A Paradigm Shift in Nonvisual Programming

Venkatesh Potluri

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2024

Reading Committee:

Jennifer Mankoff, Chair

Jon E. Froehlich

Shaun Kane

Program Authorized to Offer Degree:

Computer Science & Engineering

University of Washington

Abstract

A Paradigm Shift in Nonvisual Programming

Venkatesh Potluri

Chair of the Supervisory Committee:
Jennifer Mankoff
Computer Science and Engineering

Programming and software engineering are of keen interest to the blind or visually impaired (BVI) community, spurring accessibility enhancements to programming tools that simplify nonvisual code navigation and debugging. Though these enhancements improve the general accessibility of software engineering, they fail to address accessibility of specialized programming domains, such as user interface design, physical computing and data science due to their reliance on visual code outputs. Consequently, these domains have become inaccessible to BVI developers. My dissertation supports the following thesis: Access to visual code outputs is critical for BVI experts to contribute expertise in high-skilled programming work. New interaction techniques, access to data representations, and data-driven studies are critical to make this visual information in widely used programming tools accessible. In this dissertation, I first present details on three efforts I undertook during my PhD: (1) UITap -- a prototype system that demonstrates a set of new interaction techniques that demonstrate approaches to make user interface development accessible, (2) PSST -- a data sonification toolkit that gives BVI developers the tools to access live data generated by sensors used in physical computing, and (3) a large scale analysis to examine the accessibility of computational notebooks ---popular environments used by data scientists. While these three threads of work contribute to the accessibility of three high skilled programming domains -- user interface

development, physical computing, and data science, the rapid adoption of AI assistance may be causing a shift in how we program. I conclude with recommendations drawn from BVI developer experiences, to ensure that this paradigm shift in programming remains accessible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to extend my gratitude to my advisor, Jennifer Mankoff, for her unwavering support throughout the course of my PhD. In addition to acquiring good research skills through her mentorship, I have also grown to be a compassionate researcher. My committee members, Jon Froehlich, Shaun Kane and Amy Ko have always given the space to reach out for advice, and continue to inspire me through their wisdom and research.

I am grateful to the Paul G. Allen School of Computer Science and Engineering for providing me with the environment, resources and opportunities to grow as a researcher. Several supports, ranging from the leadership's nomination for fellowships to the advising team's constant support, and the operations team's accessibility checks on the facilities have made academic, and academic adjacent aspects of graduate life enjoyable and productive. I would also like to extend my gratitude to Apple for recognizing me as an AI ML scholar. The work done by many organizations such as UW Create, Access Computing, and Access CS for All provide a strong foundation to support students with disabilities like me to thrive in higher education.

I can't thank my family enough for the innumerable ways in which they continue to show support. My Mom, Uma Potluri has not only been a pioneer in making life accessible; her continued excitement, energy, and positive outlook has motivated me during many low points of my research career. My sister, Suma Potluri has been the invisible force behind many accessibility failures. From the 2AM calls to visually verify resumes to being a listening ear about my research ideas, she has done it all! I must say my family has been as enthuziastic as me about research and my PhD journey and this mindset has had a profound impact.

My mentors and collaborators Andrew Begel, Dhruv Jain, Lilian de Greef, Megan Hofmann, Cynthia Bennett and Richard Ladner have supported me through many pivotal moments of my PhD journey. This journey may not have been possible, positive, or complete without your support. Thank you for being such amazing mentors! My collaborators Sudheesh Singanamalla, Mauli Pandey, Tadashi Grindeland, John Thomson, Bongshin Lee, Kate Glazko, Ather Sharif, Dhruv Jain, Avery Mack, Liang He, and many others have been onboard with my research ideas, and gave me an opportunity to be onboard theirs, and helped me grow as a researcher.

The past few years have been delightful with forging many new friendships, and continuing old ones. Life would have been so much more boring without them -- Sudheesh Singanamalla, Priyal Suneja, Pratyush Patel, Han Zhang, Kartik Sawhney, Matt Ziegler, Taylor Shenoni, Emma McDonnell, Kate Glazko, and members of the Make4all, makeability, and ICTD labs at the University of Washington to name a few. Thanks for all the support, snacks, and being test subjects for my experiments demonstrating my impeccable ability to crack jokes.

Visual information has been pivotal to several aspects of my PhD education, as is evidenced by the focus of this dissertation. Though access to consume and produce visual content has not been adequate in many occasions, several professional and personal supports have been critical in making slides, formatting documents, navigating unfamiliar physical and digital spaces. I thank the UW disability resources for students office for pioneering Aira access as a student accommodation. This has been incredibly valuable in several aspects of my graduate student life. I am grateful for support from Tadashi Grindeland, Jazz Ang, Varsha Konda, Dhruv Khanna, Ben Kosa and Melissa Hovik for taking on the professional position of a visual interpreter or a scribe, a position with constantly evolving job responsibilities and guidelines. Despite this support, several occasions called for last minute visual help. Many friends and colleagues stepped in at these moments in ways that asking for help felt effortless. Anant Mittal, Sudheesh Singanamalla, Priyal Suneja, and every

one of my colleagues who has integrated visual interpreting into our collaborations, you all are awesome. I will carry this support forward, and I aspire to conduct research that makes access to visual information the default.

Lastly, though I've always had a right to be educated, this did not turn into an opportunity due to ableist assumptions about a blind student pursuing a career in science. My deepest gratitude to Dr. D. Usha Reddy, and the leadership at IIIT-H for the opportunity to be educated during my formative years.

DEDICATION

My family for moving mountains for me,

and the late Mari Christine for giving me the first opportunity to be educated.

Chapter 1

INTRODUCTION

The ability to create software empowers us to positively impact global societal challenges, making it a widely sought-after skill opening numerous work, educational, and creative opportunities. For instance, the employment of data scientists is expected to grow by 30% in the next 10 years —a growth larger than most other fields (Bureau of Labor Statistics, U.S. Department of Labor, 2024). Additionally, the ability to create user interfaces equips developers to pursue specific employment opportunities as full-stack developers. These specialized skills such as data science and design equip experts to influence decision-making systems and create usable AI-powered experiences (Kaluarachchi, 2021). It is imperative that the systems we build to influence human experiences are informed by users, and it is critical that we recognize expertise from professionals who have been minoritized. This dissertation describes my work to create positive change through the design, development, and evaluation of programming tools used in expert programming domains to make these domains accessible to developers who are blind or visually impaired (BVI developers).

Programming and software engineering have been of interest to the blind and visually impaired (BVI) community since as early as 1980, when BVI engineers published guides to accessibly work with electronic circuits (Gerrey, 1980). BVI experts have made several contributions since, which include NVDA —the most widely used screen reader, programming environment extensions, and as data scientists, designers, and cyber-security analysts —to name a few essential roles they play in society. Visual impairment has been the largest represented disability among programmers with physical disabilities according to Stack Overflow's developer survey as of 2022.

Several advances in programming tools, professional practices, and creative problem-solving efforts led by BVI people have equipped BVI developers to participate in programming and software engineering. The accessibility of developer tools has been investigated by researchers and Integrated Development Environment (IDE) makers. The resulting accessibility enhancements to popular programming tools include the use of accessible UI components to surface information about code to facilitate easy screen reader navigation, as well as audio cues to simplify non-visual code debugging (Albusays and Ludi, 2016; Potluri et al., 2018, 2019a; Schanzer et al., 2019). Nevertheless, many other accessibility issues remain unexplored —a recent study addressing the challenges related to diversity in software engineering has little discussion about disability (Rodríguez-Pérez et al., 2021) and (Mountapmbeme et al., 2022), through a literature review, highlights the need to better investigate the experiences of BVI software developers in the real world.

Early efforts to explore accessibility challenges experienced by BVI developers include the work of (Mealin and Murphy-Hill, 2012), who conducted the first empirical study of accessible software development for BVI developers. They uncover challenges stemming from programming environments and those associated with performing specific types of specialized work, such as developing user interfaces. Several studies since, including my own, delved deeper into specific accessibility barriers BVI developers face from unexamined assumptions about visual ability in IDE design. These efforts address accessibility barries in code navigation and debugging, the use of command line interfaces, and information seeking when performing software engineering tasks (Potluri et al., 2018; Sampath et al., 2021; Storer et al., 2021).

Though these studies contribute to improving the accessibility of software engineering, they do not cater to nuanced accessibility gaps that specialized programming domains might present. Through enhancements to programming tools, proof-of-concept prototypes, and large scale data driven studies, my work makes user interface design, physical computing, and data science —three popular domains that have been thus far overlooked —more open to input from BVI experts.

1.1 Thesis Statement

My work uses the aforementioned specialized programming domains as examples to support the thesis: *Access to visual code outputs is critical for BVI experts to contribute expertise in high-skilled programming work. New interaction techniques, access to data representations, and data-driven studies are critical to make this visual information in widely used programming tools accessible.*

In establishing this thesis, my work answers the overarching research question: *How can we approach the design of both domain-specific and general-purpose development environments to make specialized programming domains accessible to BVI developers?*

To answer this research question, I first describe my work that examines BVI technology users' understanding of visual semantics, such as the size, shape, and arrangement of elements in a user interface, and present details of a prototype system, UITap, that builds on this knowledge, demonstrating multi-modal (touch screen combined with an IDE) interactions that make web design accessible to BVI developers. UITap's interaction techniques and feedback mechanisms make visual aesthetic information accessible to BVI web developers.

In addition to user interfaces, another commonly occurring visual modality in programming addresses how data is represented. Programming tools and interfaces enable the creation of easy to understand data visualizations that are typically not accessible to BVI developers. To begin addressing the accessibility gaps that data visualizations can cause in programming, I next summarize my work to design the first data sonification toolkit, called PSST, that helps BVI developers make sense of real-time data generated by sensors, such as accelerometers; PSST is one of the very few toolkits that support BVI customization of data sonification.

Other specialized programming domains that are affected by inaccessible data representations are data science and machine learning. I describe my data-driven investigation to examine the accessibility of computational notebooks —a popular medium used in data science and machine learning to communicate code, data, and narratives informed by the data.

While this work may have caused a shift in how we think about accessible programming, and who we include in specialized skilled programming disciplines, the rapid adoption of AI-assisted developer tools has caused a shift in how we program. To equip us to create programming experiences that are born-accessible, I conclude with opportunities and challenges grounded in the discourse of BVI programmers over the past ten years.

1.2 Contributions

My work detailed in this dissertation broadly contributes new interaction techniques to accessible programming, tools to access visual information and code outputs, and demonstrates approaches to develop a data-driven baseline understanding of accessibility. My efforts carefully consider the readiness of widely used programming tools to support the accessibility enhancements needed for BVI developers to use them. These considerations have resulted in a combination of research prototypes to demonstrate a novel interaction technique, open source code repositories, and datasets to support deeper investigations. I contributed:

UITap: a prototype system that demonstrates a touch plus keyboard developer experience to make web design accessible to BVI developers.

PSST: a toolkit that provides tools for BVI developers to customize their own accessible representations of streaming data. PSST supports scenarios that can assist with understanding sensors, but its functionality can be extended to other scenarios that involve data.

NotebookA11y: a dataset of a hundred thousand randomly chosen set of Jupyter notebooks, accessibility scans, and the code and pipeline to reproduce our accessibility analysis on computational notebooks (Potluri et al., 2023).

1.3 Authorship and Positionality

The work described in this dissertation is a collective effort of many amazing collaborators who not only contributed their expertise but also worked to change their work practices to increase accessibility of the research conducted. This is of key importance for my work, as the specialized programming domains often were inaccessible for me to understand and contribute to as a blind researcher. To acknowledge this collective effort, I refer to the contributors of this work as "we", and switch to using "I" when providing perspectives drawn from my own reflections or when guiding the reader through this dissertation.

A researcher's personal background and experiences can influence research they conduct. As a blind person assuming several roles, i.e., student, researcher, and software developer, it is important to examine the effect of these identities on my research interest and agenda.

I am a congenitally blind Indian male who migrated to the United States to pursue a PhD focused on accessibility and human-computer interaction. Pursuing a career path in computer science meant that I experienced challenges arising from assumptions about BVI capabilities, which surface in attitudes towards inclusion, educational opportunities, and infrastructures to perform my duties as a computer scientist. Throughout my career, I helped spread awareness about the harm caused by these assumptions through public speaking engagements, contributions to auto-ethnographic accounts as a disabled graduate student, and mentorship to other disabled students. Equally important to my raising awareness are my efforts to systematically identify accessibility gaps I encounter and provide solutions to address them, so that I can perform my work to the best of my ability and learn new skills. However, I am confident that the intellectual merit of my work stands on its own.

I restrict the impact of my disability identity on research efforts by limiting disclosure to instances that impact research methodology and practice. Specifically, the study setup for my qualitative efforts to examine BVI users' understanding of visual interfaces used a visual interpreter. In documenting the design of PSST, I briefly reflect on the scaffolding needed to make toolkit

development non-visually accessible. My experiences trying to understand how sensors work using a screen reader motivated some of the demos we developed to validate the toolkit. Finally, in my work to analyze computational notebooks, noticing that notebooks were crashing when I used them with a screen reader, led to an analysis of whether these crashes were due to notebook size.

The impact of my identity on my research has been evolving over the past few years. I expect this growth to continually evolve as I pursue future research opportunities.

1.4 Organization

In this dissertation, I first provide some background on BVI programming in chapter 2. In §2.1, I provide a snapshot of the tools and infrastructures used by BVI programmers. In §2.2, I describe relevant literature on increasing accessibility of developer tools and programming practices. In §2.3, I describe efforts to make data visualizations accessible. This dissertation is an amalgamation of threads of my work that focus on specialized programming domains that are very different from one another. To assist with reading and contextualizing these different threads, I will provide relevant background across chapters that describe each of these specialized programming domains.

In chapter 3, I describe my work to make the first of three specialized programming domains, user interface (UI) development, accessible to BVI developers. First, in §3.1, I provide background on the state of accessibility of design. In §3.2, I then provide details on a qualitative study with ten BVI computer users conducted with the goal of examining the role that visual semantics (attributes of a UI such as the size and shape of interface elements) play in screen reader use. These observations set the foundation for UITap (§3.4). I close with a discussion of future directions to make real-world developer tools for UI design accessible §3.5.

The next two chapters of this dissertation highlight my work focused on the accessibility of programming experiences and specialized domains that involve data. My work looks at programatically generated streaming and static data representations.

In chapter 4, I describe my work to make the second of the three specialized programming domains, sensor programming, accessible. I first provide background on the importance of understanding data to debug sensors, and physical computing §4.1. I then describe the design of the Physical computing Streaming Sensor data Toolkit (PSST) §4.2, and close with future directions towards the real-world adoption of PSST in §4.5.

