Accessibility of UI Frameworks and Libraries for Programmers with Visual Impairments

Maulishree Pandey, Sharvari Bondre, Sile O'Modhrain, Steve Oney
School of Information
University of Michigan
Ann Arbor, MI USA
{maupande,sbondre,sileo,soney}@umich.edu

***Abstract*—The availability of numerous UI components, the promise of accessibility, and cross-platform support have made UI frameworks (e.g., Flutter, Xamarin, React Native) and libraries (e.g., wxPython) quite popular among software developers. However, their widespread use also highlights the need to understand the experiences of programmers with visual impairments with them. We adopted a mixed-methods design comprising two studies to understand the accessibility and challenges of developing interfaces with UI frameworks and libraries. In Study 1, we analyzed 96 randomly-sampled archived threads of Program-L, a mailing list primarily comprising programmers with visual impairments. In Study 2, we interviewed 18 programmers with visual impairments to confirm the findings from Study 1 and gain a deeper understanding of their motivations and experiences in using UI frameworks. Our participants considered UI development essential to their programming responsibilities and sought to acquire relevant skills and expertise. However, accessibility barriers in programming tools and UI frameworks complicated the processes of writing UI code, debugging, testing, and collaborating with sighted colleagues. Our paper concludes with recommendations grounded in empirical findings to improve the accessibility of frameworks and libraries.**

***Index Terms*—accessibility, programming, user-interface development, programming tools, UI frameworks**

I. Introduction

UI frameworks and libraries have become increasingly popular for web and mobile development [1]. They help developers by offering native and custom UI components that enable the creation of complex interfaces [2]. Several frameworks and libraries, such as Flutter [3], React Native [4], and Cordova [5], also enable cross-platform development, allowing product teams to reach a wider number of platforms and end-users while developing in a single codebase. Many frameworks also claim to be accessible out-of-the-box, suggesting that the resulting UI would be accessible for people with disabilities. Given their widespread use and the advantages they offer, UI frameworks and libraries can have an outsized effect on the accessibility of UI programming and the web. They underscore the need to understand the accessibility of UI development for programmers with visual impairments as they use these UI frameworks and libraries. The consistent growth of UI developer job roles [6]–[8] also highlights the need to understand and improve the accessibility of the field to make it more inclusive.

Prior research in Human-Computer Interaction (HCI) and software engineering has studied the accessibility challenges in UI development [9], [10]. However, their focus was mainly on understanding the accessibility issues with IDEs and the need for sighted assistance in development. This paper takes a deep dive into the challenges in UI development and collaboration due to use of UI frameworks and libraries. Specifically, we ask the following research questions: (1) What are the motivations for programmers with visual impairments to use UI frameworks and libraries? (2) How do these frameworks and libraries shape their programming experiences and collaboration with sighted developers?

We report findings from a two-part mixed-methods study. First, we performed content analysis of 96 publicly archived mailing list posts on UI development; we followed this with 18 semi-structured interviews with programmers with visual impairments who have explored or used UI frameworks and libraries as part of coursework and professional responsibilities. Drawing on our analysis, we contribute the following:

- Evidence that accessibility challenges are difficult to isolate to programming tools or UI frameworks and libraries. We need to consider the interplay between programming tools, assistive technologies, operating systems, and UI frameworks to improve accessibility (see §IV-B).
- An understanding of how accessibility challenges hindered code writing, testing, and demonstrations for programmers with visual impairments. (see §IV-C)
- Design recommendations regarding documentation and supporting help-seeking for programmers with visual impairments. (see §V)

Our findings contribute to HCI, accessibility research, and software engineering research. They are especially important to people designing visual programming tools and languages.

II. Related Work

A. Accessibility of Programming

Initial research with programmers with visual impairments provided a high-level overview of their experiences, accessibility challenges, and practices [10]–[12]. Subsequent studies performed a deeper dive into these categories, which we explain below.

IDEs and text editors rely heavily on visual aids such as syntax highlighting and indentation to assist in source code navigation, organization, and visual search [13]–[15]. IDEs

This work is supported by a gift from Google.

also organize information visually into panels and windows, which are difficult for programmers with visual impairments to locate quickly relative to sighted programmers. These challenges are amplified by IDE documentation that rely on screenshots and do not list relevant keyboard shortcuts [16], [17]. For programmers with visual impairments, a common workaround is switching to plaintext editors [9], [10], [12] in conjunction with command-line interfaces (CLIs) for installation, debugging, and version control [18]. However, the latter present text in unstructured form without any markup, which poses navigation challenges for screen reader users [18].

Researchers and practitioners have created audio-based tools to address the challenges with navigation [14], [19]–[22], code comprehension [19], editing [23], and debugging [14], [24], [25]. Besides programming tools, empirical studies have also investigated collaborative programming activities. In prior work, we reported on how the practices associated with activities like pair-programming and code reviews have evolved to support sighted programmers [9]. Thus, programmers with visual impairments often have to drive the collaboration session when working with sighted teammates [9]. These accessibility challenges are further complicated by the programmers' social environment such as project management practices [9], [26], (un)availability of accommodations [17], and interpersonal relations with sighted colleagues [9], [27].

B. Accessibility of UI Development

Several solutions at the intersection of accessibility and UI development are targeted at sighted developers and designers to support them in building accessible interfaces. We highlight them for two reasons. First, their underlying interactions and interfaces remain visual and, therefore, of limited use to programmers with visual impairments. For example, Hansen *et al.* created an interactive tool to recommend sufficient color contrast in UI designs [28]. While developers and designers with visual impairments would find utility in such a tool [9], its reliance on visual elements limits its generalizability to the group. Second, these studies provide valuable insights into the limitations of UI frameworks as they are used by sighted developers [29]. Sighted developers have found that Xamarin [30] and React Native [4] do not expose all the accessibility APIs, making it difficult to create fully accessible mobile applications [31]. Similarly, the web frameworks, Angular, Vue, and React, do not notify sighted developers about accessibility violations [32]. We confirm and build on these findings by bringing in the perspective of programmers with visual impairments.

