Understanding Accessibility and Collaboration in Programming for People with Visual Impairments

MAULISHREE PANDEY, University of Michigan School of Information, USA
VAISHNAV KAMESWARAN, University of Michigan School of Information, USA
HRISHIKESH V RAO, University of Michigan School of Information, USA
SILE O'MODHRAIN, University of Michigan School of Information, USA
STEVE ONEY, University of Michigan School of Information, USA

There has been a growing interest in Computer-Supported Cooperative Work and Human-Computer Interaction to understand the experiences of programmers in the workplace. However, the large majority of these studies has focused on sighted programmers and, as a result, the experiences of programmers with visual impairments in professional contexts remain understudied. We address this gap by reporting on findings from semi-structured interviews with 22 programmers with visual impairments. We found that programmers with visual impairments interact with a complex ecosystem of tools and a significant part of their job entails performing work to overcome the accessibility challenges inherent in this ecosystem. Furthermore, we find that the visual nature of various programming activities impedes collaboration, which necessitates the co-creation of new work practices through a series of sociotechnical interactions. These sociotechnical interactions often require invisible work and articulation work on the part of the programmers with visual impairments.

CCS Concepts: • **Human-centered computing → Empirical studies in accessibility**; **Empirical studies in collaborative and social computing**.

Additional Key Words and Phrases: accessibility; collaborative accessibility; social accessibility; programming; collaborative programming; collaborative software development; help-seeking

ACM Reference Format:
Maulishree Pandey, Vaishnav Kameswaran, Hrishikesh V Rao, Sile O'Modhrain, and Steve Oney. 2021. Understanding Accessibility and Collaboration in Programming for People with Visual Impairments. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW1, Article 129 (April 2021), 30 pages. https://doi.org/10.1145/3449203

1 INTRODUCTION

StackOverflow's annual developer survey is considered to be one of the largest and most comprehensive surveys of people who code around the world [3]. In 2019, around 1,350 of the 90,000 respondents (~1.5%) identified as having a visual impairment. This is a small and unsurprising number—people with visual impairments face systemic barriers to employment [15, 58], are less

Authors' addresses: Maulishree Pandey, maupande@umich.edu, University of Michigan School of Information, 105 S State St., Ann Arbor, Michigan, USA, 48109-1285; Vaishnav Kameswaran, vaikam@umich.edu, University of Michigan School of Information, 105 S State St., Ann Arbor, Michigan, USA, 48109-1285; Hrishikesh V Rao, hrishir@umich.edu, University of Michigan School of Information, 105 S State St., Ann Arbor, Michigan, USA, 48109-1285; Sile O'Modhrain, sileo@umich.edu, University of Michigan School of Information, 105 S State St., Ann Arbor, Michigan, USA, 48109-1285; Steve Oney, soney@umich.edu, University of Michigan School of Information, 105 S State St., Ann Arbor, Michigan, USA, 48109-1285.

2573-0142/2021/4-ART129 $15.00
https://doi.org/10.1145/3449203

likely to pursue higher education [2], and are less likely to be employed than sighted people [15]. The steady increase in lucrative programming job opportunities has the potential to positively impact the aspirations and social mobility of programmers with visual impairments. Programming is also considered a *relatively* accessible field in Science, Technology, Engineering and Mathematics (STEM); most programming is text-based, making it easier to write code with assistive technologies (ATs) such as screen readers and braille displays. By contrast, many other STEM fields rely heavily on inaccessible diagrams and equations.

In recent times, programming has moved away from command-line software towards graphical user interface (GUI)-based software like IDEs and text editors. These software have several features that advantage the sighted developers but pose challenges for programmers with visual impairments and inhibit collaboration among coworkers [7]. Prior research on Human-Computer Interaction (HCI) has studied the challenges that programmers with visual impairments face but much of this work has focused on specific tasks and individual programming tools. Challenges in mixed-ability collaborative contexts remain understudied. This is a gap worth examining because of the social, academic, and professional implications it can have for programmers with visual impairments. Most software is built collaboratively; programmers often have to collaborate with other programmers and team members, including designers and project managers [42, 49]. Challenges in collaboration are likely to reinforce some of the ableist perceptions about the abilities of people with visual impairments and limit their opportunities for employment and advancement [30].

In this paper, we investigate the collaborative experiences of programmers with visual impairments with a focus on the following research questions: (1) What are the collaborative activities and associated challenges that programmers with visual impairments encounter in professional contexts? (2) How do programmers with visual impairments address these challenges? (3) What implications do these challenges have for solo and group work? We conducted semi-structured interviews with 22 people with visual impairments who are employed as software developers, data analysts, IT professionals, and researchers. They frequently collaborate with colleagues as part of their jobs. Our findings and the subsequent discussion are relevant to employers and designers who aim to create accessible and inclusive work environments. This work makes several contributions to the Computer-Supported Cooperative Work (CSCW) and HCI literatures:

- An analysis of our interviews with programmers with visual impairments, which provides insights into the logistics of working in mixed-ability workplaces. Our findings extend prior work by focusing on sociotechnical challenges such as communication, collaboration, help-seeking, and biases. Our findings also validate many of the challenges that prior work has found with inaccessible individual tools.
- A discussion to build on the current theorizing of accessibility of group work in HCI and CSCW. We recommend that future research in this area should examine interactions around help, especially provision of help by people with visual impairments.
- A discussion on the accessibility of collaborative activities in programming, and design recommendations grounded in our empirical contributions.

2 RELATED WORK

We build on prior work in two primary areas: the accessibility of programming tools (which has thus far focused on software rather than sociotechnical challenges) and the accessibility of group work in mixed-ability contexts.

2.1 Accessibility and Programming

Our work focuses on the accessibility of programming in mixed-ability collaborative contexts. Prior work in HCI has investigated accessibility challenges related to individual tools. Mealin and Murphy-Hill were the first to touch upon the high-level accessibility challenges in programming [51]. They found that programmers faced challenges when using Integrated Development Environments (IDEs), seeking information in IDEs with screen readers, and writing User Interface (UI) code. Subsequent studies have confirmed the lattermost finding [5, 72].

The access challenges in IDEs can be broadly categorized into four groups: (1) *discoverability* of IDE features, (2) *glanceability* of information in various panels, (3) *navigability* of code, and (4) *alertability* of errors and bugs [64]. These challenges are exacerbated by a lack of accessible information about the IDE features [63]—documentation about programming tools is often designed for sighted developers, relying on visual content such as screenshots [10]. In addition, the keyboard shortcuts that programmers with visual impairments use are generally complex [10]. This increases the cost of learning and deters programmers from switching over to new tools. Commonly reported workarounds to these challenges include either using text editors [51] or seeking sighted assistance [5]. However, programmers with visual impairments feel that seeking assistance draws attention to the additional time it takes them to complete programming tasks [6]. This is likely to be magnified in collaborative contexts, which we examine in our study. Researchers have developed tools to address the above challenges, such as audio-based tools to assist with navigability [11, 35, 44, 64]. Similarly, tools like SodBeans [75] and CodeTalk [64] suggest that audio-based tools can significantly reduce debugging time. There is also a push towards developing accessible programming environments that can be integrated with various programming languages [67]. While these solutions have demonstrated promise, they result in multiple disintegrated solutions for different problems. It also places the onus on programmers with visual impairments to find these tools, maintain appropriate versions, and integrate them into their workflow.

These studies and systems also focus on accessibility challenges with individual tools in isolated contexts. There is a gap in the literature about the challenges programmers face in social contexts. Among the few exceptions are studies that focus on the experiences of students with visual impairments in computer science programs [10, 27]. These reported that it is challenging for students to participate in class discussions and access visual materials like slide presentations, diagrams, and notes on whiteboards. These challenges persist in the workplace too [5, 51]. However, the studies do not discuss the social and personal implications of these challenges for programmers with visual impairments. In addition, the collaborative programming activities that are critical for success in the workplace [24] remain understudied.

2.2 Collaborative Programming

The software engineering, CSCW, and HCI communities have recognized the importance of collaboration and communication in programming [14, 48, 68, 73]. To coordinate the development and maintenance of complex software, the process often begins with planning the *software architecture*, which is "considered to be the structure of a large piece of software, presented as a nested set of box and arrow diagrams" [43]. The architecture communicates the relationships between different components like the database, servers, and UI [38]. It enables team members to develop a common vocabulary and facilitates communication.

Some workplaces use *pair programming*—a software development practice where two programmers work side by side to write and review code [26]. One programmer (the *driver*) is responsible for typing the code, while their colleague (the *navigator*) gives instructions and feedback on the code being written. Pair programming results in higher code quality, creative problem-solving,

knowledge transfer among team members, and higher work satisfaction for the programmers involved [59, 85].

Programmers and data scientists often have to concurrently edit the same code [34, 82]. They have to adhere to agreed-upon rules of code writing and styling to maintain the readability and consistency of code [65]. In large software development companies such as Google [47], Microsoft [1], and Facebook [78], code is peer-reviewed to ensure compliance with established guidelines [33]. This process is formally known as *code review*. It helps teams find defects in code, build awareness about the project, and find alternative solutions to problems [9].

The above collaborative activities have been extensively researched in the context of sighted programmers but remain understudied in mixed-ability contexts. Our study provides a more holistic view of the contextual work practices that develop in mixed-ability teams, revealing the interaction between programming tools, collaboration software, access technologies, and team members.

2.3 Accessibility and the Social

2.3.1 Assistive Technology Use in Social Settings. An important part of our study was understanding how programmers with visual impairments use ATs during collaboration. Accessibility research in HCI is increasingly examining the situated use of ATs and emphasizes considering social contexts when designing them [32]. In prior research, people with disabilities reported that ATs tend to lag behind mainstream products in functionality and aesthetics [70]. They tend to attract unwanted attention to their users due to their design [69, 70] and breakdowns [4, 71], which foregrounds the users' disability [89]. Thus, for people with disabilities, deciding whether to use ATs in social settings is a negotiation between utility, avoiding attention, and feeling self-conscious due to the resulting attention from others in the space. We add to this body of work by studying AT use in a professional context, where such decisions can have additional implications for productivity, perceived competence, and independence.

There are also misconceptions among people without disabilities that ATs make a disabled person "normal" and that their ability is contingent on ATs [70]. Shinohara and Wobbrock therefore recommend designing ATs that enable users to convey their ability and identity [71]. For instance, studies have shown that people with disabilities value their sense of independence [46] and the outward appearance of independence [56]. ATs should then help convey one's independence to others in social settings. This is known as designing ATs for "social accessibility" [71], and is likely to foster sociotechnical access for people with disabilities and enable them to participate in social settings [58]. Shinohara *et al.* suggest three AT design tenets for fostering social access: (1) involving users with and without disabilities in the design process to ground AT design in the mainstream, (2) considering both functional and social scenarios of AT use, and (3) using design methods that foreground social contexts of use [69].