In chapter 5, I examine the accessibility of computational notebooks, widely used in data science and machine learning (the third programming domain that will be discussed in this dissertation). I describe the analysis pipeline (§5.2), results (§5.3), and provide actionable recommendations to make notebooks accessible (§5.4). I close with an impact statement detailing the current state of accessibility of notebooks.

My discussion in chapter 6 draws upon the discourse that BVI developers have been having about programming tools and careers; providing an accessibility landscape of open challenges. I close with possible future directions to usher in an accessible future of AI-assisted programming.

Chapter 2

BACKGROUND

Opportunities to pursue careers that require programming skills are constantly evolving along with the tools that are used to code. For example, being a full-stack developer requires that developers are familiar both with back-end engineering *e.g.,* database configuration and front-end engineering *e.g.,* development using HTML and CSS. Similarly, data analysts and scientists often create easy to understand representations of data and combine them with textual narratives. Additionally, the skills needed to build human-centered machine learning systems encompass the need to understand the algorithms that drive the decisions made by these systems and good UIs and experiences for the systems to be usable by everybody (Kaluarachchi, 2021).

In this chapter, I first establish BVI expertise in programming. I provide a brief overview of the accessibility technology and developer tools used by BVI developers. I then provide a comprehensive background on accessible programming. I conclude with background on the accessibility of data, and highlight gaps in making specialized programming domains accessible.

2.1 BVI Programmers

People who are blind or visually impaired often use a variety of assistive technologies to access computers. One type of assistive technologies, *Screen Readers*, such as JAWS for Windows (Freedom Scientific), NVDA (NV Access) and VoiceOver (Apple), render information about what is happening on a computer, tablet, or mobile phone screen through voice, sound effects, and/or Braille. Another type of assistive technology, *Screen Magnifiers*, used by people with partial usable

vision, enlarge the pixels displayed on the screen, making content and graphics more readable and recognizable (American Foundation for the Blind, 2021). My work described in this dissertation focuses on issues that pertain to accessibility using screen readers. Understanding how screen readers work could help appreciate the contributions made in this dissertation.

2.1.1 Screen Reading Technology

Screen readers convert content into a linear, ephemeral stream of audio or Braille output. they convey structured information, such as the interactor hierarchy of an interface, tabular information, such as the title and contents of a table, and semantics, such as non-speech audio representations of content type, using beeps, tones, or spoken identification of a heading level. For example, VoiceOver (Apple) and NVDA (NV Access) convey interactor hierarchy by presenting the UI as a list of interactable elements as well as through a separate review cursor that uses the underlying accessibility tree exposed by Windows APIs, respectively.

Screen reader user interactions predominantly involve linearly navigating user interfaces and content with a keyboard on desktop computers, and through touch gestures on a touch screen smartphone. Screen readers also support some degree of non-linear access to semantic content. They offer functionality for users to jump through structural elements like headings and links, and specific types of information such as graphics and tables. For example, these tools allow non-linear access to tables by providing navigation capabilities horizontally (along the row) or vertically (along the column). Furthermore, they provide announcements that help users identify reference points in a table, *e.g.,* row and column headers if they are semantically marked up (Web Accessibility Initiative, 1999, 2019). Despite these features, navigation is at times problematic and cumbersome for screen reader users. Additionally, user expertise also impacts which features are used during non-visual access. for example, studies of beginning users highlight challenges with understanding visual context, such as when the cursor has switched to a new application (Baldwin et al., 2017).

Unlike screen readers built for desktop use that are designed with keyboard input in mind, touch screen based screen readers are inherently more spatial because they are designed for direct rather than indirect interaction (*i.e.,* touch *vs* keyboard). Both Apple's VoiceOver and Google's TalkBack (Google), widely used screen readers on touch screen smartphones, enable screen reader users to directly access elements spatially under their touch and to interact indirectly using swipe and tap gestures at any location on the screen. An example feature that supports directly accessing UI elements in VoiceOver and TalkBack is invoked when a user places or drags their finger on the screen. Both screen readers announce the label and any additional accessibility metadata provided by UI developers when a user's finger moves over UI elements. Features that support indirect interaction with UI elements in TalkBack and VoiceOver enable users to swipe anywhere on screen to cycle through elements and double tap to perform the primary action of the selected element.

In seminal early work, Kane et al. (2008) presented design recommendations and gestures for non-visual touch interactions, many of which can be seen in mainstream screen readers today. In follow-up work, Kane et al. (2011b) performed a gesture elicitation study with both sighted and blind participants to co-design usable gesture sets. Their findings categorize touchscreen interaction techniques into menu browsing, discrete gestures, and fixed regions.

They further inform quantifiable measures of gestures, differentiating between sighted and BLV individuals. Their work expands this knowledge to larger touchscreens and demonstrates three access overlays: edge projection, neighborhood browsing, and touch-and-speak (Kane et al., 2011a). Screen reading on desktop and mobile computing environments have been refined over the years to provide an efficient, accessible, and usable computing experience to BVI users. These efficiency gains however do not translate to programming experiences that could benefit from them.

Several efforts attempted to understand screen reader use to improve information access to BVI users on the web (Bigham et al., 2007, 2008; Borodin et al., 2010). This work highlighted the complexity of information retrieval for screen reader users on the web, and pointed out differences

between web browsing by BVI and sighted users. Bigham et al. (2009) present a tool that helps create annotated instructions for task completion on webpages. Similarly, Sallnäs (2006) surfaces the communication difficulty between BLV and sighted users in virtual interfaces. They focus particularly on deictic directions used in commands, which proved problematic to BLV users. Though screen readers provide access to the web, their emphasis on web semantics to convey information makes communication with sighted users complicated due to the inherent emphasis on visual aesthetics when visually parsing a webpage (Lindgaard et al., 2006).

Efforts have focused on specific types of information that screen reader users may encounter. (Haider and Yesilada, 2020) highlight the difficulty with table navigation, informing that tables are misused for styling, and data tables found on the web violate accessibility guidelines. Wang et al. (2022) consolidate guidelines to make data tables accessible and contribute a browser extension that makes them accessible by fixing errors, summarization, and splitting tables with multiple headings into smaller ones. Complementary to improving navigation of tables, table navigation has also been explored as a means to efficient screen reader navigation. To improve information access, Khurana et al. (2018) explore non-linear interaction techniques on the keyboard to enable BLV users to navigate tables and webpages spatially. They show improved task completion times in information-seeking tasks by BLV users in a lab study. Williams et al. (2019) investigate advantages of representing web interfaces as tables and find that participants rated their table navigation experience more positively in terms of effort, memorization, ease of navigation, understanding of page information, and confidence in submitted answers. The work summarized here highlights the strengths of tables—to navigate data and layouts, and informs that tables found on the web may be inaccessible at large.

Several efforts focused on increasing access to graphical content to screen reader users. Efforts have primarily focused on making images accessible through rich interactive representations (Bigham et al., 2010; Morris et al., 2018; Low et al., 2019; Pareddy et al., 2019). Morris et al. (2018)

explore several rich representations of image content to support interactive exploration of images. Prior work has also explored the nuance of information to be included in image descriptions, which often gets surfaced to screen reader users through the alt attribute in HTML. Bennett et al. (2021) recommend care when describing race and identity information of people in pictures. Similarly, Mack et al. (2021) synthesize guidelines to describe images, and surface them in Microsoft's PowerPoint. Generating a meaningful image description is dependent on the context, content, and the representation of who is in the image and who the description is for.

In summary, the work described here introduces the state of desktop and smartphone screen reading, highlighting their differences. Additionally, we describe prior efforts to increase access to tabular and graphical information—both of which are critical to accessible programming. The state of screen reader accessibility summarized here is critical to understand how BVI developers program, which I will now describe.

2.2 Accessible Programming

There is very limited information on the representation of BVI developers, with the most recent statistic indicating that they were the largest group of developers with physical disability represented in the Stack Overflow developer survey 2021 and 2022. Stack Overflow has since stopped collecting this demographic data, so we do not have an updated statistic of representation. Additionally, there is very little easily understandable information on BVI practices when participating in programming and software development.

2.2.1 BVI Developer Workflow

BVI developers have shared first-person accounts of programming (Beijers, 2019; Dishman, 2020) and videos of demonstrations in which they write code (Shaikh, 2018). These first-person narratives and videos emphasize that *BVI developers can and do* code. They also offer readers an understanding

of the various tools used in the process. A BVI developer's development tool set generally consists of a standard computer, accessible integrated development environments (IDEs), a set of back-up editors, a command line interface, and screen readers. Developers often use these tools in various combinations based on the task, infrastructure available, and accessibility of the editor (Branham and Kane, 2015; Pandey et al., 2021).

The evidence highlighted here establishes that BVI developers *can and do* code, using widely available programming tools. They identify creative approaches to combine these tools to overcome accessibility barriers that software development practices and tools may present.

2.2.2 Accessibility of Programming

Several efforts, often lead by BVI experts, increased access to programming. Seminal work by Raman (1996) presented audio highlighting—a concept that uses voice modulation in synthesized speech to convey formatting in text and programmatic concepts in a tool called EmacSpeak. Researchers and developer tool makers alike have since worked to improve access to the features of an IDE, and focused on information access needed to debug, navigate, and comprehend code to learn programming and perform professional work (Stefik et al., 2007; Baker et al., 2015; Potluri et al., 2018). Mealin and Murphy-Hill (2012) conducted a comprehensive study to understand the accessibility barriers experienced by BVI developers and found challenges associated with using IDEs and developing user interfaces. Other studies to learn about IDE use have uncovered challenges to efficiently navigate, debug, and glance through code (Albusays et al., 2017; Potluri et al., 2018; Pandey et al., 2024).

Code navigation has been a suboptimal user experience for BVI developers due to the presentation of code in IDEs in ways that are suitable for visual consumption. Albusays et al. (2017), through interviews and in-situ task observations, identified that visual abstractions used in IDEs can cause accessibility barriers. BVI developers of the study noted that the navigation strategies they

used, though effective, often reduced their efficiency to navigate code. Improvements to the user interfaces used in developer tools, however, have often improved code navigation for screen reader users. CodeTalk and StructJumper, Visual Studio extensions, have explored the use of tree and list views to convey code structure (Baker et al., 2015; Potluri et al., 2018). Schanzer et al. (2019) explore the use of abstract syntax trees to support code navigation for beginner programmers. More recently, Grid-Coding—a set of interaction techniques—explore table-like structures to present code with a goal to support editing tasks (Ehtesham-Ul-Haque et al., 2022). Alternative interfaces and representations of code such as the use of list and tree views, and audio cues to indicate folded code, integrated into developer environments such as Visual Studio have improved code navigation and glanceability for screen reader users.

Another programming task that relies heavily on visual affordances of tools is the debugging of code. Programming environments often surface syntax errors and warnings through visual decorators and highlights. Debuggers, tools intended to simplify problem discovery, present the necessary information through interfaces optimized for visual consumption. Auditory and speech feedback have been explored as possible ways of surfacing the information needed for a developer to debug code. For example, the Wicked Audio Debugger uses audio feedback to convey state variables and program execution (Stefik et al., 2007). Similarly, CodeTalk extends the use of audio feedback by introducing the concept of TalkPoints, auditory breakpoints that can be configured by users to understand code execution paths, or values of specific variables during program runs (Potluri et al., 2018). Such auditory feedback about program state and execution paths increase the accessibility of information that an expert needs to identify issues.

These interfaces are however optimized to convey numerical and textual outputs as program state. Though they are powerful in supporting the debugging of most general purpose code, the information they provide may not be sufficient to debug code that may produce large amounts of data, or code that may produce visual outputs. Understanding the characteristics of data such

as trends and basic statistics may be essential to debug code that produces and processes data. Similarly, code that generates graphical user interfaces may produce visual outputs with visual bugs, *i.e.,* inconsistencies in the visual appearance of a graphical user interface (Issa et al., 2012; Macklon et al., 2023). Future efforts to improved developer tool accessibility could (1) add support for visual bug identification, and (2) provide affordances to understand data.

A software developer's typical workflow involves performing tasks peripheral to working with the code directly. Developers use additional tools, refer to documentation, and collaborate with their colleagues on synchronous and asynchronous programming tasks (Nagappan et al., 2003; Bacchelli and Bird, 2013; Socha and Sutanto, 2015; Kubelka, 2017). Recent work addresses accessibility gaps experienced by BVI developers when performing these essential tasks critical to be a successful software engineer. Sampath et al. (2021) enumerate the difficulties of working with command line interfaces, critical tools used alongside IDEs to perform software development tasks. They note that unstructured textual representation in these interfaces cause user frustrations when using them. BVI developers also experience accessibility barriers when browsing online documentations or question-answer websites (Storer et al., 2021), and seek help from other BVI developers (Johnson et al., 2022). Apart from the interactions with specific tools, software engineering also involves working closely with other developers in co-located tasks such as code walkthroughs and reviews. These tasks often cause accessibility gaps for BVI developers due to the use of visual communications such as screen shares and visual indicators (Pandey et al., 2021). Potluri et al. (2022a) address the access to collaborator awareness information, *i.e.,* their location, movements and activity, through integrations to the VS Code LiveShare extension (Ghost, 2020; Microsoft, 2022).

In summary, the work highlighted here, though important, focuses on generalizable programming tasks. It does not account for the nuanced accessibility needs that have to be met for specialized programming domains to be accessible to BVI developers.

2.3 Accessible Data

Access to tools that help us understand, communicate insights, and take critical decisions driven by large volumes of data is essential to perform a host of specialized programming tasks as data scientists and creators of human-centered machine learning systems (Davenport and Patil, 2012; Kaluarachchi, 2021; Zhang et al., 2022; Bureau of Labor Statistics, U.S. Department of Labor, 2024). Data for these professional programming tasks is often represented in structured formats such as tables, and visual representations—a modality that can cause significant accessibility barriers if not thought through carefully. Work detailed in section 2.1.1 describes table navigation with screen readers. These techniques expand to data tables (Wang et al., 2022).