Empirical studies offer insights into the challenges of creating webpages using HTML, CSS, and JavaScript [33]–[35]. Programmers with visual impairments have shared that they feel less confident about CSS modifications [10], [36] and seek sighted assistance to verify the layout and CSS edits [9], [33], [35]. Furthermore, artifacts such as wireframes and design specifications lack text descriptions or are inaccessible with screen readers, requiring clarification about colors, pixel values, etc., with sighted team members [9]. Prior work has also revealed that people with visual impairments find it easier to understand the spatial layout on touchscreens compared to computers [37]. During development, they can use the screen reader gestures to verify the size and position of UI elements with relatively higher independence [9]. On the other hand, most layout editors within IDEs interface poorly with screen readers [9]. They do not offer pixel positions, relative locations, and dimension information. To address these challenges, Borka developed the Developer Toolkit, an NVDA addon, that informs developers of location and dimensions of UI elements [38]. Researchers have also developed multimodal systems to convey spatial layout of webpages in non-visual formats — using tactile print-outs to represent the HTML [33]; organizing tactile beads on a sensing board to create new layouts [39]; using gestures to edit HTML/CSS on tablets with VoiceOver feedback [40]. Potluri *et al.* have discussed the potential of using AI to support color selection, iconography, layout design, etc [41]; their representation in tactile forms (e.g., color wheel diagrams, braille font charts, etc) have shown promise in teaching web development [36].

The existing literature in HCI, accessibility research, and software engineering has yet to examine the advantages and challenges the UI frameworks and libraries present to programmers with visual impairments. We try to provide that understanding through our research.

III. Methods

We adopted a mixed-methods approach and conducted two studies to understand the UI development experiences of programmers with visual impairments.

A. Study 1: Analyzing Archived Posts on UI Development

We scraped the archived posts dated from January 2018 to December 2021 from the program-l mailing list (program-l@freelists.org)—an active and free discussion group for programmers with visual impairments to ask questions and share resources. The archive for the mailing list is publicly available [42] and dates back to November, 2004. Our choice of the four-year time period was guided by the goal to capture conversations before the start of the COVID-19 pandemic and to target the most recent technologies.

The posts and replies are archived as separate web pages in chronological order. We scraped a total of 11,915 web pages (average 248.23 emails per month). We combined the original posts and their replies into threads and saved them as text files for analysis, resulting in 2,607 files.

The first author went through the subject lines to identify threads most likely related to UI development. We identified a total of 726 threads on the topic. Next, we randomly sampled 150 threads over three rounds (50 per round). The approach allowed us to perform qualitative analysis in intervals and reach thematic saturation [43]. When coding, if the content of the thread seemed unrelated to GUI development, we removed it from our analysis. In total, we analyzed 96 threads; the breakdown after eliminating unrelated threads was 33, 31, 32 threads in round 1, round 2, and round 3 respectively. The

TABLE I
DEMOGRAPHIC CHARACTERISTICS OF THE PARTICIPANTS.

ID	Age	Country	Programming Education	Current Job Title	Programming Experience	Programming Languages and Frameworks
P1	23	USA	Bachelor of Computer Science	Software Developer	4 years	Java, C#
P2	26	USA	Bachelor of Computer Science	Software Developer	3 years	Java, PHP, Node.js
P3	30	US	Bachelor of Computer Science	Full Stack Developer	6 years	Java, TypeScript
P4	39	UK	Master's in Machine Learning	Computer Science Teacher	9-10 years	Python, Java, Swift
P5	30	Switzerland	PhD in Computer Science	Software Engineer	10 years	C++, Python, C
P6	22	USA	Bachelor of Computer Science	Incoming Software Engineer	7-8 years	C#, C++, Python, JavaScript
P7	19	USA	Self-Taught	Accessibility Specialist	5 years	Python (wxGlade)
P8	27	India	Master's in Computer Applications	Accessibility SME & Tech Lead	7-10 years	Java, C# (Xamarin), C, C++
P9	46	Sweden	Self-Taught	Software Engineer	30 years	C# (WinForms), .NET
P10	23	India	Bachelor of Computer Engineering	Software Engineer	3 years	Python (PyQT), C#
P11	27	Bahrain	Bachelor of Computer Science	Applying to Programming Jobs	6 years	Python (wxPython), Java, Angular
P12	28	India	Bachelor of Technology in Electronics	Accessibility Consultant	6-7 years	Java, React, Swift, Kotlin
P13	22	Pakistan	Self-Taught	Student	2 years	C# (WinForms), HTML/CSS
P14	35	Hong Kong	Self-Taught	Research Assistant	10 years	HTML, Python (PyQT, Flask)
P15	35	Iran	Self-Taught	Freelance Software Engineer	13-14 years	JavaScript, Java, C# (Xamarin)
P16	26	Iran	Self-Taught	Junior Back-end Java Developer	3 years	Java, HTML, CSS
P17	24	Egypt	Self-Taught	Student (Preparing for Master's)	1-2 years	Python (PySimpleGUI), HTML/CSS
P18	25	India	Self-Taught	DevOps Engineer	3 years	ReactJS, Python (wxPython), Flutter

final list of threads was organized alphabetically and indexed to quote from in the present paper. We describe our analysis of the email threads in section III-C.