2.3.2 Help-Seeking and Help-Giving. Seeking assistance in the workplace is an important way for employees to resolve their problems. Gourash defines help-seeking as "any communication about a problem or troublesome event which is directed toward obtaining support, advice, or assistance in times of distress" [41]. The process of seeking help consists of three parts: recognizing the problem, consciously deciding to act on it, and selecting a source for help [29]. Individual attributes like gender, education, race, socio-economic status, and age have been considered when studying the help-seeking process [13]. In the workplace, employees prefer to reach out to experts or senior employees, as they find the help of higher status individuals and experts to be more constructive [57]. Help-seeking from superiors and experts is shaped by awareness of their expertise and ease of access to them [79]. In addition, employees need to trust that help-givers will not judge them for seeking assistance [79]. It is easier to seek help when the problem is shared by many employees, as

this attributes the problem to external sources and reduces the risk of judgment [13]. The threat to self-esteem and the inability to reciprocate help can deter people from seeking it [8]. **This raises questions about seeking assistance with accessibility challenges, a problem shared only by employees with disabilities.**

Help-giving is relatively less studied in research but is considered to be closely intertwined with help-seeking and requires interpersonal interaction among employees. It relies on employees' "sense of citizenship" since they are not formally necessitated to provide help to others [13]. The desire to reciprocate assistance is a key motivator for help-giving in the workplace [40].

Research has attempted to understand when people with disabilities seek help and how it affects them. When seeking assistance as a recourse from malfunctioning ATs, the needs of the people with disabilities are often misunderstood and their autonomy is overridden [84]. For instance, people with visual impairments have reported that sighted people tend to navigate them and provide information that is not useful. Prior studies have referred to this as unwanted help—assistance provided based on incorrect assumptions about people's abilities and without due understanding of their needs [76, 84]. Seeking assistance also has social costs [83, 88], making the person appear less competent and highlighting their disability. In research with people with visual impairments, participants indicated that they felt the need to reciprocate the help and did not want to burden their friends and family [21]. Instead, they preferred using sighted assistance from crowd workers [21, 61]. This adds more perspective to the question we raised earlier: how do programmers with visual impairments feel about reaching out to sighted people, including their colleagues?

2.3.3 Accessibility in Mixed-Ability Contexts. There is an increasing emphasis on understanding accessibility in mixed-ability contexts [22, 23, 87] and designing technologies that respond to peoples' abilities [86]. This is evident from the various technology-mediated solutions designed to facilitate group work in online photo-sharing [50, 88], learning [36, 52, 54, 55, 77], sports [12], and music creation [60].

Branham and Kane studied how accessibility was achieved and maintained by inhabitants in the context of home spaces [22]. They defined this as *collaborative accessibility*—"taking active roles in co-creating an accessible environment". They found that accessibility was intertwined with personal relationships. Thus, accessibility (and the lack thereof) affected how couples or housemates shared experiences, which in turn impacted the well-being of their relationships. Addressing certain inaccessibility-related challenges could foster kindness and care. In such contexts, technologies should be designed keeping in mind the interdependencies within such relationships [16]. Technologies should also be committed to helping people achieve what matters to them and not be focused solely on accomplishing tasks [18].

It has been shown that people with visual impairments have to perform *invisible work* [74] to address accessibility challenges. While accessibility is created through coordination between multiple technologies and people [25, 46], the onus falls largely on people with visual impairments [31, 83]. For instance, Das *et al.* observed that software updates often break the accessibility of writing tools and ATs. People with visual impairments have to reconfigure the settings and relearn the keyboard shortcuts to continue to collaborate with sighted people on writing projects [31]. Thus, people have to work beyond their professional responsibilities to find accessible solutions [23] and continually advocate for their access needs [81]. The above studies demonstrate the need to develop a situated understanding of technology use to uncover its social implications for solo and group work. Our research contributes to this growing body of work.

3 METHODS

3.1 Participants

We conducted semi-structured interviews with 23 people with visual impairments (19 male, 4 female). The eligibility criteria for our study were that participants should be at least 18 years old and self-identify as programmers. We recruited participants through personal contacts (n=3), snowballing (n=2), and posting the recruitment call online (n=18). We posted on the program-l[1] mailing list (n=16), which comprises programmers with visual impairments, and r/blind[2] (n=2), a subreddit for people with visual impairments.

Participants (P1–P23) were between 24 and 73 years old. We excluded one participant (P4) from our final analysis because he self-reported his visual impairment as low vision while the remaining participants identified as nearly or fully blind. As a result, P4 used screen magnification on his digital devices while the other participants used screen readers or a combination of screen readers and braille displays. The screen readers mentioned by the participants included NVDA, JAWS, Orca, and ZoomText.

Table 1 in the Appendix lists the demographic details of each participant, their self-described visual ability and programming experience, the programming languages they currently use, and the nature of the organization they work(ed) in. Our participants included software engineers, data analysts, IT professionals, freelancers, and researchers. They were employed in software companies, universities, research organizations, and NGOs. Our participants were based in the United States, Europe, Africa, India, and China. In some cases, there are very few professional programmers with visual impairments in the entire country, making it relatively easy to identify the participants. Therefore, to preserve participants' anonymity, we are not listing the countries they came from.

3.2 Procedure

We obtained approval to conduct the study from our university's Institutional Review Board (IRB). We conducted the interviews on participants' preferred platforms, which included phone, Skype, Google Hangouts, and WhatsApp. Interviews typically lasted 45–65 minutes and were all conducted in English as the participants were comfortable with the language. Each interview was audio-recorded (with participants' informed verbal consent prior to the start of the study) and transcribed verbatim by a third-party transcription service (approved by our university's IRB). The first author verified each of the transcripts. Each participant was compensated with an Amazon gift card worth USD 15 or the equivalent amount in their local currency.

The questions in the interview protocol were organized across three sections. The first section focused on the participants' background[3], programming education, and experiences. The second section elicited details about the software they used for programming. We asked them to describe the reason behind their choice of software and how it fit into their programming workflow. We also asked them about the accessibility challenges they faced using this software and the workarounds they adopted to address the challenges. The last section focused on their collaborative experiences with other programmers. In the initial 5–6 interviews, we asked participants to share the collaborative programming activities they participated in. This led to participants naming activities like code reviews, pair programming, UI development, etc., and giving an overview of how they carried out these activities. In the latter interviews, we asked more specific questions about the existing

[1] https://www.freelists.org/list/program-l

[2] https://www.reddit.com/r/Blind/

[3] We made a conscious decision not to ask the participants about their visual impairment. We instead asked them about their AT usage in the context of programming. When answering these questions, most participants described their visual ability.

practices around these activities and the processes through which participants adopted or modified these practices to achieve collaboration.

3.3 Analysis

Before starting the analysis, we pre-coded [66] the data as we conducted the interviews. We highlighted quotes and sections in printed transcripts, and we also wrote analytic memos [66] to identify emerging themes and missing details in the data to refine the questions for subsequent interviews. In the first round of coding, we used descriptive codes [66] to identify various programming activities, collaborative activities, challenges faced by participants, and workarounds. We further organized programming activities into three categories: (1) the pre-programming stage, focusing on installation and integration of various tools; (2) the programming stage, focusing on code writing, debugging, and compiling; and (3) the post-programming stage, focusing on code sharing. In the second round, we used pattern coding [66] to reorganize the codes from phase 1 into five high-level themes: (1) group work, (2) ecosystem of tools and assistive technologies, (3) sighted assistance, (4) extra work, and (5) social and personal implications.

4 FINDINGS

Our findings are organized into three broad sections. We begin by describing the tools that participants used to perform programming and related activities. Next, we discuss the accessibility challenges in collaborative programming activities and the practices that participants co-created with their colleagues to address these challenges. In the final section, we discuss the various social interactions like help-seeking and advocacy that participants performed to negotiate these practices.

4.1 Tools Used in Collaboration

Being a programmer involves much more than writing code [62]. Our participants mentioned that they spent significant time on project planning, communicating with colleagues, and coordinating with others to write code. In this section, we elaborate on the different tools that participants used, how these informed their programming workflows, and the accessibility challenges participants faced. We find that making programming tools (such as IDEs and debuggers) accessible is necessary but not sufficient; for effective collaboration, the entire ecosystem of tools needs to be accessible.

4.1.1 Choosing an Editor. Prior work has discussed the various accessibility issues within IDEs[4] and text editors[5] [5, 51, 64]. **We add to prior work by reporting on how participants selected and set up their programming environment. This initial process also presented accessibility challenges that could take significant time to resolve and impact programming**:

> *I think for most people that would probably agree that it's like setting up the [programming] environment to start with, takes the time and getting all the tools lined up.* – P20

Participants described the various steps they would perform before setting up an editor. First, they would assess the compatibility of the software with their ATs. The most common way to assess accessibility was checking if the software documentation mentioned screen readers and keyboard shortcuts. In other cases, participants reported emailing the developers of the IDEs to check whether they had been tested with screen readers. They would also post on mailing lists dedicated to programmers with visual impairments. On these mailing lists, participants did not have to provide additional details about ATs to other members. Plus, members would share their personal experiences with code editors, leading to a more informed choice. By contrast, members

[4]Integrated Development Environments (IDEs) allow users to write, compile, execute, and debug code within one application.
[5]Text editors only allow users to write code, meaning they must use a different application to compile, debug, and execute their code.

of larger Q&A sites like Stack Overflow seemed to have a limited understanding of ATs and could not provide useful advice:

> *No one on Stack Overflow is discussing the fact that in order to use Visual Studio Code with JAWS, you have to restart JAWS or you can only have one Visual Studio code window open at a time or you know that there's some weird interaction with the virtual cursor. Like no one's going into that level of, of niche detail on Stack Overflow.* – P9

The next step was identifying the installation options available to participants. Participants preferred to install software through the command line, but this option was not available for many editors. The more frequently available alternative was the installation wizard[6]—a GUI with a series of dialog boxes for installing and configuring software. Participants reported that sometimes the GUI *"installers aren't accessible whereas the programs themselves are"* (P1). Thus, they would have to seek sighted assistance to help install the programming tool and its packages (e.g., to click inaccessible combo boxes and pop-ups).

Beyond installing local development environments, programmers must also set up the environment in which their code will run. This includes setting up deployment infrastructure and referencing third-party Software Development Kits (SDKs). SDKs—which allow developers to access resources like proprietary data, functionality, or computing power—have become increasingly popular with the rise of cloud computing platforms. However, developers can also face accessibility barriers when configuring these SDKs.