Both consumption and authoring of data visualizations are under-supported for BVI people. In Kim et al. (2021)'s survey of 56 accessible visualization papers published between 1999 and 2020, we learn that the majority were intended for use in static contexts. Eleven supported the creation of a chart that is accessible to BVI users, but only two papers supported interactive authoring of charts by BVI users. Interaction with streaming data was not mentioned at all. Discoverability and comprehensibility for BVI users is not supported by many online visualizations (Sharif et al., 2021). Improved screen reader interactions, voice commands, sonification, multi-modal representations with touch screens and tactile graphics have been explored to improve access to data and data visualizations (Gorlewicz et al., 2020; Sharif et al., 2021, 2022; Apple, 2022; Holloway et al., 2022; Siu et al., 2022). Sharif et al. (2022) present a JavaScript plugin that makes two-dimensional data accessible to screen reader users. The plugin supports both speech based summaries and sonification to convey trends in the data, and can be used along with a screen reader. Zong et al. (2022) focus specifically on screen reader interactions of charts and inform that improvements to structural organization, navigation, and descriptions are necessary to improved screen readability of visualizations. Though not sufficient to make data visualizations accessible, they find that accompanying visualizations with tables is crucial for accessibility due to the familiarity of tables

to screen reader users. Complementary to interactivity, guidelines have emerged to provide rich descriptions of data visualizations. Lundgard and Satyanarayan (2021) contributed a foundational 4-level semantic model to provide natural language descriptions of data visualizations. They suggest the descriptions of statistical concepts and visual encoding, and note that BVI consumers reported the addition of context behind a visualization to be less important. The VisText dataset and benchmark operationalizes these guidelines through fine-tuning a large language model (Tang et al., 2023). Elavsky et al. (2022) combine this work, and contribute Chartability guidelines, a set of guidelines inspired by accessible visualization efforts. The work highlighted here offers great promise to make online data visualizations accessible in ways that would integrate with existing accessibility technology. This work however does not explore the needs of data power users—consumers and creators of data who may want to make decisions or manipulate it.

A few approaches combined descriptions with sonification to describe data visualizations and infographics (Holloway et al., 2022; Sharif et al., 2022; Siu et al., 2022). When such sonification is available, its impact is powerful, as demonstrated by the SonicX system (Garcia et al., 2019), designed for collaborative use by sighted and Blind astronomers (Merced, 2013). The system led to new scientific discoveries (Merced, 2021). Sonification has been identified as an important technique for monitoring streaming data in various domains including finance, programming, business process monitoring, system administration, human activity data visualization, and accessibility (Janata and Childs, 2004; Roginska et al., 2006; Hussein et al., 2009; Nees and Walker, 2009; Hildebrandt and Rinderle-Ma, 2013; Cherston and Paradiso, 2017). Some important facets of monitoring identified in the literature include alerting and understanding both individual streams of data and the overall data landscape (Roginska et al., 2006). A variety of techniques for sonification have been studied over many decades, ranging from orchestration to specialization (Roginska et al., 2006). The field is sufficiently mature that sonification books (*e.g.,* Hermann et al. (2011)) and design guides (*e.g.,* De Campo (2007); Dubus and Bresin (2011); Nees (2019); Enge et al. (2021)) have been published.

Several authors have suggested basic capabilities a data exploration system should support, whether sonification-based or not. As summarized by (Beilharz and Ferguson, 2009), these include the ability to parse, filter, mine, represent, refine, and interact with data (Fry, 2004), as well as the ability to analyze trends, detect patterns, and estimate and compare points (Walker and Nees, 2005).

These sonification techniques have been codified in sonification toolkits (Madhyastha, 1992; Lodha et al., 1996, 1997), and sonification authoring languages (Lodha et al., 1996; Atherton and Blikstein, 2017; Phillips and Cabrera, 2019; Trayford and Harrison, 2023). Common capabilities provided by these tools include analyzing and transforming data by rescaling and quantization, filtering, inversion, and normalization. These toolkits are also capable of mapping this transformed data onto sonification parameters, including rhythm, articulation, pitch, and timbre. These capabilities are provided through Turing-complete visual or textual languages, opening the door to an endless range of data transformation and mapping possibilities. However, these tools are not designed for BVI people, and often emphasize ease of use for sighted non-programmers. Both of these trends mean that a number of sonification toolkits are structured around inaccessible audiovisual interactions either at the time when sonifications are designed, when they are used, or both (*e.g.,* Lodha et al. (1996); Cherston and Paradiso (2017); Phillips and Cabrera (2019)). For example, Rotator (Cherston and Paradiso, 2017) expects sonification to be configured in a GUI and employs D3 visualizations, which are generally not accessible (Sharif et al., 2021). Similarly, AeSon (Beilharz and Ferguson, 2009) provides both a visual language and a GUI for authoring sonifications and SIFT (Bruce and Palmer, 2005) is a block-based language (such languages are generally not designed to be accessible to BVI people, (Ludi, 2015).

Despite the need to understand real-time data and the advantage of sonification to do so, there is a dearth of accessible sonification tools that enable BVI developers to customize accessible representations of real-time data.

2.4 Background Summary

Efforts to improve the accessibility of programming environments augmented screen reader interactions with auditory feedback and repurposed accessible UI controls to support a wide range of programming tasks. This work, though very critical in setting great standards to provide an accessible environment for BVI expert programmers, does not account for nuances of programming with visual information and data, essential for BVI experts to participate as experts in specialized programming domains to build the systems and make decisions that would shape the future. Additionally, efforts to make data accessible to BVI people did not consider them as experts, and the few that did do not account for a great deal of customizability to listen to real-time, live, streaming data. Finally, due to the inherent inaccessibility of specialized programming tasks, understanding nuanced needs to make these programming domains accessible is challenging despite there being a need to understand accessible in professional work in the real world (Huff et al., 2020). My dissertation bridges these accessibility gaps and contributes new interaction techniques, data access tools, and data driven accessibility studies to make visual code outputs accessible to position BVI experts as equal contributors to high-skilled work.

Chapter 3

UITAP: ACCESSIBLE WEB DESIGN

The ability to design a robust and intuitive user interface (UI) lets developers build accessible, usable, and useful applications, websites, and digital experiences. Careful choices about color, shape, size, and placement of UI elements, all critical features of a visually appealing and aesthetic interface, affect first impressions of an app or website's utility, user perceptions of website security, and feature discovery (Lindgaard et al., 2006).

Designers and developers typically engage in a myriad of UI design and prototyping activities, including sketching, constructing wireframes, and hi-fi prototype development (Landay and Myers, 1995) to carefully design UIs. They use UI builders, design tools such as Figma, and styling languages such as Cascading Style Sheets (CSS), to identify a usable and appealing visual design. Additionally, they use software that supports What You See Is What You Get (WYSIWYG) interaction and feedback mechanisms, traditional programming environments, and website builders to design and develop user interfaces.

The tools and processes for UI interface design, prototyping and development, however, assume visual abilities that are inaccessible to BVI people (Bennett et al., 2016; Li et al., 2019; Kearney-Volpe and Hurst, 2021; Li et al., 2021). Li et al. (2021) assessed the conformance of high fidelity prototyping tools to accessibility guidelines and found that no prototyping tool was fully accessible with popular screen readers. These tools had inaccessible UI controls, making critical tasks such as arranging elements on a canvas inaccessible. Kearney-Volpe and Hurst (2021) found that the lack of alternative representations to visual information, inaccessible developer tools, and feature limitations

made web development inaccessible. These limitations cause BVI developers to disengage with web and UI design, cede control to sighted collaborators, and lose opportunities to pursue employment.

Our work attempts to address this accessibility gap in developer tools to support accessible web design. To this end, we ask the following research questions.

- **RQ1**. When, how, and why do BVI technology users interact with the visual semantics and aesthetics of user interfaces?

- **RQ2**. How can we use BVI understanding of user interface aesthetics or visual semantics to support accessible editing of user interfaces?

3.1 Background on Accessible Design

Recent efforts attempted to understand BLV engagement in design through workshops exploring ideation (Bennett et al., 2016), toolkits to facilitate BLV participation in design (Bennett, 2018), and by re-imagining co-design with BLV people (Brewer, 2018). Novel interaction techniques and interfaces have been investigated to support the independent creation of visual layouts and tactile artifacts like 3D models and graphics. For example, *ShapeCad* uses a 2.5D display to enable BLV users to engage with 3d design—a domain complementary to the programming of visual layouts that relies on visual affordances (Siu et al., 2019). Li *et al.* conducted a formative study of seven BVI users to explore how they engage in layout design (*e.g.,* to make slides in PowerPoint and create websites with WordPress), and, incorporated their findings into an accessible layout design tool that uses auto-generated tactile sheets overlaid on a tablet to enable the editing of HTML templates (Li et al., 2019). (Li et al., 2022)designed TangibleGrid, a system that uses a baseboard, and resizable brackets to facilitate the creation of web layouts by BVI developers.

Though this body of work addresses the important issue of engaging BLV users in design, it does not investigate BVI developers' understanding of user interfaces and their aesthetics (information

often omitted by screen readers that use control types and semantics). Nor does it address the impact their understanding can have on how to build accessible developer tools that support user interface design. We conduct a formative study to investigate BVI understanding of user interfaces and use this knowledge to develop a touch plus keyboard developer experience to make the editing of webpage layouts accessible.

3.2 Study: Examining Visual Semantic Understanding

To learn how BVI users conceptualize UI design and assess their skills/knowledge about visual semantics such as the size, shape, color, and spatial arrangement of UI elements, we performed a three-part qualitative study.

The study consisted of semi-structured interviews, screen reader tasks, and lo-fi prototyping by BVI users using tactile materials like Wikki-Stix and Playdough. We then used these findings to design a multi-modal coding experience that helps BVI developers edit webpage designs. In this section, I briefly describe the qualitative study and summarize its findings; section 3.4 then describes our prototype system that demonstrates this novel developer experience.

3.2.1 Method

Participants: We recruited ten BVI participants (four women; six men) through email, social media, and snowball sampling. As summarized in table 3.1, participants were on average 35 years old (*SD*=11.18, range 24-58) and were compensated with $15 per hour, resulting in a total compensation of $22 for participating in our 90-minute study.

Procedure: The study took 90 minutes to complete and included three parts: (1) a semi-structured formative interview, (2) smartphone and desktop-based screen reader tasks, and (3) UI reconstructions using lo-fi prototyping. Study sessions were conducted by two researchers (one facilitator, one note taker) and video recorded.

Table 3.1: Demographic and technology use details of the participants. Note that P7-LV and P10-LV are both low vision which is indicated in their participant ID since it may be relevant to interpreting quotes from these participants. *Light* refers to light perception; *Prog* refers to programming experience (this data is missing for P1).

ID	Age	Gen	Level of Vision	Light	Frequently Used Access Tech	Occasionally Used Access Tech	Prog?	Web Presence
P1	30	F	Totally blind	No	JAWS, VoiceOver on OS X, VoiceOver on iOS	NVDA		None
P2	27	M	Totally blind	No	NVDA, Braille Display, VoiceOver on iOS	JAWS, Windows Narrator	Yes	Personal website
P3	28	F	Totally blind	No	JAWS, VoiceOver on iOS	NVDA, Braille Display	No	Personal website
P4	35	M	Visual acuity not measurable	Light	JAWS, VoiceOver on OS X, Braille Display, VoiceOver on iOS	NVDA	No	None
P5	24	M	Totally blind	No	JAWS, VoiceOver on iOS	NVDA	Yes	Personal website
P6	58	M	Extremely low	Light	JAWS, Braille Display, VoiceOver on iOS	Windows Narrator	No	None
P7-LV	49	F	Double and blurry vision	Light	JAWS, Screen Magnification, VoiceOver on iOS, ZoomText	NVDA	No	Personal blog
P8	27	M	Totally blind	Light	VoiceOver on OS X, Braille Display, VoiceOver on iOS	JAWS, Windows Narrator, NVDA	No	None
P9	26	F	Light perception	Light	JAWS, NVDA, TalkBack on Android	None	Yes	Personal blog
P10-LV	46	M	Peripheral; no center vision	Yes	VoiceOver on OS X, Screen Magnification, VoiceOver on iOS, CCTV	None	No	None

Part 1: Formative Interview. We began with a semi-structured interview to examine how BLV users think about, make sense of, and use visual semantics in everyday interactions on smartphones and desktops (*i.e.,* with traditional GUIs). We also asked about the perceived importance and motivations for learning such semantics.

Part 2: Screen Reader Tasks. To investigate how BLV users rely on visual semantics in their UI interactions and to understand usage across contexts, we asked participants to perform common tasks using *smartphone apps*, *smartphone webpages*, and *desktop webpages*. for each device platform and usage scenario, we requested that participants choose a task that they perform every day using a screen reader, such as sending a text message, requesting a ride share, or planning a route. We also asked participants to perform two prescribed tasks: *adding a contact* (smartphone app only) and *searching for a video in YouTube* (all three contexts). During the tasks, we asked them to 'think aloud,' and we directly observed their interactions (*e.g.,* use of gestures and spatial patterns on the smartphone). for these interactions, participants could use screen readers or voice assistants (*e.g.,* Siri). Importantly, these tasks were intended as a probe to evoke reactions with regards to visual semantics in everyday computer and smartphone use and not meant to derive specific findings related to task completion—something that has been explored in prior work on web browsing (Bigham et al., 2007; Vigo and Harper, 2013), programming (Albusays et al., 2017), and digital visual layout creation (Li et al., 2019).

Part 3: UI Reconstructions. finally, to better understand how BLV users interpret and build mental models of visual semantics, we asked participants to reconstruct a UI of their choice for each context using lo-fi prototyping materials. Specifically, participants built UI reconstructions using pre-prepared poster board templates (cut in the size of a phone and laptop computer) along with Play-Doh, Wikki-Stix, and Braille labels. We also asked them to reconstruct their own websites (for those that had them, N=4).

Data Analysis: We collected interview session video recordings, researcher notes, and participant-created prototypes. Video recordings were manually transcribed by the research team and analyzed using an iterative thematic coding approach with a mixture of inductive and deductive codes (Braun and Clarke, 2006). The unit of analysis was a segment of video containing either a participant comment or an observational note made during initial transcription.

To begin, we created an initial codebook derived from our study protocol. For each unit, we recorded the context (*e.g.,* , the question that was asked and the location in the interview), timestamp, an initial code, and a participant identifier. To refine the codebook, six participants were randomly selected and re-coded by two researchers (three participants each, no overlap). The codebook was shared and mutually updated continuously. Using the updated codebook, 10% of the data was randomly selected and coded independently by the two researchers. To calculate inter-rater reliability, we used Cohen's Kappa, which resulted in κ=0.7. The researchers then met, resolved disagreements to consensus, and updated the codebook (where necessary). Finally, one researcher recoded all of the data using the updated codebook.

3.2.2 Study 1 Findings

We describe findings from our three-part qualitative study, including the perceived importance, understanding, and use of visual semantics across smartphone and desktop contexts. As a qualitative study, we are interested in capturing nuanced views and perspectives; however, we report numbers to indicate participant preferences and trends.