B. Study 2: Semi-Structured Interviews

Our thematic analysis gave us a breadth of understanding about the accessibility challenges in UI development when using frameworks and libraries. To gain a more in-depth understanding of their use and impact on collaboration, we decided to conduct interviews. The first author conducted semi-structured interviews with 18 programmers with visual impairments. Eligible participants had to be at least 18 years old and either explored or possess experience in using UI frameworks and libraries to build web or mobile applications. We recruited participants through snowball sampling (n = 2), posting on the program-l mailing list (n = 13), and posting on the r/blind community on Reddit (n = 3).

Participants were 19 to 46 years old (median age 26.5; average age 28.16). Only one participant (P17) identified as female; the remaining participants identified as male. P2 and P5 identified as programmers with low-vision and used screen magnifiers and zooming respectively. P3, P4, P9, and P12 shared having retinitis pigmentosa; P14 shared having macular degeneration. The onset of visual impairment differed among these participants. The remaining participants reported having little to no usable vision since birth. Besides P2 and P5, each participant used a combination of screen readers. JAWS [44] and NVDA [45] were the most popular screen readers among our participants. P3, P4, and P9 reported using VoiceOver [46] along with other screen readers. Table I summarizes participants' demographics and programming experience.

Our interviews lasted between 40 and 75 minutes and were conducted remotely over participants' preferred video conferencing platform. Participants verbally consented to audio recording the interviews. We asked participants about the frameworks they have explored or currently use, challenges they encounter during programming, their experience with documentation and tutorials, and their motivations for learning UI development. The interviews concluded with a short questionnaire about participants' demographics and programming background (Table I). We compensated each participant with a $30 USD gift card or its equivalent in local currency. Each participant interview was transcribed in English for analysis, described in the next section.

C. *Analysis*

Two members of the research team analyzed the first round of email threads using open-coding to identify initial themes, followed by inductive coding [47] for all of the threads. We developed a total of 41 codes, which were clustered into 7 higher level themes. The members also wrote analytical memos [47] during the coding process to analyze emerging themes and identify gaps in the data. We performed weekly reviews as a research team to discuss the findings and prepare questions that would be relevant to follow-up on in interviews.

We were unable to transcribe the first two interviews due to the poor quality of the audio recordings. We relied on our notes for those interviews. The remaining interview transcripts were first open-coded by two team members, followed by organizing the data into codes from Study 1 and creation of 1 additional high-level theme. After coding the transcripts, we did a final reorganization of the codes, resulting in six high-level themes, which included codes on challenges in UI development, lack of documentation, considerations behind choosing UI frameworks, etc.

IV. Findings

We present the key results from our analysis, focusing on how the (in)accessibility of UI frameworks and libraries shapes the programming processes and experiences of developers with visual impairments. Quotes are slightly edited for clarity. Quotes from archived mailing list threads (study 1) contain thread IDs (T#) and quotes from interviews (study 2) include participant IDs (P#).

A. ***Motivations for Using UI Frameworks and Libraries***

We found that programmers with visual impairments were motivated by different reasons to pursue UI development. Employment opportunities were a common reason among interview participants (n = 9) to learn UI development. Participants shared that being familiar with UI development improved their chances of being hired, even though their preferred job roles were back-end development. Other participants were intrinsically motivated; they (n = 3) shared that they had always been interested in UI development. P7 shared that he had always considered himself *"as more of a designer"*. Furthermore, learning UI development established conversational fluency with front-end developers and designers:

> P16: *"Sometimes I should check something with the front end guys. And it's crucial for me to know how web development works in a big picture. [...] how HTTP works, what are HTTP methods - GET, POST, how RESTful API works and so on."*

Many interview participants (n = 5) explicitly stated that they preferred using UI frameworks and libraries rather than writing code from scratch. UI frameworks offered a *relatively* independent way of creating the front-end. For example, when a member inquired about the possibility of developing *"good-looking web interfaces as a blind person"*, members on the mailing list strongly recommended frameworks such as DOJO [48] and Bootstrap [49]:

> T15: *"It [Bootstrap] is highly idiomatic and easily calculatable with ratio of columns and rows. Its built-in components are already good-looking enough, and you can easily customise with skins or simple CSS touches. Its developers also consider and even dictate best practices for accessibility."*

Frameworks and libraries also provided helpful visual scaffolding for developers and the designers they worked with. Designers had to develop visuals using the existing components instead of requesting developers to create custom components. As P3 explained, he could directly *"use the SDK"* in his code:

> P3: *the mockups were based on the components that already exists [...] whoever builds like the visual design part has to align with the standards of the SDK. It's not like they are inventing a UI.*

Frameworks and libraries also helped differentiate between UI design and development responsibilities. Participants shared that they did not have to *"worry about colors, contrast, stuff like that"* (P3). The visual details were either considered by the framework designers or were specified by the in-house design team. Thus, programmers with visual impairments could focus on the functionality of the UI:

> T15: *my boss brought our company's graphic designer into my department to help. He has taken my super-simple UI and turned it into something my company could show off. So there definitely is a certain art to it and vision is not the issue.*

Developers also spoke of their unique expertise in making UIs accessible for end-users with visual impairments. They sought assistance to make the UI components usable for sighted end-users and coached sighted developers on how to make the components accessible for screen reader users :

> T3: *as screen reader users, we are the experts [...] You always want someone with a pair of eye-balls to check out the colors. You also want someone that has a decoration talent to help identify where each color combination should go on the site*

Thus, many interview participants considered the job roles to be interdependent. According to them, each team member, with their skills and competence with assistive technologies, improved the accessibility and user experience of the interface.