> *To start programming for the Alexa, you need to create an account on their website [...] you need to submit the form [...] And I need to spend like five minutes looking for that [submit] button and I accidentally like scrolled up to the top, and then I saw that on the top it says 'Submit'. So those are some issues, they can cost time until you find them. – P1*

Participants observed that their sighted colleagues did not always have to go through a similar process when installing or updating their programming environments:

> *Everyone else sighted who is starting out can just download this program, click the big green button, and there you go, you are done. But I have gotta learn the command and all the switches to use, and how to specify a path on the command line, just extra stuff!* – P3

Many steps from the process outlined above had to be repeated when participants changed their code editors. **Further, participants' choice of code editors was not determined individually; several social and logistical factors influenced their choice.** For starters, the decision depended on the complexity of the project, often determined by the number of lines of code, the number of files one has to work with, and the number of programmers involved. Many participants felt that it was faster to *"write a small program, say, 100 to 200 lines program"* (P23) in a text editor. But with projects involving longer programs and multiple files, they preferred an IDE:

> *I think where it gets taxing is when you have to maintain a project, say you're developing a web application in Java. Then it's so hard to do all the conflict files and just pair the WAR file and everything manually... the IDE does it so easily.* – P23

Complex software projects generally also involve larger teams with more programmers. If the programmers on the team were using a particular IDE, participants preferred using the same IDE to be consistent with their colleagues. This often meant compromising on accessibility, and additional work on our participants' part. For example, P17 had to switch between two versions of Visual Studio for improved accessibility and to keep the codebase backwards-compatible for his team:

> *I have both [Visual Studio 2017 and 2019 editions] installed on my computer and sometimes I'll need to bounce into 2019 because it works a little bit better for some accessibility. But I*

[6] https://en.wikipedia.org/wiki/Wizard_(software)

> *make sure that any of the builds and everything I do really comes from 2017 because we want it to be in the same thing that everybody's using.* – P17

The choice of programming tool also depended on how easily participants could switch to applications they were using concurrently. Generally, the teams used software that was designed for Windows and Mac operating systems. As P9 pointed out, Linux was his preferred operating system for programming and would allow him to program more efficiently. However, the screen reader on Linux would reduce the accessibility of other applications he used in parallel:

> *I find that Windows is best accessibility-wise, and it does fit best into the work infrastructure [...] it's primarily Windows directory, Outlook, Office 365 [...] If I really wanted to try to use Linux, then that would be supported [...] But I look at Linux accessibility every once in a while, and I think that in the GUI with Orca and all that it's just not far enough along for me to really be competitive.* – P9

4.1.2 Working Outside of the Code Editor. Besides IDEs and text editors, participants would interact with other software related to project management (e.g., JIRA, Microsoft Teams), file sharing (e.g., Git, Microsoft Teams), communication (e.g., Slack, Skype), software design (e.g., LucidChart, Microsoft Visio), and internal tools (such as code-reviewing platforms, databases, virtual machines, and web servers). The information on these software informed their programming activities. Therefore, accessibility breakdowns in these tools had a direct bearing on their ability to carry out their responsibilities:

> *The laptop that I'm using at work right now only has 8 gigs of RAM and it has an integrated graphics card rather than a dedicated graphics card [...] I'm having say a SQL server sticky and Excel or maybe even Visual Studio Code open all at the same time as well as ZoomText. ZoomText is so graphically intensive, it is using 60 to 90% of the GPU at any given moment, to magnify whatever I may need to be magnified. But all that is also putting stress on the computer's memory. So the computer slows to a crawl.* – P7

The above quote informs us about the complex ecosystem of tools that participants have to use simultaneously. It also tells us that the accessibility challenges can present themselves repeatedly due to concurrent use of tools. Next, we share specific instances of challenges that participants had to tackle on a regular basis.

Participants often had to use software like JIRA to track issues in their projects. They were required to log into the software to retrieve the project features and bugs assigned to them. However, the accessibility challenges in the software necessitated seeking sighted assistance:

> *I get someone visually and they come over, I say, "Okay, Joe. You told me that there is a ellipses button, that's a status button there. I'm not finding it!" And then he'll stand next to me [as] I press [the] tab key. – P17*

P17 described how he worked with a sighted colleague, Joe, to overcome the accessibility challenges with JIRA. P17's screen reader, JAWS, is unable to read some of the JIRA buttons that are visible to sighted people. As a result, P17 is unable to identify the button he should click to bring up the tasks assigned to him. Joe verbalizes the content on the screen and announces the results of interactions that P17 performs with his screen reader. Thus, completing a seemingly simple task like clicking a button and determining its result necessitates sighted assistance and additional work, i.e. a series of interactions without which they cannot complete their job as a programmer.

Similarly, many participants reported using software such as Slack, Skype, and Microsoft Teams to communicate with team members, text project details, and share code snippets. These software enabled informal conversations and quick coordination for sighted colleagues. However, the accessibility constraints of the software did not afford the same ease and efficiency of communication to

our participants. They shared that it was *"daunting and frustrating"* for them to *"go through tons of threads"* (P11) and locate the messages pertinent to them. When colleagues would use these software during remote collaboration to share their screens, participants would *"only get the talking part and [...] miss out on the screen share portion"* (P9).

Participants preferred managing the accessibility challenges independently. However, the presence of deadlines, the time required to implement different solutions, and the increasing frustration of not finding *"a way around"* (P5) necessitated seeking sighted assistance. More importantly, these challenges had an impact on their ability to write code and collaborate with their colleagues, as we describe in the next section.

4.2 Emergent Practices in Collaborative Programming Activities

Our participants shared with us details of activities where they collaborated with other members of the team: (1) code writing and styling, (2) code reviews, (3) pair programming, (4) software design, and (5) UI development. Our analysis revealed that they worked with their colleagues to modify the established work practices around these activities, resulting in practices that were more accessible.

4.2.1 Code Writing and Styling. When programming as part of a team, programmers often have to follow coding standards. These are generally rules regarding the visual presentation of code so that it is more readable and navigable, and sections of code are easily identifiable. For example, Google's JavaScript style guide[7] specifies rules regarding the use of braces and indentation, declaration of variables, addition of comments, and more.

Participants reported that they learned the code-styling rules when they started collaborating with sighted programmers. For instance, indenting code blocks enables sighted programmers to easily identify relevant sections of code when scrolling past them [53]. However, since indenting did not serve any visual purpose for our participants, they did not consider putting additional spaces in the way they wrote code:

> *From very early on, I got into habits that were better on a braille display. You don't put spaces around equal signs because [...] you could fit two more characters on your braille display [...] I would always put brace on the same line [...] So kinda stylistically I learned some things that I have since discovered are not mainstream. – P3*

The quote highlights that participants developed code-writing habits to make the best use of the limited space available on the displays (typically 20–80 characters). Thus, their manner of writing code was at odds with that of their sighted colleagues. When possible, they preferred removing characters like extra whitespaces, braces, and trailing punctuation in the code they received from their sighted colleagues. Sighted programmers' use of whitespace characters like tabs and spaces to indent code created problems on screen readers too. These characters were announced on screen readers and slowed down the participants during code reading and navigation. Thus, they preferred removing the indentations from the code. Participants further spoke about the lack of nuanced information on screen readers. For instance, inconsistent use of case styles led to illegible pronunciation on screen readers. While poor capitalization and naming are frowned upon by sighted programmers too, they can make sense of it visually. This information is invisible to screen readers:

> *It shouldn't all be,* '`thisismyname`'*. The variable is called* '`thisIsMyName`'*. It shouldn't all be in lower case! It shouldn't all be upper case!* – P17

Participants reported that they found it easier to navigate, search, and edit their own code as compared to other people's code. **Working with others' code was more challenging due to the**

[7] https://google.github.io/styleguide/jsguide.html

combination of (1) inaccessibility of programming software, as described in the previous subsection; (2) limitations of the access technologies in providing important information, described above; and (3) how their team members wrote code, described above. Participants had therefore developed code-writing strategies to work alongside other programmers:

> *I was using the comments and the separated dashes [...] when you hear a line being read as dash dash dash dash dash, then that's how somebody would know up here comes my next comment. This and kind of just as much description as possible.* – P11

> *I had made this thing that everyone would put their initials followed by the time of when they were changing a particular code block in comments above the code block and then mark begin. And then after they're done changing N number of lines, at the end they would again put a comment and say end of changing this.* – P23

Thus, strategies like unique commenting style and descriptive comments enabled participants to identify sections in the code efficiently without assistance. Participants also shared these strategies with their colleagues, either informally or in code reviews (next section), who were often willing to follow them. For the benefit of sighted programmers, they would follow visually focused styling rules. This demonstrates that participants and their colleagues would collaborate on establishing code-styling rules that were better suited for mixed-ability programming contexts.

4.2.2 Code Reviews. Some participants reported that their teams had formal code reviews, through which some of the more complex practices related to code writing were developed. Participants shared how code reviews made sure everyone on the team wrote (1) shorter code segments that made navigation easier on ATs; (2) documented the code, which reduced the task of information-seeking and made searching the codebase more efficient; and (3) reduced redundancy in code, which again positively affected code searching:

> *Everything in our code is just completely modularized. If you have more than 30 lines of code in a function, everyone's like refactor this put it into helper file.* – P18

The activity also provided a platform for all programmers to deliberate on the best code-writing strategies and share tips for improving code clarity. **Thus, code reviews ensured that our participants did not have to articulate their preferred code-writing practices to their colleagues separately**—they could do so as part of the activity. This meant less work on their part in *"getting people on the same page to code in the same way"* (P11). Plus, participants knew what was expected of them in terms of code styling and knew what to anticipate from their colleagues:

> *By then doing that, which is sticking to standard practices for programming, then it's beneficial for all.* – P17

Participants spoke positively about the code-review activity if the software used to facilitate it was accessible. This allowed them to perform efficiently without asking others for help. P18 compared his experience at his current organization with that at his previous organization. In the current workplace, his team used a web-based code-review system that was accessible with screen readers. He explained he did not have to ask for accommodations and he was able to participate in the activity like his other colleagues. In his previous workplace, a sighted employee was hired specifically to assist him with the inaccessible code-review system. This not only affected his collaboration experience but also impacted his productivity. It would take him a *"couple hours a day"* (P18) just to share his comments on improving the code. This reemphasizes the importance of looking beyond the accessibility of programming tools for collaboration in mixed-ability contexts.

4.2.3 Working Together and Pair Programming. Many participants reported that their teams practiced pair programming. As the related work section describes, one programmer (the *driver*) is

responsible for typing the code while their colleague (the *navigator*) gives instructions and feedback on the code being typed. Participants expressed that while they could perform as the driver, they could not easily reverse the roles and give directions as the navigator:

> *So usually I'm the person writing the code because obviously I can't look over the shoulder [...] I would like to have to be able to do the same thing, but it's not essential, me being the person looking over the other person's shoulder.* – P19

The quote shows how participants' contributions in pair programming are limited since they cannot access their colleagues' computer screens. The complications arise due to (1) lack of access to colleagues' computers, (2) social and legal limitations with regard to the installation of ATs, and (3) colleagues being unable to describe all the details of the problems to participants. Next, we describe each of these in detail.