Importance of and Access to Visual Semantics

Towards addressing our initial research question about *when*, *how*, and *why* do BVI users interact with visual semantics, we report on perceived importance, touch *vs.* desktop interface use, and strategies and challenges to learning.

Despite not having full access to a UI's visual semantics, all participants felt that they were critically important, particularly to: (1) improve interaction with and navigation of UIs (*N*=6), (2) enable collaboration with sighted users (*N*=6), especially to provide and receive instruction, and (3) build their own visually oriented artifacts (*N*=2), such as websites and blogs.

for UI navigation, BVI users felt that visual semantics improved interaction efficiency, especially on smartphones. for example, when discussing their smartphone use, P8 said, *''for one thing, I know where some of the icons on the screen are. So, I can just touch the area where I believe it is.''* (P8). These spatial connections are enabled by the touchscreen screen reader, which offers direct access to widgets and information via touch. As an extreme case, P9 used only direct touch interactions on her phone: *''I only tap; I don't use gestures.''* (P9).

When collaborating with sighted individuals, BVI individuals often find themselves receiving and providing instructions about visual artifacts, *e.g.,* , how to find a button or feature in an application. This information exchange typically requires both sighted and BVI individuals to share an understanding of the overall UI as well as the current element in focus. P9 described the dual challenge of giving and receiving instructions with sighted users:

> *''It's two things. One, when I am trying to convey something to someone [sighted], and a lot of times the buttons that we hear are only labels, and it's not actually written [as a] home button [or] maybe a home icon. [. . .] And the second reason why I'd love to know how it's laid out is when other people are giving me directions. The other day I was using Microsoft Teams, and I wanted to figure out how to share my screen for a presentation. All my teammates said, 'Oh, it's like the red-colored button on the top right corner.' for me, there is no top-right corner.''* (P9)

The visual modalities used by sighted users and the semantic-based understanding of layouts of BVI users lead to inconsistencies that make receiving and providing instruction challenging. These challenges cascade into layout creation. Despite differences in interaction modalities between smartphone and desktop screen readers, visual semantic access was desired across contexts, *i.e.,* , participants learned about visual semantics in both smartphones and desktops, although to a varying degree. As described in section 2.1.1, smartphone screen readers support richer touch interaction compared to their desktop counterparts. Consequently, smartphones enable direct manipulation of UI elements, thereby facilitating access to its spatial arrangement.

Due to their inherently spatial nature and support for direct manipulation capabilities of smartphone screen readers, we observed that participants learned about spatiality and visual semantics predominantly on smartphones. However, learning took place on desktops as well, though was valued less. In contrast, desktop screen readers linearize UIs, which can obfuscate visual semantics. While four participants wanted better access to visual semantics on desktop UIs, five could identify no benefit to having them. *''[When] I am using a keyboard, I use the find feature quite frequently, and so it doesn't really matter if I know the spatial layout.''* (P3). This shows that P3 preferred seeking information without using visual semantics. In contrast, P4 describes the value of visual semantics for navigation when asked if he tried to understand the layout of the website when using a desktop: *''Yes. for the same reason. for navigation purposes. I may not use it in the same manner as when I am touching the screen, but it is important to know where the locations are if I need to get to it. Even navigation is quite different from touching the screen using the navigation keys on the keyboard vs. touching the screen.''* (P4). P4 thus describes how he prefers being aware of visual semantics, though they deviate from traditional keyboard-based navigation.

Lack of access to visual semantics may be an important reason why participants feel unmotivated to learn and understand spatial layout. *''I don't know much about spatial layout information. I don't have access about how to know spatial layout information.''* (P1).

Despite there being no standard method or access technology to teach visual semantics to BVI users, participants reported using several learning strategies, including trial and error, sighted assistance, training seminars, and features in existing screen reading technology, such as exploring by touch or listening to a hierarchical representation of a webpage. On smartphones, participants primarily depended on trial and error, using existing screen reading features such as touch exploration (*N=5*). On desktops, sighted assistance (*N=3*) and trial and error (*N=3*) were mentioned.

This learning about visual semantics on desktops is interesting given that desktop screen readers do not surface spatial arrangement of interfaces or provide visual semantic information. When describing trial-and-error strategies on the smartphone, participants talked about the ability to explore by touch as well as by leveraging common UI design paradigms (*e.g.,* tab bars at the bottom of iOS apps). *''When I download a new app, I touch parts of the screen where I expect certain things to be... I will tap on the bottom and see if there are tabs there... Once I start to understand the order of things, maybe I will touch and discover like 'oh that's near the top of the screen.'''* (P3). Participants clearly conveyed the value of these mental models. Interestingly, these function well even though participants were not overly accurate when asked to construct these models.

Learning about visual semantics through trial and error on desktop was also possible. For example, P5 describes how ... *''Screen readers themselves have given me [a] qualitative sense of what the layout looks like ... You would also know that this was a heading above that. This was a heading below that.''* (P5). P5 interprets the screen reader element order to imply something about a UI's visual semantics (*e.g.,* the relative position of elements). While participants learned spatial layouts on smartphones using trial and error and sighted assistance, they only used sighted assistance to learn desktop layouts. *''With touchscreen interfaces, you can actually get a sense of how it is laid out... if you touch, I know what is on the top-right corner, what is on the top-left corner... ''* (P9). We observed that BVI users gained knowledge of visual semantics of specific interfaces from sighted friends and colleagues when trying to navigate inaccessible layouts.

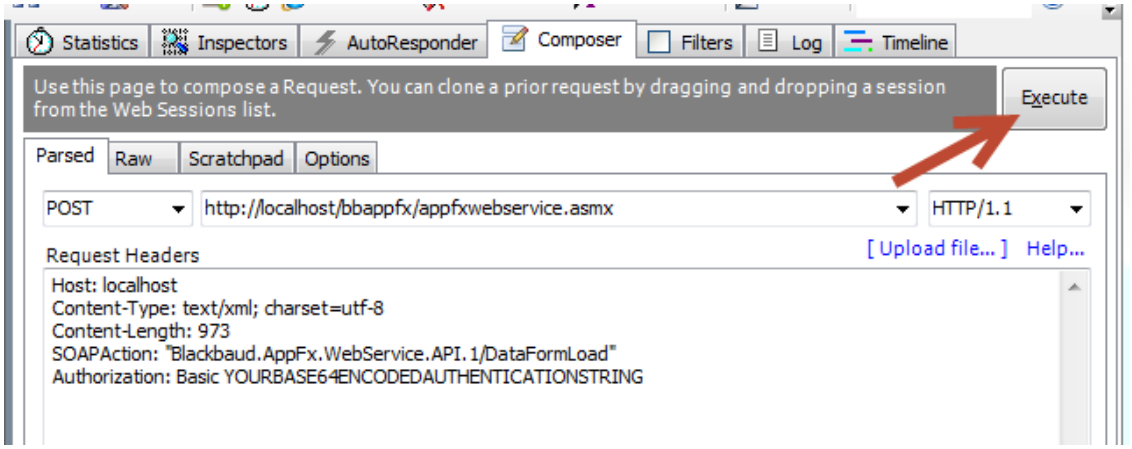


Figure 3.1: Screenshot of Fiddler (Lawrence, 2020), the web debugging software referred to by P5. The arrow shows the execute button for the script text box coming before the text box itself, which made it difficult for P5 to navigate the interface.

> *''I was working with fiddler today. What I was trying to do was edit the fiddler script, and it turns out that the way it's laid out is the button, and the button is followed by the text box to the right. It is the other way round... This did happen before that I didn't know the button was there, and I almost thought my focus was not going, and I asked my sighted friend, 'Hey, is there an accessibility bug here?'... You will almost think for a second... You will almost be sure that the screen reader is not reading that text box. So I had her look at it, and I was like, 'what's going on?', and she was like, 'that's how the interface is laid out.'''* (P5)

We see here that the BVI participant had a theory about the spatial layout of the interface that they revised when something did not work as expected, but they had to ask for sighted confirmation.

Finally, we discuss other non-dominant strategies participants used to learn visual semantics. P7-LV mentioned learning visual semantics through light perception (current partially functional vision making it possible to distinguish icons based on color for example), P9 mentioned late onset

of blindness, and P5 mentioned knowledge of underlying interface code. *''One, I would almost always get the information by looking at the actual code. Or two, [I would] again limit myself to, for example, 'Oh, this heading structure doesn't make sense' or 'Hey, it makes sense to have a horizontal rule here to separate two sections' or you know something else, for example.''* (P5). While understanding code requires a specific skill set, our participant's ability to understand visual semantics (like the relative location of elements) from linear representations is intriguing, if unusual.

Despite benefiting from visual semantics, participants informed us about two key learning challenges across contexts: missing information (N=5) and inconsistent representations provided by screen readers (N=4). From observations, we learn from four participants that inconsistent information provided by screen readers was a major hurdle for their learning and interacting with spatial layouts. for example, some screen readers differ in whether the arrow keys map onto the direction in which elements are laid out: *''... if you are reading using the PC cursor voice, you are using the down arrow to go down, but if you do it with the JAWS cursor, the direction is going left to right. So the buttons may be in a line across the top but JAWS is reading as though they are down the page vertically.''* (P6). This issue was specific to desktop interfaces due to the differences in the design of desktop and mobile screen readers.

In addition to inconsistent information, we observed that missing images and other graphic information by screen readers was a barrier for BVI users to understand visual semantics. Interestingly, P5 was under the impression, based on screen reader feedback, that there was an image on his personal webpage. Though the screen reader mentioned there being a graphic, what was visually displayed was just a placeholder. *''I actually don't know that it's actually a home icon over there and not actually home written in words there. So it's kind of a lot of gap ... ''* (P9). Moreover, several participants mentioned missing information as a hurdle to understanding spatial layouts.

Multiple participants reflected on the effort necessary to learn visual semantics and how this affected their smartphone usage, both negatively and positively. for example, P10-LV said: *''When*

I first start using something, I don't know the placement and stuff, so it takes me a bit to play around [. . .] Once I am used to it, I am hesitant to try something new.'' (P10-LV). Similarly, P2 described the trade-off between effort and payoff relative to interaction efficiency: *''If I'm using something frequently, I'll take the time to learn the spatial layout.''* (P2).

To summarize, we observed challenges to learning due to the inconsistent interpretation and incomplete representation of visual elements, graphics and images by screen readers, and the inherent cost (in terms of time and complexity) of learning new interfaces.

Mental Models of Visual Semantics

To understand BVI users' perception of visual semantics and inform strategies to narrow the gap between non-visual representations and visual semantic understanding, we report findings on perceptions of visual semantics, notions of size and shape, and understanding of overall structure in BVI users. Insights into BVI mental models of visual semantics could better situate future work to support non-visual access to this information.

BVI users' perceptions were most acute for smartphone apps, followed by webpages on the smartphone, followed by webpages on the desktop. We now describe these perceptions of size and shape of UI elements, and their overall perceptions of UI layouts.

Participants associated size with functionality and context (app and web). For example, when reconstructing the layout of the Twitter app on a smartphone, P4 said, *''These are tweets. Sometimes they can be small, depending on what people put. Sometimes, they put pictures and links.''* (P4).

Four participants had some estimation of size when prototyping smartphone apps. Participants' think-aloud comments explain this: While exploring the Google Maps app home screen before reconstruction, *''There seems to be like a basketball game event suggestion, and that seems to take up like a lot of space.''* (P3). She also said, *''I have like a skinny search bar on the top and then like two blobs of Play-Doh.''* (P3). While not as precise as P3, P5 was aware of the misrepresentation of

the size of elements. Relatedly, two participants had a notion of the size of elements on webpages as they appeared on the smartphone, and three participants had some notion of the size of elements as they appear on webpages when browsed from a desktop. Similarly, for P8, the size of the canvas dictated where elements went in terms of columns. We observed that participants constructed UIs by column and moved over to the right when they were out of vertical space.

While screen readers do not convey the meaning of shape, it was surprising to hear notions of shape described by BVI users. Our findings do not however inform comparison or contrast of these perceptions across device or contexts, since the shapes that participants created were not very distinct from each other across device platforms or usage scenarios. Participants used representative shapes instead: links were straight lines, buttons were blobs or circular shaped Play-Doh. For example, while lo-fi prototyping the Facebook iOS app, P2 represented buttons for the camera and messenger icons as round circles. Some participants, however, were very creative in physical representations of UI elements. for example, when describing her reconstructed model of a desktop website, P9 used a large rectangular piece of Play-Doh to denote a search results section and used a pen to make a horizontal indentation to denote a heading in the search results.

We find BVI technology users have notions of different kinds of UIs (app and web) to varying degrees. They had notions of general layouts (N=2), layouts of webpages on desktops (N=6), and layouts of websites they own (N=1). Participants developed these layouts primarily based on screen reader representations (N=2). Six participants primarily perceived layouts of websites on the desktop as being vertical. Evident from P8's experience, *''Old habits, really more than anything. No reason why I couldn't when I learn; I've been using a computer since I was 11. So everything as far as [the] Internet was vertical, line-by-line.''* (P8). Similarly, we observed P5's lo-fi prototype of a website as it appears on the desktop to predominantly have everything to the left, with the right side of the canvas remaining empty. Contrastingly, participants' lo-fi prototypes of phone apps were more spread across the canvas as opposed to those of websites on desktops.

Looking at a higher level than shape, P5 and P8 had some general awareness of news website structure. *''You know, for a news article. You know well there is probably going to be ads above and below the actual news content.''* (P5). This was also evident in P1's lo-fi prototype of a webpage as it appears on the desktop. She was confident to construct the layout of a generic news website as opposed to the layout of a particular one.

Lastly, participants developed their mental models of layouts through screen reader feedback. As P6 said, *''Just the concept that, to me, spatial layout is based on the screen reader interface. It's not based on what it looks like. And that happens across accessibility formats. So my perception of how a page looks using a braille display is different than using JAWS. Because the braille display can go only 40 cells at a time. So the concept of spatial is very different.''* (P6). To further understand this trend, we compare accuracy of participants' prototypes to whether or not they referred to the original interface.

Six participants referred to the original app interface and four did not. Of those that referred (N=6), three participants re-created fairly accurate prototypes. Two participants produced moderately accurate layouts, while one produced a prototype with low accuracy. Of those that chose not to refer to the interface (N=4), one produced a fairly accurate layout, one moderate and two low.

For websites on the smartphone, five participants reconstructed the layouts on referring, three chose not to refer and two did not prototype. Note that all the fairly accurate layouts were produced by participants who referred to the original web interface. Accuracy was overall moderate to low (N=8). For participants who chose not to refer, one participant produced a moderately accurate layout, while two produced low accuracy prototypes.