When choosing which framework to learn, we noted a strong preference for frameworks that were popular. For instance, P8 shared that he was *"currently working in winUI because it is the hottest technology"*. Similarly, when advising a developer about selecting a framework among Angular, Vue, and React, the mailing list members recommended the lattermost since it *"still has the lead in terms of jobs"* (T82).

In summary, the use of frameworks and libraries afforded higher levels of independence, delineated between design and development responsibilities, enabled creation of good looking UIs, and improved employment opportunities for programmers with visual impairments. However, as we explain next, the limited accessibility of front-end frameworks and programming tools could hinder developers' collaboration and performance in the workplace.

B. Accessibility Challenges

Accessibility barriers played a decisive role in our participants' programming experiences. We first discuss their experiences with software critical to UI development, such as IDEs, emulators, and browser developer tools, followed by the challenges with UI frameworks and libraries.

1) ***Inaccessible Programming Tools****:* Consistent with prior work, we confirmed that GUI builders in most IDEs were not accessible with screen readers [9]. Sighted developers can use them to *drag and drop* the UI components and create the layout quickly. Since mouse interactions are not accessible to people with visual impairments, they often had to *"hand write everything for the UI"* which took *"a lot of time"* (P15). In section IV-C1, we describe how the different approaches to UI design affected collaboration between programmers with visual impairments and their sighted colleagues.

We recorded instances of mailing list members searching for accessible GUI builders (n = 3) so that they do not have to type the entire UI code. For instance, one thread enquired about accessible interface builders for C++. While the discussion led to the discovery of an accessible extension, it only offered a limited set of widgets:

> T38: *The name of this extension is Nitisa. This is a Visual Studio extension and can be designed for C. But it doesn't use Visual studio as a Toolbox. I would love to have a GUI designer that can use Visual studio Toolbox.*

Developers logged issues on GitHub and directly reached out to development teams to improve the accessibility of GUI builders and IDEs. Some product teams acknowledged accessibility issues and proposed fixes, which developers viewed positively. However, the improvements could also be slow to come through, with the updates sometimes removed from the mainstream tool:

> T73: *you would want to have Git installed, so you can point to the accessibility branch and run WXGlade once you have switched to that branch.*

The quote above is from a thread where it was pointed out that to use wxGlade, the GUI builder for wxPython, one needed to check out the *accessibility* branch instead of working off of the main branch. In a similar vein, participants shared that the updates to the IDEs could negatively affect the accessibility features and they had to either revert to an older version or await future releases.

Emulators provided by major IDEs like Android Studio were often inaccessible with screen readers. Developers had to run the application on their personal devices, which was time-consuming in the initial stages of the project. For freelancers, the lack of accessible emulators limited the number of devices they could test their application on. They had to either ask friends and family for their devices or hope that the UI they had developed would be displayed correctly on devices with varying screen resolutions and dimensions:

> P15: *I have to test it on different people's phone if they allow me to get the result. So it takes a lot of time. It really takes a lot of time!*

The problem was amplified for macOS and iOS. Apple's policy requires testing the app with their device. However, the exclusive availability of JAWS and NVDA on Windows and the poor accessibility of IDEs with VoiceOver, Apple's screen reader, made Windows the preferred programming environment for the developers in our studies. Without an accessible emulator and availability of a device, they could not develop UIs for Apple devices:

> T92: *you need a mac in order to test your app on an ios device. This is quite frustrating [...] I could install a mac virtual machine, however then I have to deal with learning to use the OS and navigating my way around xcode. Has anyone found a way to develop apps for iOS that is accessible? Or is there an accessible iOS emulator that is good?*

The participants from Iran (P15 and P16) shared that they had to contend with an extra layer of inaccessibility. IDEs offered by Google and Apple, including the devices by the latter, were not usable in Iran because of the government sanctions imposed on the country.

Besides IDEs and GUI builders, accessibility issues with browser developer tools were mentioned most frequently in the email threads (n = 6). Poor accessibility would hinder developers from navigating and searching the DOM. To work around this, they had to either try different browser and screen reader combinations or get sighted assistance:

> T79: *I couldn't track down/find [graphical elements] in the original examples initially, but my sighted brother managed to find them sort of hidden in the DOM for me*

The different combinations of programming tools, browsers, and screen readers led to a long tail of individualized accessibility issues. The differences in programming environments made it difficult to provide instrumental help to address the accessibility problems. For example, one email thread shared tips and tricks to save Google Chrome's console logs due to poor accessibility of its Developer Tools. However, differences in keyboard layouts and browser versions made it difficult for members to apply the solutions effectively:

> T23: *"I have the option to save the logs on google chrome. I think you are not running latest beta version of google chrome. Perhaps you try updating google chrome on your windows machine"*

*2) **Inaccessible UI Components**:* To be able to use a framework efficiently, the UI components must be accessible. To assess a framework's accessibility, participants shared that they often browsed the official documentation to find a mention of accessibility. This served as a hint for whether the development team had given any thought to accessibility. However, positive search results did not necessarily guarantee accessible UI components:

> P12: *The page of that component library claimed itself to be 'out of the box accessible' and they [participant's team] blindly imported everything the modals, the accordions, the buttons, each and everything [...] And we found very disappointing results [...] the buttons looked like buttons but were announced like menus to the screen reader*

We noted a general consensus that no framework or library was completely accessible. Thus, the decision to use a framework or a library was based on competing factors such as availability of documentation, cross-platform support, and effort needed to improve the accessibility:

> T72: *Try XOJO. It is a Windows based cross-plattform development tool using Basic language to develop apps for both Windows and iOS/Mac. It is not fully accessible but I can live with them.*

The mailing list members shared components they had made accessible and compliant through trial and error so that others could refer to them. In Section IV-C2, we describe the impact of inaccessible components on programming processes such as debugging and testing.