Participants mentioned that ATs were generally not installed on their colleagues' computers, **meaning that the code was inaccessible to them during real-time collaboration. This not only made synchronous problem-solving challenging but it also prevented participants from providing help to their colleagues in real time.** This was exacerbated due to sighted colleagues *"trying to describe what's going on"* while *"operating a computer"* (P9). This can be understood as sighted colleagues attempting to explain visual and textual details that would be important for participants to provide assistance. To work around these challenges, participants preferred their colleagues to share the code with them. This way they could read the code on their computer that already had the ATs set up:

> *My coworkers either need to provide things in text form or they need to come and sit with me or I need to be the one driving the computer that's used to find the problem.* – P9

P9's quote reveals the breakdown in providing real-time assistance. He would ask his colleagues to email him the code snippets. Other participants also echoed a strong preference for email over the chat feature in software like Slack and Microsoft Teams. As section 4.1.2 describes, teams would use these software to share code snippets but accessibility challenges prevented our participants from locating the right message efficiently.

Another alternative was adopting a multi-step process, which defeats the purpose of pair programming to a degree. In this process, the code is uploaded to a shared repository, from where it gets pulled and set up on the computer, and then the changes are reviewed. This approach again does not have the benefits of working side by side, such as discussing issues spontaneously and providing feedback in real time. It is also time-consuming, especially with large codebases.

P7 shared her workaround for achieving *near* real-time collaboration instead of following the series of steps described above. She used ZoomText, an AT that combines magnification and screen-reading technology. Until very recently, she would switch off text magnification during pair programming. This allowed her sighted colleagues to read and navigate the code on her computer but prevented her from understanding the changes they were making in real time. After the changes were done, she would switch on magnification again to review the changes. This provided intermittent access to the participant and her sighted colleague. P7 then went on to talk about a recent feature that made screen sharing and, consequently, pair programming easier:

> *Dual monitors support was added like maybe a year ago [...] it gives you the option to operate one screen magnified. So my screen could be magnified for me [...] with that same image on another screen in regular size. So if I'm working with someone who's sighted and we have two monitors, then that makes it a little bit easier.* – P7

Thus, the collaboration still happened on her computer but it enabled synchronous work. We also see how the slow introduction of features to ATs, compared to mainstream technologies, negatively impacts the collaborative experiences of programmers with visual impairments.

She and a few other participants said that sometimes they would install ATs on their colleagues' computers, provided they were willing:

> *They allow for a free trial version, that's roughly 45 minutes in duration. If I have a coworker [...] if they're willing to install it, then they can put it on their computer for us to troubleshoot a bug or something [...] So that's something that's helpful and useful.* – P7

> *I have a professional license for that [JAWS] [...] that just allows me to be able to install that on any of my work computers [...] as long as no other visually impaired person or other people actually use that software package.* – P17

This reveals that participants have to switch and share computers in the workplace to collaborate effectively. To access colleagues' computers, they may have to reinstall the AT. But sharing of ATs, specifically screen-reader software, is complicated by the limits on trial versions, policies around AT use, and due consent of colleagues. In addition, the onus of establishing access generally falls on the participants and not their colleagues.

When it was essential for both the participant and their colleague to be working on their respective computers together, they preferred using communication software to do a screen share with the participant sharing their screen. This can be thought of as switching to a collaboration style akin to *remote* pair programming. The screen reader did not interfere with their discussions. Plus, the screen share allowed the sighted colleague to view where the participant was in the codebase. Both programmers could paste specifics in the chat window to facilitate efficient search and edits to the code:

> *I'll pull the file up that they're doing as well and they say what they're looking at [...] And then by us having the instant message window open, they can paste in the line of their code that they're talking about [...] or they can tell me the line number.* – P17

Thus, participants needed to access the information on their computers as well as the information on their colleagues' to achieve collaborative tasks. They had to collaboratively establish different workarounds with their colleagues to achieve what sighted programmers are able to carry out relatively easily by virtue of being able to view each other's screens. This shows that accessibility of the end user's computer is necessary in the workplace but not always sufficient.

4.2.4 Software Design. Participants reported facing challenges in accessing and creating diagrams that represent the software architecture. Participants shared that their teams used online tools (LucidChart[8], Microsoft Visio[9], draw.io[10]) and whiteboarding to prepare the diagrams, which were often presented in team meetings. Participants felt their colleagues *"couldn't translate those diagrams into words"* (P16) and describe all of the necessary details to explain the software architecture. **The lack of access to software design (1) impacted participants' understanding of the software, (2) prevented them from providing feedback to their colleagues, and (3) limited their contributions to the project**:

> *I wouldn't know how the computers connected together [...] So I couldn't contribute there [...] the way the others worked was anyone could have managed issues of the sequencing of the design process [...] I didn't ever have a sequence of jobs that needed fixing. It was always just a single program that was part of a process that had a breakdown in it.* – P16

[8] https://www.lucidchart.com/
[9] https://en.wikipedia.org/wiki/Microsoft_Visio
[10] https://drawio-app.com/

P16 further spoke of the visual nature of the *"progress reports of a program"*. His colleagues could view the reports to measure the impact of their contributions while he had to acquire the same information from his boss. He was unable to take on more active roles and had to *"differ to the team leader for the jobs to be done"*—only doing the jobs that were accessible according to the team leader. Thus, his contribution was pigeonholed to code writing and troubleshooting. It is important to mention that P16 is a retired software developer. He was recounting his past work experiences in the 1970s and 1980s, and it is likely that his experiences may not generalize to that of programmers today. However, it highlights how lack of access to high-level software design can result in programmers with visual impairments being assigned fewer responsibilities in comparison to sighted programmers.

Teams often had the requirement that the architecture should be prepared visually and not rely on descriptions, especially for complex projects. P18 shared that, for minor projects, he wrote out descriptions of the system design along with the functions that needed to be implemented. He shared the *descriptive design* with his team to seek their feedback in design meetings. For larger projects, he would *"try to put off or skip the step"*. P18 mentioned that his performance on this activity had even come up in his annual review. His manager felt that creating and presenting these diagrams in meetings showcased his contributions to other teams and therefore insisted on visual diagrams. However, with the available *"design tools and design languages"*, P18 could not *"really do much except write a description"* of what he was doing. In P18's case, he and his manager agreed to collaborate on the activity:

> *My manager said maybe I can write up a description of what all the components do and how they all work. Then he can sit with me and help me make a component diagram, which he says should be pretty simple and straightforward.* – P18

The co-creation of the system diagram was going to reduce the extra time and emotional stress that P18 would have to go through if he were to work on it alone or seek help from others. The nature of this collaboration also presented him as the primary contributor, since he was writing the descriptions on which the diagrams were based:

> *If he can help make the little diagram based on what I wrote in the text [...] That's just a relief! [...] I'm not one of those people who cares about I need to independently do everything myself. [...] As long as someone helps me get it done and I'm doing the majority of my own work, that's fine with me.* – P18

4.2.5 UI Development. Prior work has mentioned UI development as particularly challenging due to the visual nature of the activity [51, 72]. **We add to the prior work by describing the nuanced reasons due to which participants had to seek sighted assistance frequently in this activity. These included (1) high-level design specifications, (2) inaccessible design documents, and (3) inaccessible UI developer tools.** We also share practices and workarounds that reduced the need for sighted help, enabling participants to work relatively independently.

Generally, the development of the interface is preceded by a discussion phase where designers and programmers come to a common understanding of the form, functionality, and interactions of the GUI. Often these discussions happen over visual artifacts like wireframes and design documents, which developers refer to as they write code. These artifacts contain details on colors, sizes, and placements of GUI elements on the screen. Alternatively, the discussions can be informal, with the developers being informed of the general layout and interactivity of the GUI by either the designer or the manager. In this case, the design guidelines are high level and strict rules and guides are not provided. The decisions regarding the granularity of design documentation depended on the

nature and practices of the workplace. For instance, one participant reported that her previous organization was fairly small and therefore developers were also responsible for the design:

> *They [the employer] didn't really have the concept of teams, everyone was an individual contributor [...] So when you have an independent project that you are working on, you don't just code, you also have to design the interface. [...] it [the UI design] was communicated in text [...] these are the forms and these are the controls that we need [...] nothing in detail like, this should be '10 pixels away from this' sort of thing, no!* – P23

When working with loosely defined documentation, participants had to seek sighted assistance often. We noted multiple challenges for our participants. First, they could not verify the aesthetics of the UI as sighted programmers could. They could not decide if the UI *"looked good enough"* (P23) and needed sighted people's opinion. Second, when inspecting the visual output with screen readers, it would announce the UI elements linearly, i.e. in the order in which they appeared in the code. Thus, participants could not conclude if something was off the screen margins. Similarly, the UI element could be present on the screen but not necessarily be visible:

> *Sometimes some of them will overlap with each other... And though I could hear two different buttons but it could be the buttons are on top of each other.* – P23

Third, in the context of web-based applications, a sighted programmer can use web-inspector tools in the browser to make temporary changes and inspect how this modifies the interface. However, web-inspector tools were not accessible to the participants. For example, P11 shared how the screen reader would not announce the URLs on HTML pages nor inform her about text styling, i.e. whether it was italicized, bold, underlined, etc. To verify this information, she had to refer back to the HTML and CSS code in the text editor. She would have to search and navigate to the right section of the code to get the necessary information and make the changes. This reveals the additional steps that she has to perform compared to a sighted front-end programmer. Finally, participants shared that it was difficult to calculate measurements for width, height, margins, paddings, and placements of UI elements. P6 shared one way was to calculate the start and end positions of each element on the web page when designing the layout:

> *If you want a div on a page that's 100 pixels wide and 100 pixels tall, [...] a reasonable point would be for 10 pixels from the left edge, 10 pixels from the top edge [...] That gives you a good placement on where you could put your other stuff on the page. But it failed at a lot of points because then people told their stuff wasn't lined up right [...] you still need have to have somebody spot check it.* – P6

As P6 explained, this still required spot-checking from a sighted person. It was also a mentally intense process that could not be scaled for complex websites. Screen readers with the right add-ons could announce the measurements in percentages but this again required mental calculations on the part of the participants. P6 described his work on the Developer Toolkit (DTK)[11], an NVDA add-on to support the visually impaired programming community in UI development.