For desktop web, six participants chose not to refer to the original website interface while prototyping. Of these six, five participants chose not to refer, and reconstructed prototypes with low accuracy. Of those that referred (N=3), one constructed a fairly accurate prototype, one constructed moderately accurate and one participant's prototype was less accurate.

Surprisingly for personal websites, accuracy was moderate (N=4) both for participants who referred (N=2), and did not (N=2). One participant who did not refer produced a prototype of low accuracy. Our findings reveal how participants represented UIs, what they struggled with, and their preferences while prototyping like reference points, strategies to describe layouts, and navigation order while constructing. Existing lo-fi prototyping techniques and software are not accessible to BVI individuals. However, to illustrate BVI conceptualization of visual semantics, we next present insights from the low-fi prototyping task.

Prototyping Visual Semantics

To better understand BVI users' conception of visual semantics, we asked participants to prototype familiar UIs with lo-fi materials. We analyzed and report on our observations, their material designs, and their ''think aloud'' comments. We detail our observations on prototyping preferences and strategies to describe UIs.

We observe that participant preference did not vary with device (smartphone vs. desktop) and usage scenario (app vs. web) while lo-fi prototyping.

Ten participants used reference points for smartphone app reconstructions. P4, for example, started construction in the bottom-left corner of the canvas, and P5 started on the top-left. Similarly, P9 started on the top and bottom, thereafter proceeding with placing straight lines of Wikki-Stix from bottom to top. We observe that P3's notion of a reference point included static elements of the phone's UI. When asked to prototype the layout of a smartphone app, she clarified by asking, ''*No status bar, right? Like the time and stuff.*'' (P3). Relatedly, we could observe reference points for five participants as they constructed layouts of websites they owned and built. P3, for example, started by constructing the browser's address bar. Similarly, P5 started from the top. P9, in addition to starting from the top, divided the canvas into sections. We assume that this division was due to her prior experience as a web developer.

While ten participants constructed lo-fi prototypes, we perceived four participants to be generally interested in these prototyping tasks. All participants prototyped the layout of the smartphone app. When asked to construct the layout of a phone app, P9 was very enthusiastic and accepted the task like a challenge. She said, *''I think I can do the phone app. It has more stuff. feels like I've become a kid again. It's been a while since I played with all of these. Especially I used to play the fun school game.''* (P9). Two participants expressed disinterest but continued to prototype the layout of a smartphone website, and two chose not to prototype. One participant chose to construct the layout of a generic webpage. for desktop web, one participant did not prototype, and one participant chose to prototype the layout of a generic news website, mentioning that she was not familiar with the layouts of any specific website that she browsed on the desktop. *''I don't have in my mind the layout of a particular website. I am trying to think [. . .] also the websites [. . .] they look different from one another to me.''* (P1). All five participants who had a personal website prototyped its layout. Note that not all participants expressed a preference or sentiment about our prototyping task.

We summarize different participant strategies to describe and construct interfaces to inform the readers of BVI notions of verbalizing visual semantics. Description strategies included the use of relative positions with some elements as anchor points (*e.g.,* , ''the button is below the title header''), use of corner elements, use of element counts and absolute positions (*e.g.,* , ''the button is at x,y location''), with relative positions being predominant (three for smartphone apps and three for personal webpages). We also present other strategies, including assumed location and absolute positions, and discuss participant strategies for constructing interfaces, *e.g.,* navigation order.

Three participants used relative positions to describe UIs and lo-fi prototypes. *''In my view, the settings app is having one kind of notification area on the top, which kind of shows me tips if my airplane mode is on; below that, I have a list of things network and Internet, display and sounds; I only go to network and Internet all the time.''* (P9). Similarly, three participants used relative positions to describe layouts of web pages they owned or built.

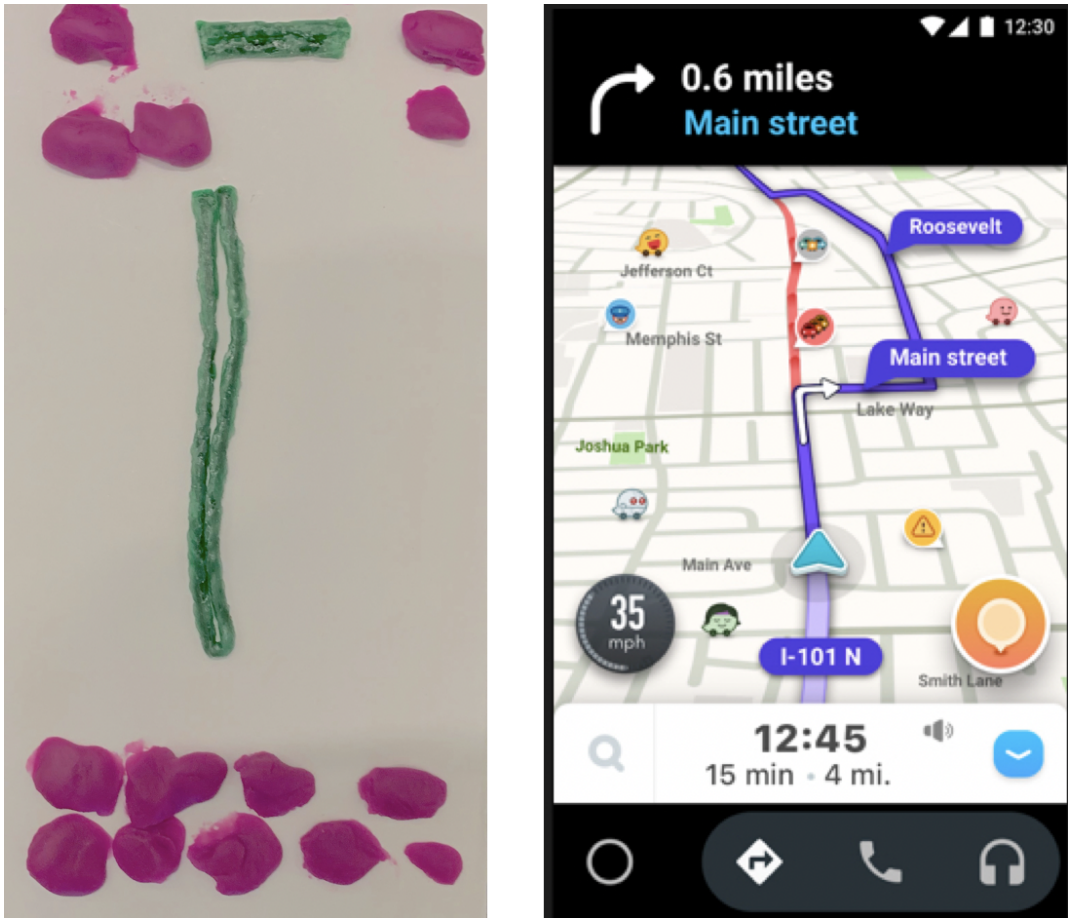


Figure 3.2: P6 smartphone app reconstruction (left) of the Waze navigation app (right).

‘‘The way I think about this is on the top is my email, to the right is my phone number. Below that [is] essentially a 2 by 2 kind of thing, which has my social profile.’’ (P5). Four participants used edge elements to describe layouts. *‘‘Well, at the top of the screen is the buttons that tell you like the directions. There’s also. . . when you first open the app, first there’s search and then voice; here’s a microphone. In the middle, there’s the map and then, at the bottom, there’s the buttons for menu and sound and speed and stuff like that.’’* (P6). Similarly, three participants used corner elements to describe smartphone layouts. *‘‘And in the top right is the add contact button.’’* (P6). Interestingly, one participant, P9, used absolute position to describe the layout of the contacts app.

We next present findings on navigation order strategies that participants used to: prototype layouts, refer to the original app for construction, and to describe the constructed layouts. In the context of screen reader access, navigation order is determined by the UI builders. However, it was interesting to see participants using a variety of strategies, like going left-to-right, top-to-bottom, and bottom-to-top when constructing, referring to the original interface, and describing the reconstructions. We observe that participants preferred going left-to-right, top-to-bottom on the canvas when they were prototyping layouts (six for smartphone app, seven for websites on smartphones and desktops, and four for personal websites). The second most common observation was that participants placed UI elements on the canvas in a top-to-bottom fashion (two for smartphone app, one for smartphone web, two for desktop web, and one for personal website). We noticed that the constructions where participants went in a top-to-bottom fashion had less detail, likely resulting in their not going left-to-right. Lastly, it was interesting to see two participants go in no specific order when prototyping a smartphone app.

While describing their interfaces, participants went predominantly from left-to-right, top-to-bottom across contexts (eight for apps, seven for websites on smartphones and desktops, and three for personal websites). The second most commonly observed strategy was to only go top-to-bottom (two for apps, one for smartphone websites, two for desktops and one for personal websites). Interestingly, we observe that one of the participants lacked sufficient detail in their reconstruction to go left-to-right, top-to-bottom.

Four participants followed a left-to-right, top-to-bottom order when referring to the smartphone app during reconstruction. Three participants did not refer to the app, and we were unable to discern the order for one participant. for websites on the smartphone, four participants followed a left-to-right, top-to-bottom order, and two participants did not refer to the original webpage. for desktop websites, while three participants used screen reader ordering, it was interesting to see five participants not referring to the original layout.

We observe similar behavior with respect to personal websites (two used screen readers, three did not refer to the original). Though an accuracy comparison between the reconstruction and actual interface would help clarify this, we do not report accuracy related findings; limited non-visual access to visual attributes of interfaces, the disconnect between visual ordering of elements and semantic structure provided to screen readers, and the lack of access to visual information may make such a comparison unfair.

Building Web Pages

Five of our ten participants had webpages that they built or maintained. These pages were also meant for visual consumption. We explore BVI developer strategies for making visual design decisions and building visual layouts.

Participants used WordPress and WordPress templates (three participants), sighted help (N=3), and HTML (N=3) to develop or maintain their webpages. WordPress was the back-end used by three of the five participants who had a webpage. Three participants perceived it as a tool, and two participants commented on the value of WordPress templates. When asked if they built their own website, P9 said, *''Yes and no. Like not kind of like I coded it. I just plugged in kind of a lot of WordPress plugins.''* (P9). To the same question, P3 said, *''I used a template. I don't know what it is called [. . .] I write the content. The layout is generated by the template.''* (P3). Three other participants used their knowledge of HTML markup to develop parts of their personal webpages or blogs. *''I didn't use a platform as such, like WordPress or whatever. It's just plain HTML and CSS.''* (P5). Lastly, three participants relied heavily on sighted assistance to develop their website. *''I literally copied someone's code. . . with their permission obviously and changed things.''* (P5).

Surprisingly, based on our observations and interpretations, no participant independently selected the template for their webpage nor pushed major updates to it—informing major areas of contribution for future design tools.

Participants used templates and settings that are known to be accessible (N=2) and assistance from friends and family members to make visual design decisions on their webpages. In P3's words:

> *''I used a template. I don't know what it is called. When I got a website, I had a friend set it up for me. He chose a template that is supposed to be more accessible. I don't really know what he did. He turned off the WYSIWYG editor... He made a couple of tweaks to make that easier for me. I hate that I don't know what he did.''* (P3)

P3's words reveal the sense of dependency associated with making a visual layout choice, and the lack of autonomy and knowledge that result from this dependency.

From the observations, we see that BVI individuals find utility in visual semantics across context. While existing access technology provides this information, there are significant limitations that prevent BVI users from making the best use of this information. furthermore, insufficient access to visual semantics increases the difficulty of non-visually building visually appealing layouts.

> *''I am really really bad at this is what I understood. [...] Its also very interesting to understand and appreciate the more commonly used websites and know about them so that you can better design your website [...] All that I really care about are things that visually might look awkward to people but how they exactly look, I don't care that much. If I think about it now, maybe its not the right thing. Maybe you want to understand how the layouts actually look and make some contribution there.''* (P5)

Here, we find an introspective realization of importance of visual semantics for BVI users, further strengthening the need for non-visual visual semantic access. P5 did not care much about the appearance of user interfaces beyond their appearing awkward, but after answering our questions and prototyping four UIs, he reflects on the need to have access to this information and expresses desire to pay attention to these.

3.3 Summary of Findings

Part One: Semi-Structured Interviews: Despite not having full access to a UI's visual semantics, all participants felt that *this information was critically important*. Visual semantic information equipped participants to efficiently interact with and navigate UIs, collaborate with sighted users to provide and receive instruction, and build their own visual artifacts such as websites and blogs.

Though our participants considered this information important, they indicated that *having access to visual semantic information is not necessary at all times*. This suggests that (1) technology like screen readers that may want to expose more visual semantic information could explore user controls to enable and disable access to this information, and (2) apps tailored to tasks that require the use of visual semantics can explore creative interaction techniques and feedback mechanisms.

Our findings show that participants had a stronger understanding of visual semantics of smartphone apps and for web interfaces that they browsed on a smartphone. The spatial nature of touch interactions in smartphones, the capabilities offered by touchscreen screen readers such as VoiceOver to directly touch and manipulate a UI control on screen, and careful placement of UI elements (Kane et al., 2011b,a) may have contributed to this strong understanding. Though participants felt that they understood some desktop UIs, this information was valued less and visual semantic understanding was lower compared to smartphone UIs. This difference in understanding of UIs indicates *the need to explore new interaction paradigms that make use of spatial layout familiarity that BVI users may have* to support tasks that need visual semantic information.

Parts 2 and 3: Screen Reader Tasks and Low-fi Prototyping. Our screen reader tasks and low-fi prototyping activities were designed to inform us about BVI understanding of specific visual semantics. Participants demonstrated knowledge of the overall layout of apps and websites, size of smartphone UIs, and a rough understanding of the shapes of UI controls. They associated the size of a UI element with the functionality it provided. Further, they referred to UI elements using their absolute (*e.g.,* here, there, at the top) and relative (*e.g.,* next to, below, above) positions.

Participants' understanding of shapes of UI controls, however, was not as well developed as their appreciation of other visual semantics. They used approximate shapes (such as circles) for buttons and links to represent UI elements in their tactile prototypes.

Finally, some participants who had their own personal websites and blogs echoed findings from prior work that the lack of access to visual semantics was a major barrier for them to independently engage with visual designs of their own artifacts. These findings highlight the need to expose visual semantic information in tools that support visual design. Our work leverages computer-aided assistance and automatic repair approaches to support BVI developers and designers making visual design decisions, addressing the gaps in BVI understanding of UI aesthetics.