*3) **Inaccessible Layout Managers**:* Layout managers—tools that automatically group and arrange UI components according to developer-specified constraints—considerably improved the UI development experience. Our participants shared that they often relied on these when tasked with creating the UI and preferred libraries such as PyQT and wxPython that offered a *relative* way of organizing the UI controls:

> P10: *If you don't do a layout manager, you need to explicitly say everything. [...] You need to pass the coordinates [...] But, again, that doesn't make any sense to me because I don't know which coordinate to give because I am not seeing it.*

Layout managers also enabled the participants to edit UIs more easily since the dimensions were updated automatically when changes were introduced. As a result, participants felt more confident and competent working with these frameworks:

> P7: *[With wxGlade] I can have a reasonable degree of confidence that those controls are where I say they are*

However, not all frameworks and libraries offer layout managers. For instance, P7 shared that he has not *"found a similar thing"* that allows him to develop front-end for web applications with the *"same convenience"* as wxGlade does for desktop applications. Furthermore, layout managers can also be offered through IDEs or third party tools, which may not be accessible.

C. Impact on Programming Processes and Performance

The accessibility challenges mentioned above affected the workflows, collaboration, and performance of programmers with visual impairments, which we describe below.

*1) **Writing UI Code**:* As mentioned earlier, programmers with visual impairments either try to find accessible GUI builders—which are rare—or manually code the UI. Developers expressed concerns about the number of lines required to create UI components when typing the code in comparison to using GUI builders:

> T6: *If you design items [...] using the XML editor in Android Studio, as the graphical way of [...] is still inaccessible, you define every component in 4 lines if we don't count the wrappers. With Swing, you have a few lines more: you have to create a container too and add both to the frame which you created previously.*

Inaccessible GUI builders could also complicate collaboration with sighted colleagues. It prevented them from creating *"clean looking resource file"* (T30) that their sighted colleagues could review quickly. They also felt that the additional lines of code made readability and navigation difficult with screen readers, especially when editing the UI. They had to redo the calculations if dimensions or positions were changed. In contrast, the resource file containing the UI code was automatically adjusted for sighted developers as they manipulated the measures with the GUI builder. Similarly, identifying and updating the location of visual parameters was difficult given the nested nature of the source code:

> T2: *I just found myself overwhelmed by the number of options and layouts with very little idea how to make sure they do what I want. I lose track once I am about two levels deep into the user interface element structure.*

Some GUI builders also produce incomprehensible code. One GUI builder, for example, produced generic variable names for UI controls. It was difficult for the developers to map these names to UI controls' position and functionality:

> T56: *Putting 2 buttons on a WPF designer surface, then tabbing around, forces the screen reader to say 'grid', 'button', 'button', 'window'. What button is what one?*

P9 shared that he had instructed his team to provide meaningful names to the UI controls to make collaboration on UI code easier. After laying out the controls, his sighted colleagues would edit the variable names in the resource file. Participants also shared that sighted developers did not realize that if they dropped the elements in random order, it did not change the UI visually but disorganized the accessibility tree. Accessibility trees are based on the DOM tree and expose a semantic version of the UI to screen readers via platform-specific APIs [50]. If the UI elements are not in the correct order or misrepresented, then it affects screen reader navigation and interaction. For programmers with visual impairments, this hindered their ability to debug and test. P9 mentioned that he had told his sighted colleagues to be mindful of the *"tab order"* when using the GUI builder:

> P9: *if we have the correct tab order, you start in the upper left corner and you go through the controls and the labels and grids. But if the tab order is out of order, you can jump between [imitates screen reader]. That makes it very hard to manage.*

*2) **Debugging and Testing**:* A major consequence of poor accessibility was the difficulty in debugging and testing one's output. Furthermore, the broken accessibility of certain components prevented developers from reproducing the bugs of their sighted colleagues. For P16, it hindered his collaboration with front-end developers:

> P16: *When I want to reproduce a bug [...] some parts of this web UI is not very accessible [...] For example, when I press enter in a web element, it does not work [...] I found out that if I press* `insert + space` *to go from a browse mode to focus mode in my screen reader [...] it will work.*

As mentioned earlier, even the software and frameworks that enjoyed the consensus of being *largely accessible*, presented some issues. The scarcity of documentation on accessibility of UI components meant that programmers with visual impairments often had to just *"dive in and try"* (T8) to assess the severity of issues across frameworks and libraries:

> P14: *I produced a Qt 5 interface that I cannot interact with [...] after a long, long time of research, I learned about some basic things that can adjust the code to make it accessible to the screen reader.*

Given the general unavailability of documentation on the accessibility of UI components, mailing list members reached out to one another for documentation and resources and gathered reviews on a framework's accessibility. They would mention the framework they were using and the specifics of their programming environment. Others on the list would share their experiences with the framework in their specific environments and even offer to test the source code or the specific UI components at their end:

> T28: *wx uses native controls, so I don't see why they shouldn't work on the mac or Linux. When I get home I'll run one of my in progress wx apps on my iMac and I can give you definitive* information.

Sharing debugging and accessibility experiences allowed the developers to work around the lack of documentation and identify the platform and screen reader combinations on which their UIs would work. However, this kind of sharing and support was not possible for programmers working on proprietary and private codebases.

