3C. 2020. Authoring Tool Accessibility Guidelines (ATAG) Overview. Retrieved from <https://www.w3.org/WAI/standards-guidelines/atag/>.
  - [84] W3C. 2020. Python Indentation. Retrieved from [https://www.w3schools.com/python/gloss\\_python\\_indentation.asp](https://www.w3schools.com/python/gloss_python_indentation.asp).
  - [85] Kim T. Zebehazy and Adam P. Wilton. 2014. Straight from the source: Perceptions of students with visual impairments about graphic use. *J. Vis. Impair. Blind.* 108, 4 (2014), 275–286.
  - [86] Zoom Video Communications, Inc. 2021. Video Conferencing, Web Conferencing, Webinars, Screen Sharing—Zoom. Retrieved from <https://zoom.us/>.

Received August 2020; revised January 2021; accepted March 2021