3.4 A Multimodal Approach to Accessibly Edit Webpage Designs

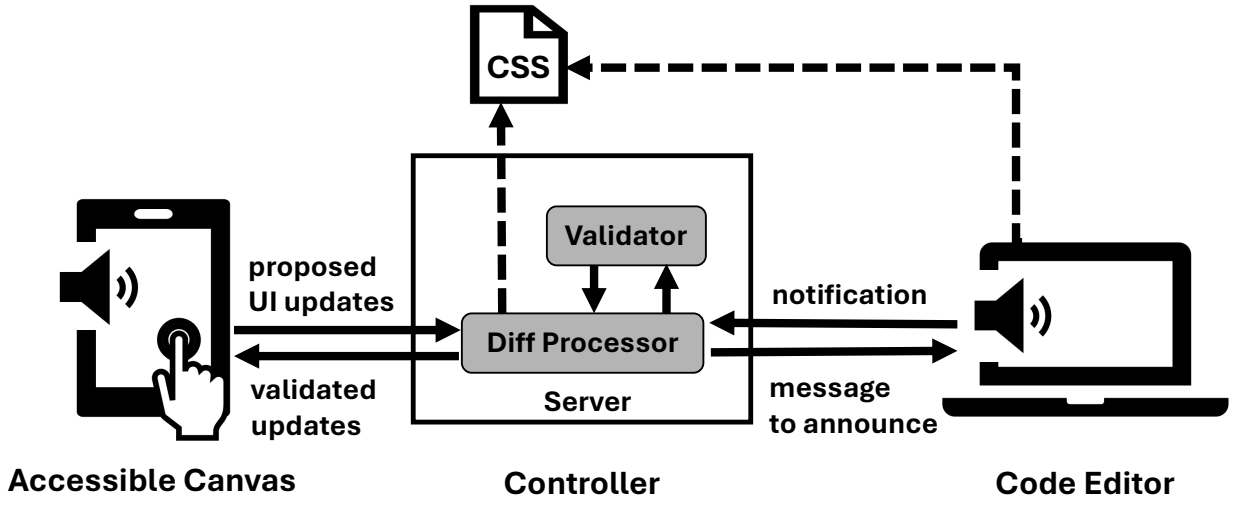


Figure 3.3: UITap components and the workflow to process UI edits.

To address the accessibility gap of editing of UI visual semantic information, we explored a set of multi-modal interactions using a touch screen and keyboard in a prototype system, *UITap*. UITap's multi-modal interactions utilize BVI familiarity of spatial layouts offered by smartphone

screen readers to make the editing of webpage layouts accessible. To account for the gaps in visual semantic understanding that BVI developers may have about webpages, we explored the use of computer-aided assistance to identify *visual appeal bugs* and provide feedback to BVI developers.

To edit visual attributes of webpages using UITap, BVI developers could (1) perform a set of custom gestures on an accessible canvas (an iPad running the UITap software) or (2) directly edit the CSS of the webpage in VS Code (an IDE that provides an accessible editing experience) running our UITap extension. Once a developer makes an edit and confirms it (by using VSCode's save function or by performing a three-finger triple tap on the canvas), the controller processes the changes and checks for adherence to visual design guidelines, provides speech feedback about any visual bugs, and rejects or accepts the changes. Figure 3.3 shows the different components of the system, and the accompanying video (Potluri et al., 2019b) demonstrates system features.

We conducted a preliminary user study of the prototype with a congenitally blind professional software engineer. The study had three parts: a semi-structured interview on current webpage development strategies, an evaluation of our system through three tasks (i.e., changing the typeface of a paragraph, increasing the bottom margin for a list of hyperlinks, and aligning an image and a paragraph), and a post-study interview regarding usability. The participant completed all 3 tasks successfully and agreed that the gestures were usable, although he had trouble memorizing gestures for operations. He understood task outcomes, including those that passed and failed design guideline violation checking. The participant expressed the need to have more control; for example, he wanted to have access to the history of changes that lead to a rejection.

3.5 Towards Real-World Accessible UI Developer Tools

3.5.1 Limitations

Our ten-person, exploratory qualitative study (section 3.2) to examine how BVI users understand visual semantics had samples drawn from a single geographic area in the US, and had five users who

previously built visual interfaces such as web pages, making our sample skewed toward expertise. This bias is appropriate to determining *what* information to convey, a primary goal of study, as evidenced by the influence it had on the design of the UITap system. Though effective to design developer tools, our findings may not fully explain novice perspectives. Furthermore, Our findings from a preliminary study of UITap demonstrate that even with technical ability, BVI participants did not find visual interfaces fully understandable. Less proficient BVI users, and beginner developers interested to build visual semantics may have even larger hurdles to overcome. future work should examine visual semantic education across users from diverse backgrounds. Further, surfacing this information in standard screen readers could increase exposure and facilitate familiarity.

3.5.2 Ubiquitous Access to Visual Semantics

The full potential of visual semantic access of UIs could be realized if participants could access these semantics during regular use (*e.g.,* using a screen reader), and when using devices prevalent in everyday computing (*e.g.,* touchscreen devices). Given the understanding of visual semantics demonstrated in our study with respect to phones, it seems clear that this is possible without requiring costly special hardware or expertise to generate its physical representations. This recommendation to provide access to visual semantics in everyday devices and user experiences is in line with prior work exploring sociotechnical considerations for accessible visualization design (Lundgard et al., 2019) and the cost associated with tactile graphics outlined in (Gorlewicz et al., 2020).

While existing screen readers present visual information by giving audio feedback on the mouse's location, conveying indentation and text formatting, and using three-dimensional sound to indicate the UI element location that is being activated (NV Access, 2019b; Omer, 2019), this approach does not expose the full range of visual semantics *e.g.,* size and shape of UI elements.

Our findings show us that BVI users attempted to understand a wide range of visual semantic information, whether or not specific features in screen readers surfaced this information.

These observations lead us to ask:

How can screen readers further help BVI users learn about the interfaces they use? Given that BVI users do not require visual semantic information at all times, we recommend that screen readers could provide a visual verbosity setting, where users can access visual information about interface elements, *i.e.,* , an interface design mode similar to punctuation verbosity. Similarly, screen readers could provide visual descriptions of entire interfaces *e.g.,* existing screen reader functionality to read screen content could be augmented to support descriptions of visual semantics. Our findings show BVI preference for description order of these semantics to be: left-to-right, top-to-bottom.

However, given that this preference may not be uniform (our participants showed variation in the element order when describing visual interfaces), we recommend ordering customizability.

Another factor that could add to the complexity in describing visual semantic is the length of a description, and the necessary detail. Our participants expressed familiarity with edge and corner elements of interfaces as well as familiarity with certain layouts, like news websites and frequently used smartphone apps. This suggests an opportunity to use data from UI repositories such as *Rico* (Deka et al., 2017) and *Webzeitgeist* (Kumar et al., 2013) to compare a new interface to those known to be used frequently by the same user. A layout description could then be generated that compares the current UI to the familiar interface and use AI methods to generate richer semantic descriptions. Future work could explore guidelines to describe user interfaces with visual semantic information, and assess the utility of screen captioning datasets and large language models to generate these descriptions (Wang et al., 2021; Abukadah et al., 2024; You et al., 2024).

3.5.3 Accessibility of Real-world Developer Tools

Our formative study findings echo related work (Li et al., 2019, 2021, 2022) and highlight that tool support to provide meaningful information to make UI design decision limits BVI people and developers to independently prototype, create, and edit user interfaces. The UITap system addresses

this information gap by demonstrating the idea of automated visual feedback. Though useful, results from our preliminary study indicate the need for more autonomy and granular visual feedback to respect a developer's expertise. Providing this feedback may require tool builders to re-think the notion of a *Design bug*, where the feedback is tailored to human factors such as the design system that the UI is being designed for, and subjective opinions on quality. Though no popular IDE or design tool currently supports accessible visual design and UI development by BVI developers, the advent of declarative languages like Apple's SwiftUI, with tools that support live previews of UI code on accessible touch screen devices like the iPad, present potential to become accessible design tools and embody the multi-modal developer experienced demonstrated by UITap.

3.5.4 Ableist Notions of Visual Semantic Access

While it may seem like the goals of this work are to norm visual semantic access, our objectives are contrary. We are not proposing that BVI users *should* have the same experiences with visual semantics as sighted users, but rather our work questions ableist assumption that BVI users can not or do not want to understand visual semantics. At the same time, designers should have a deeper discussion about placing responsibility on blind people to understand visual semantics to have similar experiences as sighted people.

Our work also deviates from a problematic corollary assumption, that BVI users cannot design visually pleasing interfaces that sighted users could use. Our findings show promise for a larger conversation about BVI users as capable and interested visual designers (Fleet, 2023). In an inherently normative world where people form impressions in a split second (Lindgaard et al., 2006), an understanding of visual semantics is critical for BVI users who wish to present themselves online. Our work provides a start at understanding this important problem.

3.5.5 Social Factors of Visual Semantic Access

It is important to consider the disability context when designing methods to introduce accessible visual semantics, particularly about the expectations relating to knowledge that BVI users may have about visual aesthetics. While this could open up new possibilities, it also has the potential to put undue burden on BVI users to learn new tools, to produce good quality output in a modality that they do not have complete access to, and open them up to additional, potentially unfair criticisms. We could observe some of these tensions while running our user studies: though not quantifiable, a few participants were not very comfortable with the prototypes they produced, and were highly critical of their skill in prototyping. While we reiterated that we *are not* testing their skills or ability to do a certain activity, the effect this experience may have had on their confidence is unknown. Likewise, if a BVI user creates an accessible UI tool that provides feedback on what is appealing, and if the output is not appealing to an end user or a person evaluating the interface generated by this tool, *is it fair to criticize the BVI creator?* In contrast, given that the BVI creator is responsible for decisions related to the output, *what could constructive criticism look like?*

The potential for errors increase with the rapid adoption of generative AI tools to assist with the creation and consumption of visual semantics (Glazko et al., 2023). For example, Generative AI (GAI), although helpful, can hallucinate and provide visually inaccurate descriptions and visual artifacts. On the contrary, based on my experience, image-to-text apps meant for accessibility may account for some of those errors as a result of a poorly captured image and provide a correct-sounding response, offering a correctness useful for consumption and *not for creation*. Finally, current GAI systems are not capable of giving information that could help assess image shareworthiness, opening up additional possibility for criticism of visual artifacts created by BVI creators (Glazko et al., 2023). Approaches in systems built for this nuance of error such as *GenAssist* (Huh et al., 2023) could be extended to provide nuanced feedback in the context of visual semantics. It is important to be explicit about the limitations of future tools that provide nonvisual access to visual semantics.

3.6 Contributions

Prior to this work, the research community had very little knowledge about how BVI technology user/developers conceptualize UI elements and react to the aesthetics of user interfaces. Furthermore, multi-modal programming experiences for accessibility had been limited to educational contexts (Milne et al., 2017). We demonstrated the first multi-modal approach to support accessible visual design that draws upon expertise that BVI users may already have and uses commodity hardware. In addition to demonstrating a multi-modal programming experience, UITap also demonstrates the utility of computer aided support for BVI developers to identify and fix visual bugs.

In connection to my thesis statement, UITap contributes a *new, multi-modal (touch plus keyboard) developer experience* to provide access to *Visual Semantic information—visual outputs of UI code* accessible to equip BVI developers to participate in editing webpages—a task representative of front-end development, critical in professional domains such as full-stack software development and the design of human-centered machine learning systems. In the next chapter, I discuss the accessibility of streaming data, another common occurrence of visual code outputs in programming.

Efforts related to the design and development of UITap led to a CHI 2021 publication (Potluri et al., 2021), an ASSETS 2019 poster (Potluri et al., 2019b), and a report outlining future directions to explore computer-aided assistance in accessible visual design (Potluri et al., 2019a).

Chapter 4

PSST: ACCESSIBLE STREAMING DATA

The ability to program physical computing devices such as Arduinos and Micro:Bits lets us personalize how the physical environment responds to us, serves as a training ground for STEM professionals, and helps make STEM education more accessible to people from diverse backgrounds (Buechley and Hill, 2010; Hodges et al., 2013; Galadima, 2014; Hodges et al., 2020; Chung and Lou, 2021). Despite this potential for impact, learning to program physical computing devices is challenging. Physical computing systems do not always convey their state, and unpredictability in hardware makes debugging them difficult (Booth et al., 2016; McGrath et al., 2017). Critical to successfully programming physical computing devices is the ability to understand how sensors respond to stimuli and identify appropriate thresholds to program desired responses (Booth et al., 2016; Chung and Lou, 2021).

To address these prerequisites for programming and debugging physical computing and embedded systems, a variety of tools and research projects incorporated support for data visualizations (Gendreau Chakarov et al., 2021). The Arduino can graph output sent to its serial port (Zait, 2018), and the micro:bit's Jacdac programming interface includes visualizations showing the current status of all sensors (Austin et al., 2020). This emphasis on visualizations of live data poses accessibility barriers to BVI developers attempting to understand and debug sensors. Most of the websites and tools developed to program these boards, and the tutorials created for them, are partly or wholly inaccessible; for example, in their study of over 3,000 online tutorials, (Davis et al., 2020) found that less than 2% met web accessibility criteria. Our efforts detailed in this chapter aim to

address the accessibility of programming of physical computing devices. We answer the following research question: *how can we enable BVI developers to understand and debug sensor behavior?*

4.1 Background on Accessible Physical Computing

BVI people have been working with physical computing since as early as 1980, as evidenced by the publication of documents by the (Gerrey, 1980). Prior work has investigated several aspects of physical computing accessibility for BVI people, including tooling for understanding hardware circuits (Gerrey, 1980), such as oscilloscopes and continuity testers, tactile circuit diagrams (Davis et al., 2020; Race et al., 2019), and soldering (Race et al., 2020b). Traditionally, the handling of these electrical tools and assembling electronics have been necessary preconditions for working with circuits. However, the advent of recent physical computing boards, such as the micro:bit (Austin et al., 2020), has obviated the need for some of these low-level tools and skills, resulting in a low floor for beginners to work with circuits and sensing technology.

This evolution, however, introduced largely inaccessible new block-based programming languages and web-based tutorials (Davis et al., 2020). While alternatives to block-based languages are available on a range of platforms from the Arduino to the micro:bit, the information necessary to understand sensors is communicated visually. Accessible data visualization for understanding sensors is not well supported (see section 2.3) resulting in BVI people experiencing accessibility barriers to work with physical computing devices.