The time and effort needed to test and fix the accessibility of UI components could range from adding ARIA attributes [51] to the markup to using scripting tools like Web Accessibilizer [52] for fixing issues at scale to even writing code that uses separate UI components for different platforms to offer a consistent user experience with screen readers:

> P6: *what I ultimately had to do was add logic into the program that if you're running it on windows, it uses one version of the tree control and if you're running it on anything else, it uses a different version*

*3) **Social and Personal Implications**:* As prior work has found, programmers with visual impairments were often tasked with educating their colleagues about accessibility issues and advocating for accessible solutions [9]. Participants were often also tasked with explaining accessibility issues to their sighted colleagues. For example, P6 had to demonstrate the trade-offs of a cross-platform framework across Linux, Mac, and Windows and explain how UI components behaved differently with various screen readers:

> P6: *I would show him here's how it sounds on windows, here's how it sounds on Mac, here's how it sounds on Linux. Here's the information that one of the tree controls is giving you in one environment versus the other, and this is why this is a problem*

Participants also described having to advocate for accessible solutions within their team. Often the decision to use a particular framework or programming tool was taken by the team collectively. If they chose things with poor accessibility, it could severely impact the productivity of programmers with visual impairments. For instance, P8 had to convince his team to use Xamarin and Visual Studio for the Android application they were building; the poor accessibility of Android Studio would keep him from giving his *"full efforts"*:

> P8: *I explained to them that if we develop using Xamarin, we will be able to do it in less time.*

The decision to switch to Xamarin came with trade-offs for P8. He said Xamarin did not provide access to all the Android APIs. He had to rewrite code to wrap some of the libraries on his own. We recorded concerns about the poor support for native libraries, including accessibility APIs, on the mailing list threads (n = 5) as well.

Inaccessible UIs also prevented our participants (n = 2) from using the UI and experiencing the user workflows independently. For instance, P10 had joined as a back-end developer on an existing project. He could not *"go back and make the"* UI accessible in one go. Unable to use the application fully, he could not build sufficient context about the project. He shared that he had to attend multiple meetings with the design team and his manager to understand the UI design and functionality expected from controls he could not access.

Both P10 and P16 shared that they had pushed for making their UIs accessible, not only to make themselves more productive but also for other screen reader users. However, it was difficult to implement accessibility in legacy UIs, an issue also raised in several email threads (n = 5). Furthermore, workplace dynamics complicated the implementation of accessibility. P10 mentioned that the changes had to be approved by senior management, who may consider the trade-offs between his productivity as a developer and the time it would take to improve the accessibility. P16 shared that his position as the only blind person in the organization and as a new member of the team foregrounded his request. Insisting upon accessibility could suggest to the team that he was not able to do his job as well as other developers.

Participants (n = 3) shared that poor accessibility of the UI presented challenges during demonstrations. In meetings involving stakeholders and clients, it could also suggest poor quality of work by the team:

> P16: *The problem is that when you want to give a demo to a client and there is accessibility issues, it slows you down [...] and they might think that you are not capable enough to do these things*

P16 further added that in remote client meetings during the COVID-19 pandemic, poor demonstrations could disclose his disability and reinforce ableist perceptions about his ability and competence as a programmer. Therefore, when presenting the UI to an external audience, participants generally had a sighted team member *"click on buttons for fill these forms for me"* (P16) while they handled the technical narration. The approach allowed them to present and highlight their contributions. P3 also shared that he occasionally recorded his screen while operating the UI to capture the workflow and do *"non-live demo"* and independent presentation (P3).

These instances highlight that accessibility issues in UI development could affect responsibilities beyond software engineering tasks, which developers are expected to perform in professional settings.

V. Discussion

A. Accessibility of the Programming Environment

The long-standing focus of HCI and software engineering research has been on improving the accessibility of programming tools [14]–[16] and programming activities such as debugging [14], [24], navigation [19], [20], and UI development [36], [40]. While these efforts are needed, our findings show that accessibility issues cannot be isolated to any particular programming tool or activity. They result from the interactions between the various software that make the programming environment — IDEs, browser developer tools, UI frameworks and libraries, operating systems, and screen readers. The combinations of these result in myriad configurations, which leads to a long-tail of individualized accessibility issues. The situation is exacerbated by the lack of (official) documentation and online resources that discuss accessibility. In the case of UI development, they complicate the processes of code writing, debugging, and ensuring accessibility with screen readers. They also impact collaboration between programmers with visual impairments and their sighted colleagues since they use different approaches to UI development.

While sighted developers can turn to large forums like Stack Overflow, the recourse for programmers with visual impairments is to reach out to one another and report the accessibility problems to the developer teams. However, we show that the differences in programming environments also make it difficult for programmers with visual impairments to give and receive instrumental help. We recommend researchers and designers consider the accessibility of the entire programming environment instead of considering accessibility improvements to any particular software. We also highlight the need to design platforms that can support information-seeking and help-seeking for programmers with visual impairments for accessibility challenges. We can draw on the archives of various online communities such as the program-l mailing list to create a wiki that documents preferred programming tools, UI frameworks and libraries, accessibility breakdowns to watch for, and their workarounds.

B. Meeting the Promises of UI Frameworks and Libraries

Our findings show that UI frameworks have the potential to allow for *relatively* independent UI creation with reduced need for sighted assistance. Familiarity with popular UI frameworks and libraries also made programmers with visual impairments eligible for growing employment opportunities in the field. However, the choice of the framework was moderated by the availability of accessible UI components and accessible programming tools, native *look and feel*, and cross-platform support. Our analysis revealed that many frameworks listed themselves as out-of-the-box accessible and cross-platform. However, programmers with visual impairments often found that both promises were only partially met. The behavior of the components depended on the interaction between the programming environment and screen readers, thereby interrupting the process of debugging, testing, and demonstrations for programmers with visual impairments. Since the visuals and the performance of the UI components remained consistent for sighted developers, they seldom realized the impact of using these frameworks and libraries on their colleagues. Thus, programmers with visual impairments had to either convince their team to switch to more accessible alternatives or work with the choices made by their colleagues. We recommend that official documentation of the UI frameworks and libraries should prioritize accessibility and mention screen reader compatibility. The approach would also benefit sighted programmers by making them aware of the accessibility issues and fixes required for the UIs to work consistently with different screen readers and platforms.