The access challenges also depended on the kind of UI participants were developing. Many participants shared that one of the advantages of mobile UI development was that they could verify the output and interactions by installing the mobile app on their phones:

> *I can get an idea how big the button is relative to the window and the screen and I can get an idea where the edges of the buttons are. I think that's quite nice. You slide your finger across a touch screen and the moment you encounter the button, you hear its name [...] It makes it very easy to explore graphical layouts.* – P15

[11] https://addons.nvda-project.org/addons/developerToolkit.en.html

Thus, touchscreen interfaces alleviated the issues pertaining to verifying visual feedback. However, participants also observed that writing the code for mobile UI development had certain drawbacks. While web UIs could be programmed by writing HTML/CSS/JavaScript in text editors, mobile UIs often had to be developed within IDEs. As reported in prior work, IDEs are more complex software and present several accessibility challenges compared to text editors to programmers with visual impairments. For UI development, IDEs have features that are supposed to facilitate quick UI design (e.g. Android Studio's Layout Editor10[12], Visual Studio's Windows Forms Designer11[13]). Sighted programmers can use these features to drag and drop the widgets and prepare the visual design relatively quickly. The same features present significant mouse work for our participants, requiring them to write the entire UI *"by hand"* (P12):

> *The designers for user interfaces are not accessible [...] It doesn't read your controls, it doesn't review pixel presidency, it doesn't read you anything in the UI designer.* – P12

The challenges of calculating pixel positions and requiring spot-checking assistance were significantly alleviated when participants were provided detailed design documents. This enabled participants to work faster, as they could look up the measurement details in the documentation and program accordingly. However, the design documents needed to be prepared to be accessible on screen readers. Participants had to articulate to their team members how to prepare accessible documents (e.g. not relying on screenshots only) and include design details for all UI elements in textual format:

> *For instance, some designs that I would receive would have [...] dividers between different buttons [...] And if they weren't specified in text [...] I wouldn't be able to see them [...] they would follow my instructions on what would make my job faster.* – P1

Participants also reported spending time with the designers to understand the layout of the UI. They felt that their colleagues generally struggled to explain things verbally, a detail we reported in the context of pair programming and system architecture diagrams too. Thus, **participants felt the onus was on them to ask the right questions to understand the UI design, especially in the initial days of the project**:

> *When I put the question very precise one [...] They answer and they are eager to answer. But if I ask for example, can you give me an idea of the layout, why it is too general, and they used to say maybe much more than I need or maybe they miss some parts. It's to me, just to try to at the beginning, to ask very, very precise questions.* – P13

Accessibility challenges in UI development also shaped many participants' decisions to pursue programming that would require them to deal with the front end as little as possible:

> *It's easier that there are far fewer accessibility concerns with back end and as far as its employment goes, they have a need for it.* – P9

Participants who had specialized in front-end programming felt they faced significant challenges finding employment. Participants shared several instances of employers doubting their programming abilities and the credibility of their education:

> *I was more than qualified for some of these jobs, like web designer, or web developer one at a university. I went to this interview and it was a panel interview with the manager of the group and the whole entire team. [...] Well, in the interview, the manager of this group actually asked me how many web design classes were you exempted from?* – P6

[12] https://developer.android.com/studio/write/layout-editor
[13] https://docs.microsoft.com/en-us/visualstudio/designers/windows-forms-designer-overview?view=vs-2019

4.3 Social and Personal Implications

In the previous sections, we have discussed the challenges participants faced in collaboration and how they managed these challenges. In this section, we describe the impact of accessibility challenges on seeking accommodations and help.

Most participants, independent of the country they resided and worked in, hesitated to ask their employers to provide them with commercially available ATs like JAWS, ZoomText, and braille displays. Participants felt their employers would perceive it as an expensive request and felt uncomfortable asking their *"boss to spend so much money"* (P12). A few participants felt their request would be seen as *"excuses"* (P23). Given the challenges in finding employment, participants preferred to not emphasize lack of access as it may be misinterpreted as a lack of programming ability. Thus, participants preferred switching to free and open-source alternatives or using their personal licenses instead of asking employers to provide the resources they were proficient with.

As explained in the previous sections, **participants often established access by explaining their preferred work practices through one-on-one and informal conversations.** They gave small demonstrations on how they used ATs to *"show people [rather] than to tell them"* (P15) about potential breakdowns. They felt such interactions were better at familiarizing colleagues with ATs, their workflows, and changing misperceptions about their programming ability. It also made their colleagues more open to adopting their preferred strategies:

> *So what I try to do is try to teach as I go because I find that if you break things up, it doesn't seem as daunting and then the more they get to know what your style is, little by little, it happens naturally, you know.* – P11

While informally educating collaborators made them more amenable to changing their work practices, it was also a slow and laborious process. Participants shared that explaining certain concepts verbally was *"tough"* (P19) and they had to *"remember the virtue of trying to be patient with people"* (P11). Participants also felt that not everyone was open to changing their perceptions about them, in which case they had to advocate more strongly for themselves or avoid such colleagues:

> *You kind of have to take a step back and you say, "Hey, don't disregard me, I know what I'm doing" and sometimes people listen, sometimes they don't! I try to avoid the people that don't listen and work with the people that do.* – P19

Participants shared that advocacy and collaboration were easier when their team had employed a person with a disability previously. In this case, sighted employees had some experience of working in a mixed-ability context and participants did not have to put in additional work in educating their colleagues or requesting access. They found that sighted colleagues were more comfortable modifying their practices to cater to them:

> *I think I sort of had an easy time getting into the workplace because they already had experience with a blind employee.* – P15

Participants also spoke about their experiences in organizations that had policies in place with regard to inclusion and accessibility. Participants felt more comfortable in requesting accessible alternatives and voicing their concerns:

> *At [Company X][14] I didn't have to do that because you have a big team and then they already know what is inclusion, and what is accessibility. It was a place where I could say that, "Okay this is not something accessible to me so why don't you help me with this or why don't you delegate it to someone else?"* – P23

[14]We substituted P23's organization's name to preserve anonymity. It is a large international software company.

P23 spoke more positively about her experiences in her current organization. She could propose to her team that she be assigned programming tasks that allow her to play to her strengths without worrying about misperceptions about her competence or ability. Her previous organization did not have any accessibility-related policies in place. In addition, P23 had faced significant hurdles in finding employment and the previous organization was her first employer. Her status as a junior employee made her position precarious. She felt voicing her concerns about the lack of accommodations and accessibility would draw attention to her disability. Thus, she preferred to work harder in addressing the accessibility challenges and avoided advocating for accessibility or seeking sighted assistance.

Participants' perceptions of the workplace also impacted how they sought help. This was to a large degree shaped by social and technical factors. For instance, in small teams, participants would work with the same group of people on all projects. Therefore, participants were familiar with everyone and felt comfortable reaching out to their *"supportive group of coworkers"* (P21). Many participants reported a positive working experience when internal tools like code-review systems and internal websites were accessible:

> *[Company Y][15] has a whole accessibility team. They're mostly located in retail accessibility, but they provide advice and consulting for all the other teams. Generally, it seems like they make an effort to make all their websites and internal tools accessible as best they can.* – P18

Accessible internal tools enhanced participants' work experience in three ways. First, they enabled participants to work more efficiently. Second, participants had to only occasionally seek help and only with *"minor things like clicking the combo box"* (P18). They did not have to worry about incurring social debt by wasting their colleagues' time. Such quick and infrequent acts did not necessarily draw attention to participants' disability. It was instead understood as a shortcoming of the software. Third, it suggested to them that the organization was committed to providing an accessible work environment. The presence of an accessibility team meant that there was recourse against more serious challenges in internal software and the organization was also likely to fix them. It also externalized their problems and did not necessarily make them unique to them.

By contrast, without accessible tools, participants had to seek assistance on a more regular basis. Participants felt the act of seeking assistance did not emphasize inaccessibility as much as draw attention to their disability. Participants also could not easily reciprocate the help due to lack of ATs on colleagues' computers, as discussed in section 4.2.3. Participants also worried about their colleagues feeling obligated to help them—they did not want their colleagues to feel that *"one of their responsibilities is to help"* (P17). To avoid this, participants would try to reach out to different colleagues every time. A few participants shared they would spend time finding coworkers who would be more willing to spend time answering their questions:

> *There used to be a lady in the cube just right across mine. She was very nice! She left a while ago. And there is nobody very close by that I feel really comfortable with.* – P10

The above quotes show decisions around help-seeking are shaped by sociotechnical considerations. Participants' experiences are shaped largely by their team—as shown by the contradicting experiences of P18 and P23, who had worked in the same organization but within different teams. This demonstrates the degree to which each participant's experiences are socially situated.

Advocacy and help-seeking in the context of programming complicated participants' sense of independence. Participants felt they could *"influence how things are done"* (P19) by advising their team on building accessible software. They would share their personal experiences and technical

[15]We substituted P18's current organization's name to preserve anonymity. It is a large international software company.

expertise to improve the accessibility and user experience of software for end users with disabilities. By doing so, they were able to advocate for accessibility in the software development process and also teach their colleagues about accessibility. P12 shared how his input regarding accessibility on a client-based project proved to be critical to the success of the project. Such instances enhanced the participants' sense of independence. At the same time, seeking assistance for challenges, however small and infrequent, impinged on their sense of independence. This resulted in *relative accessibility* of programming that contributed to a *relative sense of independence*:

> *You kinda run into this weird thing of partly empowering because computers are everywhere [...] and you are one of the people that knows more about them than most [...] at the same time, you are far less able in a lot of ways because you can't access the same diagrams and tools [...] So it's a weird mix of more independent because I can do more on computers than a lot but less so, because at the same time I can't do as much.* – P3

5 DISCUSSION

Our findings show that programmers with visual impairments use a complex ecosystem of tools. This ecosystem includes software related to programming, project management and communication, and internal corporate tools. Each of these is critical to the core task of programming and often must be used concurrently. The accessibility challenges in the ecosystem affect collaboration and help-seeking practices in mixed-ability contexts. Programmers with visual impairments and their sighted colleagues co-create new work practices in order to collaborate effectively. The practices are also shaped by characteristics of the team, advocacy, and additional work on the part of programmers with visual impairments. Based on our analysis, we have framed our discussion around (1) accessibility of group work, focusing on real-time collaboration and help interactions among colleagues, and (2) implications for collaborative programming that can serve researchers, designers, and employers.

5.1 Accessibility of Group Work

5.1.1 The burden of additional work. As we found in our study, several programming-related workflows (including pair programming, UI development, and system design) rely on visual artifacts and were inaccessible to participants as a result. Nonetheless, our participants found unique workarounds to circumvent the challenges. For example, in synchronous programming, participants and their colleagues used communication software to do a remote screen share and inform each other of their whereabouts in the codebase by announcing line numbers. This enabled them to work on their respective computers and access each other's programming contributions without having to change their AT settings. Finding such workarounds is *invisible work* [74]; it is necessary but falls outside the purview of formal definitions of work for our participants. By highlighting this work, we bring to fore the otherwise invisible work done by people with visual impairments in creating and maintaining access [16]. The invisible work is not limited to finding workarounds to circumvent inaccessibility. It includes other activities, also not included within formal definitions of work, such as information-seeking on mailing lists, identifying the right colleague to seek assistance from, and educating colleagues about improving the accessibility of software in the development process.

Furthermore, to perform their roles in the various programming workflows, especially in the context of collaborative tasks, participants had to articulate their own ways of working in the first place. They used informal demonstrations and one-on-one meetings with team members to communicate their strategies. Das *et al.* similarly found that people with visual impairments engaged in conversations with their collaborators to change work practices [31]. Through these

interactions, participants conveyed their preferred methods for pair programming, code styling, communication, and more. Generally, the articulation for access needs happened outside the context of programming-related tasks and, as characterized by participants, was a slow and repetitive process. Again, this goes to show that access is not inherent in the workplace or programming workflows. **It is the *articulation work*—the work to make work possible—performed by people with visual impairments that leads to the creation of access and modifies the established arrangements around work practices [28].** The articulation work remains invisible and is central to achieving collaborations in mixed-ability contexts. Our findings further showed that the nature of articulation work was contingent on the workplace and participants' perceptions of it. Participants were more at ease advocating for their needs and had to do less articulation in workplaces that had previously hired people with visual impairments or that seemed to prioritize inclusiveness and accessibility. In less-accommodating workplaces, participants had to perform emotional labor as they tried to be patient in explaining their workflows to their colleagues.