A small body of work has begun to address the accessibility of some of these evolved experiences with physical computing (Bennett et al., 2019; Race et al., 2020a; Jain, 2021; Morrison et al., 2021). Unfortunately, access to sensor data has not been a focus of these efforts. Additionally, most tools to program accessibly do not explore real-time access to sensor data (see section 2.2). I will describe efforts to communicate sensor data to support the debugging of physical computing devices. This body of work informs the requirements to support access to realtime, streaming data.

4.1.1 Communicating Sensor Data

In a study comparing the efficacy of physical computing to other methods for teaching programming, (Chung and Lou, 2021) found that some of the learning goals students needed to achieve included basic understanding of how and why sensors responded to stimuli, and analog to digital conversion. (Booth et al., 2016) found problems such as using the wrong thresholds for a sensor, in addition to multiple hardware bugs that were visible in unpredictable sensor readings. Understanding state is a general problem that is critical in all programming tasks (Ko, 2004). It is, however, particularly hard to visualize the state in physical computing and embedded systems because the state can change so dynamically, and may not be stored in a variable.

A variety of tools and research projects have incorporated support for data visualization to facilitate understanding and debugging of physical computing and embedded systems. The Arduino has the ability to graph output sent to its serial port (Zait, 2018), and the micro:bit's Jacdac programming interface includes visualizations showing the current status of all sensors (Austin et al., 2020). Similarly, the Data Sensor Hub(DaSH) (Gendreau Chakarov et al., 2021) is a tool built with the micro:bit that enables students to measure and analyze data collected from various sensors.

Several systems show the voltage or current flowing through a circuit or breadboard (Ochiai, 2014; Drew et al., 2016; Wu et al., 2017). Scanalog visualizes analog circuits and interactive tuning of signal transformations (Strasnick et al., 2017). McGrath et al. (2017) went beyond visualization to lay out design goals to help users identify the source of a fault, make internal state visible, and provide context to debug visible behavior. The efforts highlighted here show the promise of visual outputs to understand and work with sensors. They however do not cater to the needs of BVI people to engage in physical computing.

Design Goal	Realization in PSST
DG1 Learn how sensors respond to stimuli (Chung and Lou, 2021)	Provide real-time access to streaming sensor state
DG2 Identify appropriate thresholds for sensors (Booth et al., 2016)	Provide reactive filters that highlight extremes as the user explores a sensor's range
DG3 Learn about unexpected sensor readings (Booth et al., 2016)	Let the user highlight sudden changes or unusual values
DG4 Analyze trends (Walker and Nees, 2005) and provide statistical summaries (Beilharz and Ferguson, 2009)	Support transformations and calculations of statistics over a stream, such as sonifying the slope, or calculating a running average
DG5 Display multiple concurrent information streams (Beilharz and Ferguson, 2009)	Select and assign sonifications to any number of different sensor streams

Table 4.1: A list of capabilities derived from our literature survey and the features of PSST that support them. Overall, PSST has the goal of making sensor state ''visible'' (McGrath et al., 2017).

4.2 PSST

We designed and developed *PSST* (Physical computing Streaming Sensor data Toolkit), a toolkit that supports the configuration of data displays—accessible interfaces to understand live streaming data. Though there are no clear design guidelines to support BVI developers to explore streaming data and debug sensors, an extensive body of work gives us clear guidance about requirements to support debugging and communicating sensor data. We draw upon this work and synthesize design guidelines supporting the exploration of sensor data in real time, summarized in table 4.1.

4.2.1 System Overview

`DataSinks`, `DataHandlers`, and `DataOutputs` are three major components of PSST. All of these components are driven by the `OutputEngine`, the starting point for sonifications, and a central repository that stores these components. Each stream of incoming data is associated with

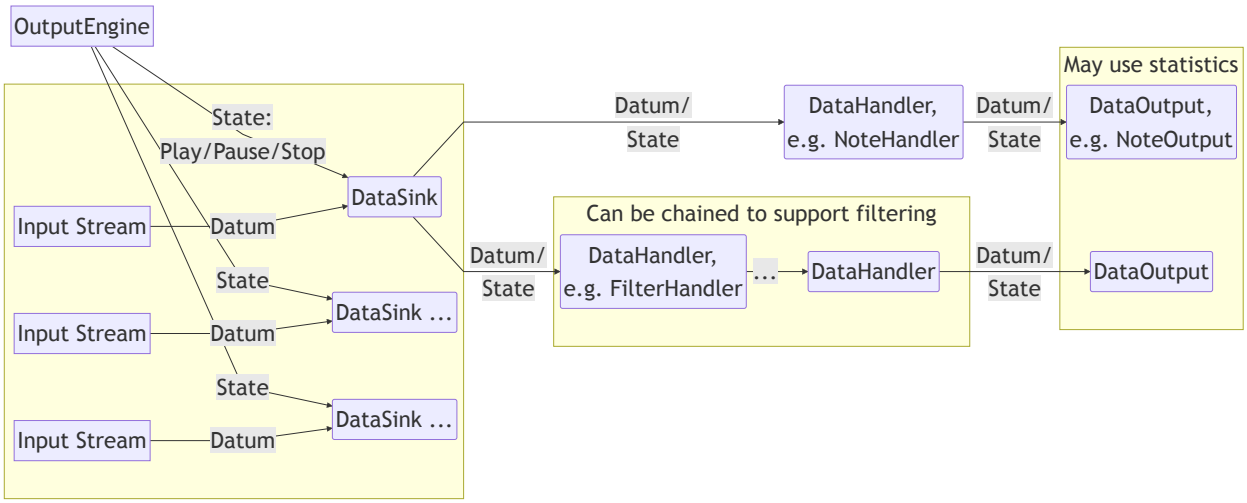
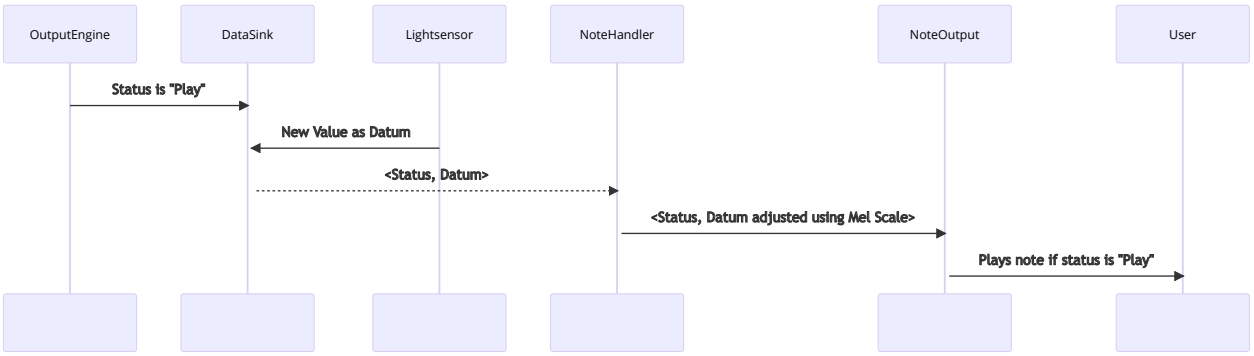


Figure 4.1: Architecture of PSST. Every object in PSST is an RxJs stream. By chaining together objects of different types, the user can specify the sonification of a stream.

a `DataSink`, which keeps track of one or more `DataHandlers`, and passes the data, along with the system state, *e.g.,* play or pause. A `Data Handler` handles data based on system state, possibly manipulates or filters the data, and passes the data to its `DataOutputs` or to additional `DataHandlers` that are chained to it. For example, a `FilterRangeHandler` *filters* data and only sends values in a certain range to its outputs. Similarly, a `NoteHandler` *manipulates* incoming sensor data into an audible range before sending it to its outputs (typically this is the `NoteOutput`). `DataOutputs` receive data from `DataHandlers` and output them in accessible formats. PSST's built-in outputs play specific audio files, speak values, play white noise, play a note corresponding to the output value, or add elements to an SVG file.



(a) A sequence diagram showing two events streamed to a DataSink—first, a Status, and second a new Datum. These are passed on to the NoteHandler, which adjusts the Datum value before passing it to the NoteOutput. The NoteOutput plays the note if Status is "Play."

```
        /**
         * Create a NoteOutput and configure
         * it to play audio from both speakers
         **/
        output = new NoteOutput(0)

        /**
         * Create a NoteHandler and specify the
         * range of input values will be between 0
         * and 1. By default, NoteHandler converts
         * values in this range to an audible range
         * using the Mel scale.
        **/
        handler = new NoteHandler([0, 1], output)

        /**
         * Register a new sink with the OutputEngine
         * and add the NoteHandler to it
        **/
        lightSensorSink = OutputEngine.getInstance().addSink('lightSensorSink')
        lightSensorSink.addDataHandler(handler)

        /**
         * Send an OutputStateChange.Play downstream
         * to turn sonification on. Similar
         * calls can Pause and Stop play.
        **/
       OutputEngine.getInstance().next(OutputStateChange.Play)
        /**
         * Any number of methods can be used to
         * pass on sensor data by calling
         * lightSensorSink.next([id],[sensorvalue])
         **/

```

(b) Sample code showing the configuration of a handler and output to listen to a light sensor

Figure 4.2: Scenario showing PSST configuration to hear light sensor values.

We illustrate an example of a user configuring PSST to hear data from a light sensor. The user first creates a `NoteOutput`, configured to use both speakers for output. The user then creates a `NoteHandler`, passing the expected range of sensor values ($[0, 1]$ in our example) and the `NoteOutput`. The `NoteHandler` will use this information to modify any incoming `DataObject` into an audible range before streaming it to the `NoteOutput`. Finally, the user creates a `DataSink` (which can receive input from a RxJS stream or by direct calls to `sink.next()`). This will receive the light sensor's values and stream them to the `NoteHandler`. The code corresponding to this scenario is in fig. 4.2b and the data flow is shown in fig. 4.2a. We see the `OutputEngine` streams ''Play'' followed by a new `Datum` to the `DataSink`. This in turn streams to the `NoteHandler`, which modifies the value of the `Datum` into an audible frequency relative to its position in the range $[0, 1]$. Finally, the `NoteOutput`, receives the converted value and plays a note for the user.

4.2.2 PSST Library Components

PSST is implemented in TypeScript. Audio output is handled using the Web Audio framework and speech is generated using the Web Speech API—both technologies built into modern web browsers. RxJS is used for all streaming-related functionality.

DataHandlers

PSST includes *Data Handlers* (listed in table 4.2) for filtering, tracking data patterns, and scaling. By default, Data Handlers support subscription by one or more Output objects.

The handlers provided fall into two main classes. *Transformations*, the first of the two classes, modify the data without filtering it. For example, PSST's `ScaleHandler` uses a provided function to translate from an input range to an output domain. This transformation is useful, for example, in its subclass handler, `NoteHandler`, which includes some specific parameters for the

Classification	Handler Example	Description	What is Created
Transformation: Converts values from an input range to an output domain	`ScaleHandler`	Generic transformation function	Scaled values
	`NoteHandler`	Specifically maps properly to audible notes	Audible frequency values
	`SlopeParityHandler`	Indicates whether the line's slope has changed	Boolean values
Filter: Removes a subset of values based on query criteria	`FilterRangeHandler`	Removes values not in a specified range	Remaining values
	`ThrottleHandler`	Keeps every n^{th} data point	Remaining values
	`NotificationHandler`	Keeps values that match a provided array	Remaining values
	`ExtremaHandler`	Keeps running extrema (maximum and/or minimum) based on all previous values	Most recent extreme value(s)

Table 4.2: Different default handlers supported by PSST. Users can extend this set by implementing new handlers.

domain relevant to correct sonification as audible notes and a function that specifically handles the complications of audio. Another handler, `SlopeParityHandler` returns true or false only when the slope has changed parity.

Filters, the second class of handlers, modify which data points are seen downstream. The `FilterRange` handler, one of the handlers included in PSST, only outputs `Datum` that have values that fall within a range. The `Notification` handler, another handler of this class, only outputs `Datum` that match an array of specific points. Similarly, the `Extrema` handler dynamically changes what is filtered based on the values it has seen before (the previous maximum or minimum).

PSST supports additional data exploration possibilities by allowing for handlers to be chained. For example, chaining the `Extrema` and the `SlopeParity` handlers could be used to highlight whenever the data moves from a peak to a trough and vice versa.

Output Example	**Description**
Note	Outputs a tone corresponding to the data. Can configure to play from the user's left, right, or both speakers.
Noise	Outputs white noise. Can use to highlight data after filtering.
Speech	Speaks the data value. Can configure the volume and rate of the speech. Useful after filtering and throttling.
Audio File	Outputs a wav file. Can choose a user-defined file name. Useful to alert to specific changes in the data.
SVG	Generates a graphical output. Can be used to generate output suitable for a hand-cranked music box.

Table 4.3: Outputs supported by PSST.

DataOutputs

Outputs are the leaf nodes in the data handling chain. They simply take data and ‘‘display’’ it. Their functionality is independent of the handler they are attached to. PSST supports output to a frequency (using an oscillator to generate a tone), white noise, playing an audio file, and speech (table 4.3). By default, the frequency or note output is continuous (*i.e.,* it plays until the next data point arrives) and all other outputs are of fixed duration. Frequency can be played in either the left or right channels or a combination, but does not currently support live panning. A new data point will interrupt the prior output if it arrives before the output is done. While this interruption may not impact the listening experience for note outputs (it may even be desired), its impact may be more perceived in the speech output, which supports a configuration to not interrupt.

4.2.3 PSST Dashboard

Although PSST is designed for programmatic use, a graphical user interface (GUI) that exposes a subset of the toolkit can reach additional users without requiring them to have programming experience. By encapsulating the PSST library as an accessible web application, the PSST dashboard interface (shown in fig. 4.3) enables BVI users to author sonification displays based on live sensor streaming data. We designed the dashboard with the aim to optimize accessibility via a screen reader and to support the design goals derived from our earlier literature survey (**DGs1-5**).

To support live streaming data, the interface provides connectivity with Jacdac microcontroller and sensor devices.Jacdac((Microsoft, 2022)) is a hardware and software stack that supports plug-and-play connections for low-cost hardware components and facilitates lightweight web development to build interfaces around it. We automatically create a `DataSync` for each sensor attached to the Jacdac device that is connected to the dashboard. With the PSST Dashboard, users can connect a Jacdac microcontroller, display live streaming data from attached sensors (*e.g.,* light sensors, accelerometers), configure sonification displays, and get immediate feedback by listening to sonified streaming data. Below, we explain how a BVI user can interact with the PSST Dashboard with an example scenario. A user starts by connecting a Jacdac microcontroller via USB to their computer, and then selects the Connect button on the dashboard interface. Once the user selects the microcontroller driver from a modal dialog, the interface populates with information about all sensors connected to the microcontroller.