C. Limitations and Future Work

Despite our efforts to have a balanced gender representation, our interview study's sample was heavily skewed towards men. We believe this was due to the software engineering field and the online communities we recruited from being male-dominated. In future work, we aim to understand the accessibility challenges and experiences of gender-based minorities.

The programming experiences of our participants were likely shaped by the workplace norms and laws specific to their country and culture. While we highlight the access issues resulting from government sanctions on our Iranian participants, the interview study's sample size did not permit an analysis of differences due to participants' resident country.

Our participants and mailing list members had a variety of vision-related disabilities. Due to the small sample size and since visual ability varies on a spectrum, we did not analyze how the visual impairment's nature and onset correlated with our participants' programming experiences. Our findings and recommendations are intended for people designing programming tools and visual languages for screen reader users. We will interview developers who use screen magnifiers to expand our results to other assistive technologies in future work.

The period of this research overlapped with the COVID-19 pandemic. Only one interview participant (P16) shared how the pandemic affected his remote work experience. While none of the sampled threads mentioned the pandemic directly, an analysis correlating with the pandemic dates could surface accessibility challenges due to remote collaboration.

VI. Conclusion

We conducted mixed-methods qualitative research to understand the experiences of programmers with visual impairments with UI frameworks and libraries. We show that the promises of cross-platform support and out-of-the-box accessibility are only partially met for programmers with visual impairments. Our findings highlight that accessibility barriers in UI frameworks and libraries interrupt critical programming processes and affect collaboration. We recommend prioritizing accessibility in the official documentation of UI frameworks and libraries. We also urge HCI researchers and practitioners to consider supporting the information and help-seeking needs of programmers with visual impairments.

References

[1] J. Brains, "The state of developer ecosystem 2021," 2021. [Online]. Available: https://www.jetbrains.com/lp/devecosystem-2021/miscellaneous/

[2] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, "Evaluating cross-platform development approaches for mobile applications," in *International Conference on Web Information Systems and Technologies*. Springer, 2012, pp. 120–138.

[3] Google, *Flutter*, Google, Mountain View, CA, 2022. [Online]. Available: https://flutter.dev/

[4] Meta, *React Native*, Meta, Palo Alto, CA, 2022. [Online]. Available: https://reactnative.dev/

[5] The Apache Software Foundation, *Cordova*, Apache, 2022. [Online]. Available: https://cordova.apache.org/

[6] S. Overflow, "Stack overflow developer survey results 2019," 2019. [Online]. Available: https://insights.stackoverflow.com/survey/2019

[7] Stack Overflow, "Stack overflow developer survey 2021," 2021. [Online]. Available: https://insights.stackoverflow.com/survey/2021

[8] S. Overflow, "Stack overflow developer survey 2021," 2021. [Online]. Available: https://insights.stackoverflow.com/survey/2021

[9] M. Pandey, V. Kameswaran, H. V. Rao, S. O'Modhrain, and S. Oney, "Understanding accessibility and collaboration in programming for people with visual impairments," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW1, pp. 1–30, 2021.

[10] S. Mealin and E. Murphy-Hill, "An exploratory study of blind software developers," in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 2012, pp. 71–74. [Online]. Available: http://www4.ncsu.

[11] R. M. Siegfried, "Visual Programming and the Blind : The Challenge and the Opportunity," *Science Education*, pp. 275–278, 2006. [Online]. Available: http://www.adelphi.edu/ siegfrir/molly

[12] K. Albusays and S. Ludi, "Eliciting Programming Challenges Faced by Developers with Visual Impairments: Exploratory Study," 2016. [Online]. Available: http://dx.doi.org/10.1145/2897586.2897616

[13] K. Albusays, S. Ludi, and M. Huenerfauth, "Interviews and Observation of Blind Software Developers at Work to Understand Code Navigation Challenges," 2017. [Online]. Available: https://doi.org/10.1145/3132525.3132550

[14] V. Potluri, P. Vaithilingam, S. Iyengar, Y. Vidya, M. Swaminathan, and G. Srinivasa, "Codetalk: Improving programming environment accessibility for visually impaired developers," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–11.

[15] E. Schanzer, S. Bahram, and S. Krishnamurthi, "Accessible AST-Based Programming for Visually-Impaired Programmers," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education - SIGCSE '19*. New York, New York, USA: ACM Press, 2019, pp. 773–779. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3287324.3287499

[16] V. Petrausch and C. Loitsch, "Accessibility Analysis of the Eclipse IDE for Users with Visual Impairment," 2017. [Online]. Available: http://www.cooperate-project.de/images/publications/EclipseSurvey.pdf

[17] C. M. Baker, C. L. Bennett, and R. E. Ladner, "Educational Experiences of Blind Programmers," 2019. [Online]. Available: https://doi.org/10.1145/3287324.3287410

[18] H. Sampath, A. Merrick, and A. Macvean, "Accessibility of command line interfaces," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–10.

[19] A. Armaly, P. Rodeghero, and C. McMillan, "Audiohighlight: Code skimming for blind programmers," *Proceedings - 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018*, pp. 206–216, 2018.