5.1.2 Fostering better interactions around help-seeking and help-giving. We saw instances of people seeking help from colleagues to circumvent challenges with technologies and activities that relied on visual artifacts. Similar to prior work in workplace contexts [23, 31], we found that the nature of the relationship between our participants and their colleagues affected when and how the former sought help. For instance, in smaller teams, participants shared a good professional relationship with most colleagues and felt comfortable seeking assistance with accessibility challenges. Consistent with prior work, our participants expressed concerns about incurring social debt [21, 83, 88], and they were concerned about the impact help-seeking would have on their sense of independence [84]. However, we also observed that decisions around seeking assistance were based on participants' perceptions of the accessibility of the work environment. For example, participants were more at ease in seeking assistance from colleagues when they felt their workplace made efforts to provide accessible internal tools and accessibility support. In such cases, the act of help-seeking was minor, quick, and infrequent. It was not likely to foreground the person's disability or result in colleagues spending too much time assisting the person with a visual impairment. When seeking help with minor challenges, participants also had to perform less work in explaining the issue to their sighted colleague. This was also evident from participants' preference for using mailing lists primarily composed of programmers with visual impairments to seek information about the accessibility of programming tools. They preferred emailing on these lists instead of posting on large programming websites like Stack Overflow where most members were unaware of the workflows of visually impaired programmers. Here, the desire to avoid the work associated with explaining concepts like accessibility and screen readers, which were inherently understood by people on accessibility mailing lists, guided participants' decisions.

Beyond help-seeking, there can be challenges for programmers with visual impairments in *giving* help. Help-giving relies on employees' "sense of citizenship" [13] and is motivated by "principles of reciprocity" in the workplace [40]. In collaborative activities like pair programming and software design, this reciprocity is inherently present—the expectation is that programmers seek assistance and provide help to their collaborators. For our participants, the expectation to provide help was amplified by their concerns about incurring social debt when seeking help for accessibility challenges. Assisting colleagues with their problems provided participants the opportunity to return the favors, meet workplace expectations, and also mitigate some of their concerns about help-seeking costs. However, the help-giving process for our participants was complicated due to various factors. These include reasons like unavailability of ATs on colleagues' computers, lack of AT licenses, work involved in setting up ATs, etc. Thus, participants and their colleagues have to adopt workarounds to provide help, especially when immediate or real-time

assistance is needed. Further, help-giving can also be an important way for our participants to establish competence through everyday actions and interactions [37], thereby conveying their abilities [39, 46]. Shinohara and Wobbrock have touched upon how the use of ATs to provide help boosts self-efficacy and self-confidence among people with disabilities [71]. In the context of programming, help-giving is closely intertwined with other processes like feedback and information provisioning. For instance, collaborative activities like pair programming require programmers to brainstorm solutions collectively. The inability to provide help and support in these activities may not only have implications for their competence and confidence but also pigeonhole them in specific roles. For example, it may mean that programmers with visual impairments are only doing code writing and not making recommendations toward code improvement during collaboration.

We recommend that designers use the Design for Social Accessibility (DSA) framework when designing to facilitate group work. This framework emphasizes to designers that ATs should be a vehicle to convey the end user's ability and identity in social settings [71]. This is done by considering both functional and social factors of AT use [69]. We argue that designers should specifically use methods to foreground interactions around help in professional contexts, especially help-giving by people with visual impairments. For instance, we noted how participants worked around the problem of a lack of ATs to assist their colleagues. They would install the trial version of ATs on colleagues' computers or use their license to install multiple versions on different computers. While these workarounds allowed synchronous assistance, they necessitated extra work on the part of the programmers with visual impairments. Additionally, they required colleagues' consent and were further complicated by legal limits on installations. Therefore, when designing ATs and collaborative tools, we recommend considering the time and work required by these workarounds as well as their impact on real-time help-giving and collaboration.

5.2 Implications for Collaborative Programming

Prior studies on the accessibility of programming have been limited in their scope. They have studied the experiences of programmers with visual impairments removed from group-work settings, which require carrying out multiple collaborative activities. **Our empirical contributions serve as a generative site for thinking about accessibility in collaborative programming.** We discuss some of the design implications in this section, situating them in the perspectives recommended for designing for disability [16, 17, 69].

In mixed-ability contexts, programming is a sociotechnical achievement. Programmers with visual impairments carry out a series of social and technical interactions to address accessibility challenges—using creative code-writing strategies, articulating their workflows, advocating for their access needs, and more. Designers should consider and foster these interactions and build on the workarounds that programmers with visual impairments have identified [17]. Designers should take into account the factors that shape the choice of programming tools such as project complexity, workplace requirements, and concurrent use with other tools in the ecosystem. We also strongly recommend examining the *setup process*, as well as maintaining accessibility across software updates [31]. This requires ensuring the accessibility of activities in the installation process such as account creation, assessing the tool's accessibility, and finding the appropriate installer.

Current code-styling standards are largely intended to improve code navigation and readability for sighted programmers. However, in our study we report on the emergence of a new set of practices that were beneficial for our participants as well as their sighted colleagues. Some of the rules, like writing modular code and frequent documentation, were useful to everyone. On the other hand, visually focused practices (e.g., indenting code segments, using inline spaces, or placing braces on different lines) did not necessarily help programmers with visual impairments but they adopted them in their collaboration with sighted programmers. We also noted that participants' strategies

(such as adding descriptive comments, using camel case for names, and long variable names) were incorporated by their colleagues. We recommend that code-styling standards, especially when shared online by large software companies like Google[16], should also advocate for the adoption of strategies preferred by programmers with visual impairments and thereby present a more inclusive document. This would inform sighted programmers about the code-writing preferences of programmers with visual impairments and reduce the work of communication for the latter. It would also improve the efficiency of tasks associated with code reading and writing on computers with and without ATs in mixed-ability contexts. Additionally, an inclusive set of standards can lead to more efficient collaboration in the code-reviewing activity. It would also affirm the organization's commitment to accessibility, resulting in a more positive experience for programmers with visual impairments.

In UI development, participants had to expend mental effort in calculating the pixel position of elements when design documents were high level. Participants also reported that ATs lacked relevant information and UI development tools were largely inaccessible. They had to seek sighted assistance frequently to verify the placement and aesthetics as they were developing the UI. As in the context of homes [22], repeated assistance with things like spot-checking may be minor but can add up. Participants preferred help that was minor and infrequent and allowed them to independently carry out the majority of the work. A lack of accessible tools also had implications for participants' employment opportunities and careers. It prevented participants from contributing to front-end development and made them choose other sub-domains within programming like backend programming or data management. This speaks to the relative accessibility of programming—it is more accessible than other STEM fields but domains within it remain relatively inaccessible. This again motivates thinking about the accessibility of UI development tools using the DSA framework to convey programmers' ability and competence at developing UIs [69]. For instance, one participant explained that some IDEs had relatively accessible UI tools but these were replaced with inaccessible options in later versions. Another participant was working on developing an NVDA add-on to support his peers. Such tools can serve as starting points to brainstorm about improving the accessibility of front-end development tools. They also provide opportunities to engage programmers with visual impairments in the design process as "designing bodies" [17].

Crowd-supported solutions like VizWiz [19], BeMyEyes[17], and AIRA[18] are recommended alternatives to sighted assistance from personal and professional networks. Past research has shown that people with visual impairments prefer using these networks because they offer quicker and more contextual help without leading to social costs [21, 50]. Thus, assistance for certain programming activities like spot-checking, assessing the UI, and accessibility challenges in setting up the programming environment can be outsourced to these services. Given their familiarity with ATs, they may be better suited to provide assistance than workplace IT support. They are also likely to reduce the extra work that programmers with visual impairments have to perform in explaining the accessibility challenges to sighted people. However, usage of these services is likely to be regulated by an employing organization's policies around intellectual property, as there is a risk of disclosure of internal ideas and artifacts. This warrants thinking about formal integration of these services in the workplace to support programmers with visual impairments.

[16] http://google.github.io/styleguide/
[17] https://www.bemyeyes.com/
[18] https://aira.io/

5.3 Limitations and Future Work

Despite our best efforts to have a more balanced gender representation, most of our participants were men (18 of 22). This was possibly due to two reasons. First, most of our participants were recruited online (17 of 22) and online communities are predominantly male [80]. Second, the field of programming is heavily skewed towards men [45], and disabled people marginalized on the basis of their gender face further barriers to participation in computing fields [20]. We are also aware that our participant base skews towards young programmers. This may again be because most participants were recruited through online channels. In future work, we would focus on understanding the perspectives of gender-based minorities and older adults in programming.

Our participants hail from various countries. We are aware that cultural and legal differences persist in the workplaces of different countries and this is likely to shape our participants' experiences. To compare and contrast the findings, we would need a larger sample of participants from each of the countries. In future work, we intend to address this by recruiting more participants and analyzing the data taking into account the legal, educational, and cultural landscapes.

Our study findings rely on the self-reported data gathered from programmers with visual impairments, and we do not have the perspectives of their sighted colleagues. We therefore cannot speak to how sighted programmers feel about changing work practices. In future work, we would conduct interviews with and collect observations from our participants' sighted colleagues to uncover micro-interactions pertaining to collaborative activities in mixed-ability contexts.

6 CONCLUSION

Work at the intersection of accessibility, HCI, and programming tends to examine people with visual impairments and their interaction with a single category of tools [5, 6, 63, 64]. However, our study suggests that, in a collaborative environment, programmers with visual impairments use an ecosystem of tools to accomplish their tasks. They have to access internal resources such as databases and virtual machines, acquire the information on responsibilities assigned to them from project management software like JIRA and Microsoft Teams, and use communication software to coordinate collaborative programming activities. In this light, we echo the findings of Das *et al.*, who also find that people with visual impairments use multiple ATs and word processors in work environments to collaboratively write with their colleagues [31]. Similarly, our study highlights the need for access studies in HCI to be broader in their examination of programmers' interactions with tools to collaborate with their colleagues.

ACKNOWLEDGMENTS

This study would not have been possible without our participants. We thank them for sharing their experiences and insights with us. We also thank Anubha Singh, Robin Brewer, and Silvia Lindtner for their time and feedback at various points in this research. The work was supported by a gift from Google.

A DETAILS OF INTERVIEW PARTICIPANTS

Table 1. **Participant Information**
*Note: "Did not share" refers to participants not describing their visual ability at any time during the interview. We interpret this as their decision to not foreground their disability in the interviews.