[20] C. M. Baker, L. R. Milne, and R. E. Ladner, "StructJumper: A Tool to Help Blind Programmers Navigate and Understand the Structure of Code," 2015. [Online]. Available: http://dx.doi.org/10.1145/2702123.2702589

[21] J. M. Francioni and A. C. Smith, "Computer science accessibility for students with visual disabilities," in *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, 2002, pp. 91–95.

[22] J. Hutchinson and O. Metatla, "An Initial Investigation into Non-visual Code Structure Overview Through Speech, Non-speech and Spearcons," in *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. New York, New York, USA: ACM Press, 2018, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3170427.3188696

[23] T. V. Raman, "Emacspeak—direct speech access," ser. Assets '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 32–36. [Online]. Available: https://doi.org/10.1145/228347.228354

[24] A. Stefik, A. Haywood, S. Mansoor, B. Dunda, and D. Garcia, "Sodbeans," in *2009 IEEE 17th International Conference on Program Comprehension*. IEEE, 2009, pp. 293–294.

[25] A. Stefik and E. Gellenbeck, "Using spoken text to aid debugging: An empirical study," in *2009 IEEE 17th International Conference on Program Comprehension*. IEEE, 2009, pp. 110–119.

[26] E. W. Huff, K. Boateng, M. Moster, P. Rodeghero, and J. Brinkley, "Examining the work experience of programmers with visual impairments," in *2020 ieee international conference on software maintenance and evolution (icsme)*. IEEE, 2020, pp. 707–711.

[27] K. M. Storer, H. Sampath, and M. A. A. Merrick, "" it's just everything outside of the ide that's the problem": Information seeking by software developers with visual impairments," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–12.

[28] F. Hansen, J. J. Krivan, and F. E. Sandnes, "Still not readable? an interactive tool for recommending color pairs with sufficient contrast based on existing visual designs," in *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, 2019, pp. 636–638.

[29] Y. Zhuang, J. Baldwin, L. Antunna, Y. O. Yazir, S. Ganti, and Y. Coady, "Tradeoffs in cross platform solutions for mobile assistive technology," in *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. IEEE, 2013, pp. 330–335.

[30] Microsoft, *Xamarin*, Microsoft, Redmond, WA, 2022. [Online]. Available: https://dotnet.microsoft.com/en-us/apps/xamarin/

[31] S. Mascetti, M. Ducci, N. Cantù, P. Pecis, and D. Ahmetovic, "Developing accessible mobile applications with cross-platform development frameworks," in *The 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, 2021, pp. 1–5.

[32] M. Longley and Y. N. Elglaly, "Accessibility support in web frameworks," in *The 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, 2021, pp. 1–4.

[33] J. Li, S. Kim, J. A. Miele, M. Agrawala, and S. Follmer, "Editing spatial layouts through tactile templates for people with visual impairments," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–11.

[34] C. Kearney-Volpe and A. Hurst, "Accessible web development: Opportunities to improve the education and practice of web development with a screen reader," *ACM Transactions on Accessible Computing (TACCESS)*, vol. 14, no. 2, pp. 1–32, 2021.

[35] K. Norman, Y. Arber, and R. Kuber, "How accessible is the process of web interface design?" in *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, 2013, pp. 1–2.

[36] C. Kearney-Volpe, C. Fleet, K. Ohshiro, V. A. Arias, and A. Hurst, "Making the elusive more tangible: remote tools & techniques for teaching web development to screen reader users," in *Proceedings of the 18th International Web for All Conference*, 2021, pp. 1–14.

[37] V. Potluri, T. E. Grindeland, J. E. Froehlich, and J. Mankoff, "Examining visual semantic understanding in blind and low-vision technology users," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–14.

[38] A. Borka, *Developer Toolkit*, 2019. [Online]. Available: https://addons.nvda-project.org/addons/developerToolkit.en.html

[39] A. Shetty, E. Jarjue, and H. Peng, "Tangible web layout design for blind and visually impaired people: An initial investigation," in *Adjunct Publication of the 33rd Annual ACM Symposium on User Interface Software and Technology*, 2020, pp. 37–39.

[40] V. Potluri, L. He, C. Chen, J. E. Froehlich, and J. Mankoff, "A multi-modal approach for blind and visually impaired developers to edit webpage designs," in *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, 2019, pp. 612–614.

[41] V. Potluri, T. Grindeland, J. E. Froehlich, and J. Mankoff, "Ai-assisted ui design for blind and low-vision creators," in *the ASSETS'19 Workshop: AI Fairness for People with Disabilities*, 2019.

[42] "program-l: V.i. programmers discussion list." [Online]. Available: https://www.freelists.org/archive/program-l/

[43] J. M. Morse, "The significance of saturation," pp. 147–149, 1995.

[44] Freedom Scientific, *JAWS for Windows*, Vispero, 2022. [Online]. Available: https://www.freedomscientific.com/products/software/jaws/

[45] NV Access, *Nonvisual Desktop Access*, NV Access, 2022. [Online]. Available: https://www.nvaccess.org/

[46] Apple, *Accessibility - Vision - Apple*, NV Access, 2022. [Online]. Available: https://www.apple.com/accessibility/vision/

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

[48] O. J. Foundation, *Dojo*, Open JS Foundation, 2022. [Online]. Available: https://dojo.io/

[49] B. C. Team, *Bootstrap*, 2022. [Online]. Available: https://getbootstrap.com/

[50] *Accessibility tree - MDN Web Docs Glossary: Definitions of Web-related terms — MDN*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Accessibility_tree

[51] *Aria - Accessibility: MDN*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA

[52] WebAccessibilizer, *Bootstrap*, 2022. [Online]. Available: https://www.stsolution.org/WebAccessibilizer/