#	Age	Gender	Self-described Visual Ability	Self-described Prog. Experience (in years)	Prog. Languages	Prog. Editors	Organization
P1	29	M	Vision loss from retinitis pigmentosa in early 20s	7	Java	Visual Studio	Freelancer
P2	26	M	Did not share	1-2	HTML, CSS, PHP, Python, Java	Notepad, Eclipse occasionally	NGO
P3	30	M	Blind since birth	11-12	Python, SQL, PHP, JavaScript	Notepad++	Sports Company
P4[19]	45	M	Gradual vision loss from retinitis pigmentosa	20+	.NET, JavaScript, HTML, CSS	Different text editors	Software Startup
P5	24	M	Blind since birth	3-4	Java, Python	IntelliJ, Visual Studio	IT Company
P6	45	M	Blind since 1 year old	10+	Python, HTML, CSS, PHP, Java	Visual Studio, VS Code	Freelancer
P7	32	F	Legally blind with corrected vision 20/200	3	Python, Java, HTML, CSS, JavaScript	VS Code	Healthcare Company
P8	27	M	Blind since 6 years old	4	Python, JavaScript	Visual Studio	IoT Startup

Continued on next page

Table 1 – continued from previous page

#	Age	Gender	Self-described Visual Ability	Self-described Prog. Experience (in years)	Prog. Languages	Prog. Editors	Organization
P9	39	M	Blind since birth	20	Python, Go, Perl, SQL	Vim	U.S. State Government ITS
P10	52	M	Lost total vision in an accident at 50	24	Python, JavaScript (Node.js)	VS Code	Telecommunications Company
P11	39	F	Did not share	2	HTML, CSS, JavaScript	Native Text Editor	U.S. State Government ITS
P12	28	M	Blind since birth	5	C#, Python, Java	Visual Studio	Digital Software Agency
P13	41	M	Blind since birth	15	HTML, CSS, Java, JavaScript, Python	Eclipse preferred, Occasionally Notepad++	Software Startup and University
P14	32	M	Blind since birth	19	C#, Android, PHP	Visual Studio	Freelancer
P15	29	M	Blind since birth	13	C, Go, Python, Haskell	Emacs	University and Independent Research Organization
P16	73	M	Vision loss from retinitis pigmentosa in late 30s	30	COBOL	Organization's internal text editor	Retired from Bank
P17	50	M	Vision loss from retinitis pigmentosa in early 20s	26	Visual FoxPro	Visual FoxPro	Healthcare Company

Continued on next page

Table 1 – continued from previous page

#	Age	Gender	Self-described Visual Ability	Self-described Prog. Experience (in years)	Prog. Languages	Prog. Editors	Organization
P18	39	M	Blind since birth	4	JavaScript, Python	Emacs	Large International Software Company
P19	30	M	Blind since birth	4-5	Go	Different text editors, avoids IDEs	Big Data Analytics Company
P20	55	F	Did not share	20+, scattered programming experience	Python, C, ChucK	Notepad++	University
P21	35	M	Blind since birth	16-17	C#, SQL	Visual Studio	Advertising Agency
P22	33	M	Vision loss from macular degeneration in early 20s	10	HTML, PHP, Python, JavaScript, AutoHotkey	Notepad++	University
P23	27	F	Vision loss from retinitis pigmentosa in mid-teens	7	.NET, Java, PHP, HTML, CSS	Eclipse for Java, Python, Visual Studio for .NET, Notepad for PHP	Large International Software Company

REFERENCES

[1] [n.d.]. ([n. d.]).

[2] 2019. Blindness Statistics. https://www.nfb.org/resources/blindness-statistics

[3] 2019. Stack Overflow Developer Survey Results 2019. https://insights.stackoverflow.com/survey/2019

[4] Ali Abdolrahmani, William Easley, Michele Williams, Stacy Branham, and Amy Hurst. 2017. Embracing Errors: Examining How Context of Use Impacts Blind Individuals' Acceptance of Navigation Aid Errors. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (Denver, Colorado, USA) (CHI '17). Association for Computing Machinery, New York, NY, USA, 4158–4169. https://doi.org/10.1145/3025453.3025528

[5] Khaled Albusays and Stephanie Ludi. 2016. Eliciting Programming Challenges Faced by Developers with Visual Impairments: Exploratory Study. (2016). https://doi.org/10.1145/2897586.2897616

[6] Khaled Albusays, Stephanie Ludi, and Matt Huenerfauth. 2017. Interviews and Observation of Blind Software Developers at Work to Understand Code Navigation Challenges. (2017). https://doi.org/10.1145/3132525.3132550

[7] Steve Alexander. 1998. Blind programmers face an uncertain future. ComputerWorld 32, 44 (1998), 86–87.

[19]Participant's data excluded from the analysis

[8] Paul R Amato and Julie Saunders. 1985. The perceived dimensions of help-seeking episodes. Social Psychology Quarterly (1985), 130–138.

[9] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In 2013 35th International Conference on Software Engineering (ICSE). IEEE, 712–721.

[10] Catherine M Baker, Cynthia L Bennett, and Richard E Ladner. 2019. Educational Experiences of Blind Programmers. (2019). https://doi.org/10.1145/3287324.3287410

[11] 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. (2015). https://doi.org/10.1145/2702123.2702589

[12] Mark S Baldwin, Sen H Hirano, Jennifer Mankoff, and Gillian R Hayes. 2019. Design in the Public Square: Supporting Assistive Technology Design Through Public Mixed-Ability Cooperation. Proceedings of the ACM on Human-Computer Interaction 3, CSCW (2019), 1–22.

[13] Peter Bamberger. 2009. Employee help-seeking: Antecedents, consequences and new insights for future research. Research in personnel and human resources management 28, 1 (2009), 49–98.

[14] Andrew Begel. 2008. Effecting change: Coordination in large-scale software development. In Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering. 17–20.

[15] Edward C Bell and Natalia M Mino. 2015. Employment outcomes for blind and visually impaired adults. (2015).

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

[17] Cynthia L Bennett and Daniela K Rosner. 2019. The Promise of Empathy: Design, Disability, and Knowing the" Other". In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. 1–13.

[18] Cynthia L Bennett, Daniela K Rosner, and Alex S Taylor. 2020. The Care Work of Access. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. 1–15.

[19] Jeffrey P Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, et al. 2010. VizWiz: nearly real-time answers to visual questions. In Proceedings of the 23nd annual ACM symposium on User interface software and technology. 333–342.

[20] Brianna Blaser, Cynthia Bennett, Richard E. Ladner, Sheryl E. Burgstahler, and Jennifer Mankoff. 2019. Perspectives of Women with Disabilities in Computing. Cambridge University Press, 159–182. https://doi.org/10.1017/9781108609081.010

[21] Erin L. Brady, Yu Zhong, Meredith Ringel Morris, and Jeffrey P. Bigham. 2013. Investigating the appropriateness of social network question asking as a resource for blind users. In Proceedings of the 2013 conference on Computer supported cooperative work - CSCW '13. ACM Press, New York, New York, USA, 1225. https://doi.org/10.1145/2441776.2441915

[22] Stacy M. Branham and Shaun K. Kane. 2015. Collaborative Accessibility: How Blind and Sighted Companions Co-Create Accessible Home Spaces. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (Seoul, Republic of Korea) (CHI '15). ACM, New York, NY, USA, 2373–2382. https://doi.org/10.1145/2702123.2702511

[23] Stacy M. Branham and Shaun K. Kane. 2015. The Invisible Work of Accessibility. In Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility - ASSETS '15. ACM Press, New York, New York, USA, 163–171. https://doi.org/10.1145/2700648.2809864

[24] Eric Brechner. 2003. Things they would not teach me of in college: what Microsoft developers learn later. In Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. 134–136.

[25] Robin N Brewer and Vaishnav Kameswaran. 2019. Understanding Trust, Transportation, and Accessibility through Ridesharing. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. 1–11.

[26] Sallyann Bryant, Pablo Romero, and Benedict du Boulay. 2008. Pair programming and the mysterious role of the navigator. International Journal of Human-Computer Studies 66, 7 (2008), 519–529.

[27] Mary Elaine Califf, Mary Goodwin, and Jake Brownell. 2008. Helping Him See: Guiding a Visually Impaired Student through the Computer Science Curriculum.

[28] Juliet M Corbin and Anselm L Strauss. 1993. The articulation of work through interaction. The sociological quarterly 34, 1 (1993), 71–83.

[29] Nicola Cornally and Geraldine McCarthy. 2011. Help-seeking behaviour: A concept analysis. International journal of nursing practice 17, 3 (2011), 280–288.

[30] D Crary. 2008. Employer bias thwarts many blind workers. Associated Press (2008).

[31] Maitraye Das, Darren Gergle, and Anne Marie Piper. 2019. "It doesn't win you friends" Understanding Accessibility in Collaborative Writing for People with Vision Impairments. Proceedings of the ACM on Human-Computer Interaction 3, CSCW (2019), 1–26.

[32] Guy Dewsbury, Karen Clarke, Dave Randall, Mark Rouncefield, and Ian Sommerville. 2004. The anti-social model of disability. Disability & society 19, 2 (2004), 145–158.

[33] Carolyn D Egelman, Emerson Murphy-Hill, Elizabeth Kammer, Maggie Morrow Hodges, Collin Green, Ciera Jaspan, and James Lin. 2020. Pushback: Characterizing and Detecting Negative Interpersonal Interactions in Code Review. (2020).

[34] Hongfei Fan, Jiayao Gao, Hongming Zhu, Qin Liu, Yang Shi, and Chengzheng Sun. 2017. Balancing Conflict Prevention and Concurrent Work in Real-Time Collaborative Programming. In Proceedings of the 12th Chinese Conference on Computer Supported Cooperative Work and Social Computing. 217–220.

[35] Joan M Francioni and Ann C Smith. 2002. Computer science accessibility for students with visual disabilities. In Proceedings of the 33rd SIGCSE technical symposium on Computer science education. 91–95.

[36] Vinitha Gadiraju. 2019. BrailleBlocks: Braille Toys for Cross-Ability Collaboration. In The 21st International ACM SIGACCESS Conference on Computers and Accessibility. 688–690.

[37] H. Garfinkel. 1991. Studies in Ethnomethodology. Wiley. https://books.google.com/books?id=zj_leg8-tIEC

[38] David Garlan and Mary Shaw. 1993. An introduction to software architecture. In Advances in software engineering and knowledge engineering. World Scientific, 1–39.

[39] Erving Goffman et al. 1978. The presentation of self in everyday life. Harmondsworth London.

[40] Alvin W Gouldner. 1960. The norm of reciprocity: A preliminary statement. American sociological review (1960), 161–178.

[41] Nancy Gourash. 1978. Help-seeking: A review of the literature. American journal of community psychology 6, 5 (1978), 413.

[42] Rajesh Hegde and Prasun Dewan. 2008. Connecting programming environments to support ad-hoc collaboration. In 2008 23rd IEEE/ACM International Conference on Automated Software Engineering. IEEE, 178–187.

[43] Ric Holt. 2002. Software architecture as a shared mental model. Proceedings of the ASERC Workhop on Software Architecture, University of Alberta (2002), 64.

[44] Joe Hutchinson and Oussama Metatla. 2018. 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. ACM Press, New York, New York, USA, 1–6. https://doi.org/10.1145/3170427.3188696

[45] D. Izquierdo, N. Huesman, A. Serebrenik, and G. Robles. 2019. OpenStack Gender Diversity Report. IEEE Software 36, 1 (2019), 28–33.

[46] Vaishnav Kameswaran, Jatin Gupta, Joyojeet Pal, Sile O'Modhrain, Tiffany C Veinot, Robin Brewer, Aakanksha Parameshwar, and Jacki O'Neill. 2018. 'We can go anywhere' Understanding Independence through a Case Study of Ride-hailing Use by People with Visual Impairments in metropolitan India. Proceedings of the ACM on Human-Computer Interaction 2, CSCW (2018), 1–24.

[47] Niall Kennedy. 2006. Google Mondrian: web-based code review and storage. https://www.niallkennedy.com/blog/2006/11/google-mondrian.html

[48] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2016. The emerging role of data scientists on software development teams. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). IEEE, 96–107.

[49] Tien Fabrianti Kusumasari, Iping Supriana, Kridanto Surendro, and Husni Sastramihardja. 2011. Collaboration model of software development. In Proceedings of the 2011 International Conference on Electrical Engineering and Informatics. IEEE, 1–6.

[50] Reeti Mathur and Erin Brady. 2018. Mixed-Ability Collaboration for Accessible Photo Sharing. In Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility. 370–372.

[51] Sean Mealin and Emerson Murphy-Hill. 2012. An exploratory study of blind software developers. In Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC. 71–74. https://doi.org/10.1109/VLHCC.2012.6344485

[52] Oussama Metatla, Alison Oldfield, Taimur Ahmed, Antonis Vafeas, and Sunny Miglani. 2019. Voice user interfaces in schools: Co-designing for inclusion with visually-impaired and sighted pupils. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. 1–15.

[53] Richard J. Miara, Joyce A. Musselman, Juan A. Navarro, and Ben Shneiderman. 1983. Program Indentation and Comprehensibility. Commun. ACM 26, 11 (Nov. 1983), 861–867. https://doi.org/10.1145/182.358437

[54] Jonas Moll and Eva-Lotta Sallnäs Pysander. 2013. A haptic tool for group work on geometrical concepts engaging blind and sighted pupils. ACM Transactions on Accessible Computing (TACCESS) 4, 4 (2013), 1–37.

[55] Jonas Moll, Kerstin Severinson-Eklundh, and Eva-Lotta Sallnas. 2007. Group Work About Geometrical Concepts Among Blind and Sighted Pupils Using Haptic Interfaces. In 2007 2nd Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems. 330–335.

[56] Cecily Morrison, Edward Cutrell, Anupama Dhareshwar, Kevin Doherty, Anja Thieme, and Alex Taylor. 2017. Imagining Artificial Intelligence Applications with People with Visual Disabilities using Tactile Ideation. In Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility. 81–90.
[57] Arie Nadler, Shmuel Ellis, and Iris Bar. 2003. To seek or not to seek: The relationship between help seeking and job performance evaluations as moderated by task-relevant expertise. Journal of Applied Social Psychology 33, 1 (2003), 91–109.
[58] Mala D Naraine and Peter H Lindsay. 2011. Social inclusion of employees who are blind or low vision. Disability & Society 26, 4 (2011), 389–403.
[59] John T Nosek. 1998. The case for collaborative programming. Commun. ACM 41, 3 (1998), 105–108.
[60] Shotaro Omori and Ikuko Eguchi Yairi. 2013. Collaborative music application for visually impaired people with tangible objects on table. In Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility. 1–2.
[61] Steve Oney, Alan Lundgard, Rebecca Krosnick, Michael Nebeling, and Walter S Lasecki. 2018. Arboretum and arbility: Improving web accessibility through a shared browsing architecture. In Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology. 937–949.
[62] Dewayne E Perry, Nancy A. Staudenmayer, and Lawrence G Votta. 1994. People, organizations, and process improvement. IEEE Software 11, 4 (1994), 36–45.
[63] Vanessa Petrausch and Claudia Loitsch. 2017. Accessibility Analysis of the Eclipse IDE for Users with Visual Impairment. (2017). https://doi.org/10.3233/978-1-61499-798-6-922
[64] Venkatesh Potluri, Priyan Vaithilingam, Suresh Iyengar, Y Vidya, Manohar Swaminathan, and Gopal Srinivasa. 2018. Codetalk: Improving programming environment accessibility for visually impaired developers. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. 1–11.
[65] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice. 181–190.
[66] Johnny Saldaña. 2015. The coding manual for qualitative researchers. Sage.
[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. ACM Press, New York, New York, USA, 773–779. https://doi.org/10.1145/3287324.3287499
[68] Helen Sharp, Robert Biddle, Phil Gray, Lynn Miller, and Jeff Patton. 2006. Agile Development: Opportunity or Fad?. In CHI '06 Extended Abstracts on Human Factors in Computing Systems (Montréal, Québec, Canada) (CHI EA '06). Association for Computing Machinery, New York, NY, USA, 32–35. https://doi.org/10.1145/1125451.1125461
[69] Kristen Shinohara, Cynthia L. Bennett, Wanda Pratt, and Jacob O. Wobbrock. 2018. Tenets for Social Accessibility: Towards Humanizing Disabled People in Design. ACM Trans. Access. Comput. 11, 1, Article 6 (March 2018), 31 pages. https://doi.org/10.1145/3178855
[70] Kristen Shinohara and Jacob O Wobbrock. 2011. In the shadow of misperception: assistive technology use and social interactions. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. 705–714.
[71] Kristen Shinohara and Jacob O. Wobbrock. 2016. Self-Conscious or Self-Confident? A Diary Study Conceptualizing the Social Accessibility of Assistive Technology. ACM Trans. Access. Comput. 8, 2, Article 5 (Jan. 2016), 31 pages. https://doi.org/10.1145/2827857
[72] Robert M Siegfried. 2006. Visual Programming and the Blind : The Challenge and the Opportunity. Science Education (2006), 275–278. http://www.adelphi.edu/{~}siegfrir/molly
[73] Darja Šmite, Nils Brede Moe, and Richard Torkar. 2008. Pitfalls in remote team coordination: Lessons learned from a case study. In International Conference on Product Focused Software Process Improvement. Springer, 345–359.
[74] Susan Leigh Star and Anselm Strauss. 1999. Layers of Silence, Arenas of Voice: The Ecology of Visible and Invisible Work. (1999). https://link.springer.com/content/pdf/10.1023/A:1008651105359.pdf
[75] Andreas Stefik, Andrew Haywood, Shahzada Mansoor, Brock Dunda, and Daniel Garcia. 2009. Sodbeans. In 2009 IEEE 17th International Conference on Program Comprehension. IEEE, 293–294.
[76] Anja Thieme, Cynthia L. Bennett, Cecily Morrison, Edward Cutrell, and Alex S. Taylor. 2018. "I can do everything but see!" – How People with Vision Impairments Negotiate their Abilities in Social Contexts. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18. ACM Press, New York, New York, USA, 1–14. https://doi.org/10.1145/3173574.3173777
[77] 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 2017 Conference on Designing Interactive Systems. 739–752.
[78] Alexia Tsotsis. 2011. Meet Phabricator, The Witty Code Review Tool Built Inside Facebook. https://techcrunch.com/2011/08/07/oh-what-noble-scribe-hath-penned-these-words/

[79] Janine van der Rijt, Piet Van den Bossche, Margje WJ van de Wiel, Sven De Maeyer, Wim H Gijselaers, and Mien SR Segers. 2013. Asking for help: A relational perspective on help seeking in the workplace. Vocations and learning 6, 2 (2013), 259–279.

[80] Bogdan Vasilescu, Andrea Capiluppi, and Alexander Serebrenik. 2014. Gender, representation and online participation: A quantitative study. Interacting with Computers 26, 5 (2014), 488–511.

[81] Herman Wahidin, Jenny Waycott, and Steven Baker. 2018. The Challenges in Adopting Assistive Technologies in the Workplace for People with Visual Impairments. In Proceedings of the 30th Australian Conference on Computer-Human Interaction (Melbourne, Australia) (OzCHI '18). Association for Computing Machinery, New York, NY, USA, 432–442. https://doi.org/10.1145/3292147.3292175

[82] April Yi Wang, Anant Mittal, Christopher Brooks, and Steve Oney. 2019. How Data Scientists Use Computational Notebooks for Real-Time Collaboration. Proceedings of the ACM on Human-Computer Interaction 3, CSCW (2019), 1–30.

[83] Emily Q. Wang and Anne Marie Piper. 2018. Accessibility in Action: Co-Located Collaboration among Deaf and Hearing Professionals. Proc. ACM Hum.-Comput. Interact. 2, CSCW, Article Article 180 (Nov. 2018), 25 pages. https://doi.org/10.1145/3274449

[84] Michele A Williams, Caroline Galbraith, Shaun K Kane, and Amy Hurst. 2014. "Just let the cane hit it" how the blind and sighted see navigation differently. In Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility. 217–224.

[85] Judith D Wilson, Nathan Hoskin, and John T Nosek. 1993. The benefits of collaboration for student programmers. ACM SIGCSE Bulletin 25, 1 (1993), 160–164.

[86] Jacob O Wobbrock, Shaun K Kane, Krzysztof Z Gajos, Susumu Harada, and Jon Froehlich. 2011. Ability-based design: Concept, principles and examples. ACM Transactions on Accessible Computing (TACCESS) 3, 3 (2011), 1–27.

[87] Chien Wen Yuan, Benjamin V Hanrahan, Sooyeon Lee, Mary Beth Rosson, and John M Carroll. 2017. I Didn't Know That You Knew I Knew: Collaborative Shopping Practices Between People with Visual Impairment and People with Vision. Proceedings of the ACM on Human-Computer Interaction 1, CSCW (2017), 1–18.

[88] Yuhang Zhao, Shaomei Wu, Lindsay Reynolds, and Shiri Azenkot. 2017. The effect of computer-generated descriptions on photo-sharing experiences of people with visual impairments. Proceedings of the ACM on Human-Computer Interaction 1, CSCW (2017), 1–22.

[89] Annuska Zolyomi, Anushree Shukla, and Jaime Snyder. 2017. Technology-Mediated Sight: A Case Study of Early Adopters of a Low Vision Assistive Technology. In Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility (Baltimore, Maryland, USA) (ASSETS '17). Association for Computing Machinery, New York, NY, USA, 220–229. https://doi.org/10.1145/3132525.3132552

Received June 2020; revised October 2020; accepted December 2020