In the case of the figure 4.3, the user has connected sensor devices that measure: light level, temperature, and humidity. The user interface exposes the connected sensors via an aria-live region. This ensures that whenever the user adds or removes a sensor to the microcontroller, their screen reader re-announces the changes. In the interface, each sensor stream is displayed visually with the most recently recorded value (*e.g.,* 72% for the light level, 24.2°C for temperature) and a button to add a `DataHandler` to that sensor data stream. This helps to support collaboration with sighted

users. Though this visual value is displayed to the user using a screen reader, its changes are not communicated using aria-live regions, as that would make the output impossible to understand with the screen reader announcing constantly changing values from multiple sensors.

The user can start the process for creating a sonification display by attaching a `DataHandler` to a sensor's stream. The user clicks the button to add a `DataHandler` to the humidity sensor after adding a `NoteHandler` to the light level sensor and a `FilterRangeHandler` to the temperature sensor (fig. 4.3).

To validate the extrema for a light sensor and calibrate it to the current ambient light level, the user can customize the sonification via the parameters and outputs for the attached `DataHandler`. In the case of the `NoteHandler`, which requires min and max values for a note range, the user provides 0 and 100, respectively. However, the user is not certain whether this is correct for the light level sensor (it will just play a very high or very low sound if they guessed the range wrong). To confirm the range, they add an `ExtremaHandler` and configure it to verbally speak the value of each `Datum` it encounters that is smaller than or equal to the current known minimum, or larger than or equal to the current known maximum (**DG2**).

Once the user presses play (located at the bottom of fig. 4.3), the system begins to sonify any incoming sensor readings based on the configuration. Now the user can play with the sensor and explore how it responds to physical stimuli (**DG1**). The user physically covers and uncovers the light level sensor with different objects, discovering which ones allow some light through. They confirm that total darkness produces a value of 0. They shine a bright flashlight light directly at the sensor and discover that the correct maximum is 1, not 100 as they had previously thought (**DG3**). They can update the `NoteHandler` with this value dynamically in the interface and it will immediately adjust how it plays notes. The PSST dashboard supports real-time feedback of different sonification configurations (*e.g.,* adding and removing `DataHandlers`, or changing the parameters of a `DataHandler`).

Next, the user wants to investigate the temperature and humidity readings, both of which come from the same Jacdac sensor device. By holding the sensor over a steaming cup of tea, the user hopes to learn how closely the two readings are related. The user configures the `FilterRangeHandler` to have a range of 28 to 100°C and to have an earcon `DataOutput` that plays a ‘‘Bell’’ tone whenever the sensor data falls within that range. The user then adds a `SlopeChangeHandler` to the humidity data stream and configures its `DataOutput` to be an earcon that plays a ‘‘Whistle Down’’ tone. The user now wants the tone to only play when the humidity reading decreases over time (**DG4**), so they set the direction to be ‘‘Negative Only.’’ Now, holding the sensor over the steaming cup of tea, the user hears a ‘‘Bell’’ tone to signify the temperature has reached 28°C. When the humidity decreases, they will hear a ‘‘Whistle Down’’ tone. A few moments after placing the sensor back on the table, the ‘‘Bell’’ tone has stopped and the ‘‘Whistle Down’’ tone finally emits from the speakers. From this, the user learns that some delay exists between the temperature and humidity readings (**DG1**).

4.2.4 Supporting Design Goals

We provide example scenarios that demonstrate PSST’s value in supporting each of our design goals summarized in table table 4.1 and highlight scenarios that inspired the tasks in our study.

Learning how sensors respond to stimuli (D1)

One of the basic needs for physical and embedded computing programming is making a sensor’s state visible to understand how it responds to stimuli (D1). An example of meeting this goal is in our scenario: using a light sensor to explore ambient light in a room. Another example is learning how the accelerometer values along each axis changes when it is rotated. We included both as tasks in our study.

Identifying appropriate thresholds for sensors (D2)

Whether exploring how a new sensor functions, calibrating it, or confirming its function in comparison to its description, a user may need to understand the minimum and maximum values the sensor is outputting. As an example, the first author of this paper learned something new about how an accelerometer worked when he first configured a sonification with a range of 0-1 and then discovered it was outputting a value of 2 when he shook it hard. In our study, participants are asked to explore the minimum and maximum values of a light sensor.

Identifying unexpected readings (DG3)

A key goal to understand sensors is to identify expected and unexpected values that the sensor may output in specific scenarios. This can help with debugging, as well as discovery. For example, we asked participants in our study to design a sonification that plays white noise when the sensor is placed flat on a table. A naive user might expect the accelerometer values to be unchanging when it is not being touched, but sonification should help the user to hear that this is not actually the case. I learned something new about accelerometers through this capability—that the sensor generated a change in values even when it is placed flat on a table.

Analyzing and summarizing data (D4)

Transformations of data can be useful in deciding how and when to display it, or in providing new information that would otherwise be difficult to infer. This is especially important for BVI users, as certain types of trends that are easily understood in the gestalt experience of viewing a chart might be less clear in the linear world of sound. As described earlier, it is straightforward to add new transformations to the PSST ecosystem, based on calculations that can take into account a single data point (as with `ScaleHandler`) or multiple data points (as with `SlopeHandler`). This was not used as a task in our study.

Display multiple concurrent information streams (DG5)

When building physical computing systems, it is often the case that multiple sensors must work together in various ways. Understanding how two different sensors respond to the same stimuli, for example, can be important to building a more robust approach to recognizing when that stimuli is present. In the study, we asked participants to tell us how a temperature and a humidity sensor responded when placed at the mouth of a bottle filled with hot water.

4.2.5 System Design Summary

In summary, PSST makes sensor state accessible to BVI developers using the following features.

Realtime access to sensor data. PSST provides audio or speech feedback to data generated by sensors in realtime (D1).

Tools to explore sensor behavior. PSST provides filters that alert users to new extreme values as they are generated by the sensors and highlight sudden changes in values and compute statistics, giving developers tools to identify the range of values of sensors (D2, D3 and D4).

Access to multiple data streams. PSST provides functionality to sonify multiple data streams through different stereo channels and speech (D5).

I will describe a set of demonstrations that show the wide range of scenarios that PSST supports.

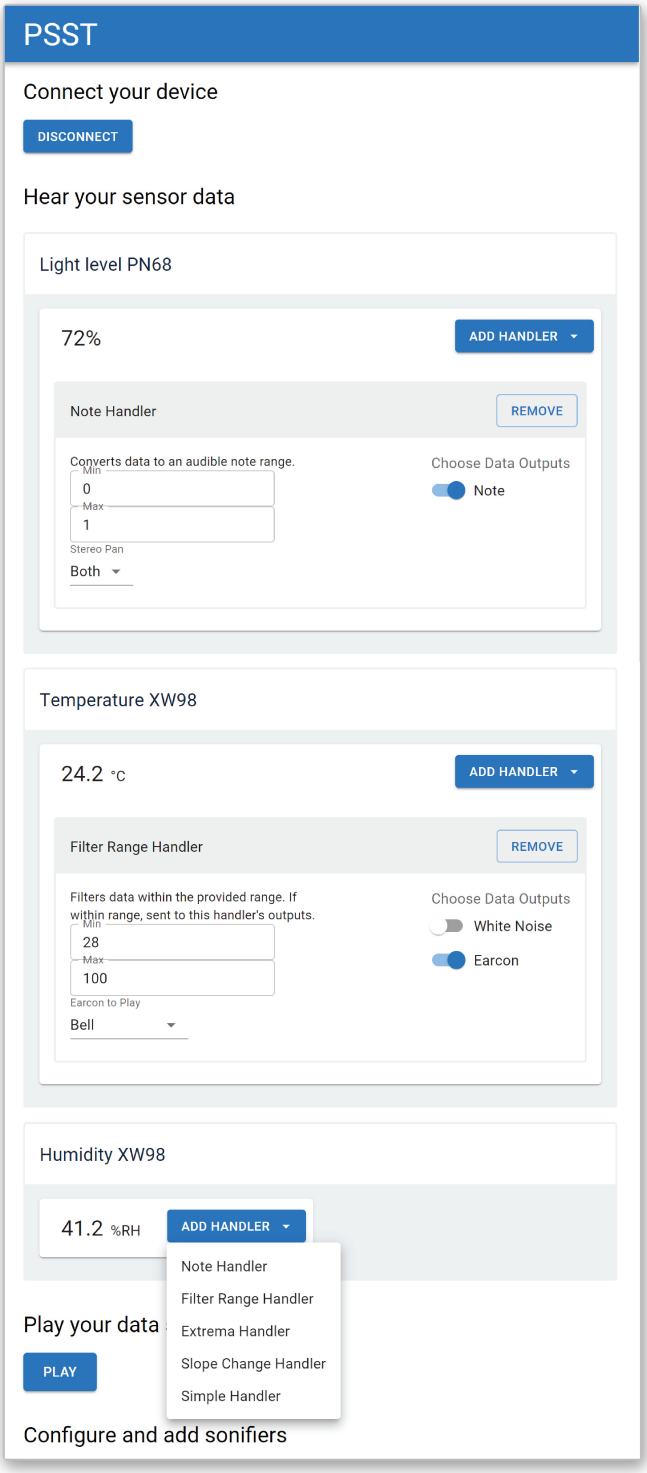


Figure 4.3: PSST dashboard for configuring sonification displays based on real-time sensor data.

4.3 Validation Demonstrations

Our demonstrations validate the utility of PSST across a wide range of application spaces. we implemented a physical data log that creates tangible output (and is compatible with a hand cranked music box); a piano interface that can be controlled with the keyboard; and replicated a feature of some screen readers, sonification of mouse position. These demonstrations showcase the wide range of inputs and outputs supported by PSST.

A Physical Data Log To explore PSST's ability to support a wider variety of modalities, we created a `TangibleOutput` for PSST. there is a genre of music box that takes as input a cardstock material with punched holes as input, and generates music. Our goal with this tangible output was to represent data in a way that can be sonified using this music box. The holes in the cardstock create an interesting tactile affordance (this is not the same as generating a tactile graphic). the handler-output combination created for this demonstration generates an SVG with a history of data that it has seen over a fixed time period. The SVG can then be laser cut to create a tangible record of the data that can be explored tactually. This was done by extending the PSST library with a new output object containing approximately 25 lines of new code beyond the basic constructor.

Keyboard and Mouse Input We implemented demos showing that keyboard and mouse input can be handled just like any other source of streaming data. For example, we convert keyboard letters to notes using a lookup table, which become straightforward to handle with a `NoteHandler`. Similarly, we can map mouse position to notes using the stereo output capabilities of PSST combined with pitch. All of these are possible with no changes to the PSST architecture or library.

4.4 Study: Validating PSST

To explore the value of the PSST dashboard to end users, we recruited two BVI individuals and observed them using the dashboard. Our goals were to assess if the PSST dashboard successfully supports BVI people in authoring sonifications and answering questions about sensor data, and to understand how beginners could use PSST to learn about the sensors used in physical computing.

Toolkit evaluations with BVI developers in under-explored domains such as making and physical computing are likely to require careful consideration (Mack et al., 2022). The study setup must be integrated with the right set of programming tools and a suitable definition of metrics such as task completion. In our evaluation using the PSST dashboard, we strove to avoid unnecessary accessibility barriers to ensure the best possible quality of data from our study. Designing the PSST dashboard for it to be accessible prepared us to handle some anticipate accessibility issues that our study participants may encounter.

4.4.1 Study Setup

We recruited two participants who self-reported as blind, with an advanced level of programming expertise, but with no physical computing experience. One participant used the NVDA screen reader and the other participant used JAWS.

We started with a semi-structured interview. We asked participants to tell us about their experience with using sensors in programming, what worked, and things that they felt did not work well. We then introduced participants to the PSST dashboard and gave them the list of tasks mentioned above. We asked participants open-ended questions about each sensor involved in a task before and after the task. We then returned to interviewing, to find out what they liked. This uncovered confusions about the dashboard, and how PSST could be made more relevant to participants' own data-related needs. Each study was attended by 2 facilitators and a participant. We used a laptop running Windows 10, with both the JAWS and NVDA screen readers

installed. Participants could use the screen reader of their choice, and could modify the screen reader configuration to suit their preference. All sessions were videotaped with two cameras and a software screen recorder to capture both the screen and the Jacdac hardware participants were interacting with. We reviewed researcher memos, notes taken during the study, and transcripts to identify themes, which we then discussed as a team.

4.4.2 Results

We describe our findings about the relevance of PSST, dashboard use, and describe, through participant observations, how PSST meets its design goals.

Relevance of PSST

Initially, participants seemed skeptical about the relevance of physical computing and sensor-based programming. Their answers to questions about what they would like to learn about most of the sensors were somewhat rambling, unfocused and not very animated or excited. Even though participants had some understanding of sensors, it was not clear that the tasks seemed relevant to them. This contrasts strongly with their attitude after the study. As they explored the sensors, they became increasingly engaged. P1 articulates this by saying:

> *''It was just kind of cool to see that for the first time in action I guess, with an actual tangible device... It was good to actually use them. I mean, you know, a lot of it, I think what might be true for a lot of blind people could be very theoretical things when you read about them but to actually kind of like see them in action when you don't really use all these things in an actual lab and to see how things change when you kind of experiment, I guess, that was cool.''* (P1)

As further evidence of PSST's relevance beyond physical computing, both participants suggested

new application domains for sonification. For example, one was interested in applying it to high volumes of telemetry data generated by applications. Another participant found value for this toolkit in data science work: *''You know, for example, kind of like in Pandas [a python data analysis library] if you can give a data frame or something and then, if I can actually set up these handlers through this library and then export or you know save the output into a file . . . and get a more refined data dump . . . that would be helpful.''* (P1)

Below, we describe how they used the PSST dashboard, including their ideation process and its accessibility, and their engagement with the goals identified in table 4.1.

Dashboard Use

For most tasks, participants used the note handler. The extrema handler in combination with the speech output was used to find the sensor values for specific stimuli. One of our participants misinterpreted the file output—an output that plays static audio files. They believed it would also vary the pitch of the selected audio file, similar to functionality offered by the note handler. Due to this misunderstanding, the participant initially assumed that moving an accelerometer was not causing the sensor values to change, until we explained that file outputs do not change pitch.

Participants iterated on displays to gain new insights. They first ideated on what would work, and then tried customizing the display—there was no single way to answer each question for them. They seemed excited by the process. For example, P2 found the ability to customize what they hear to be very useful: *''The thing I like the most about this was the customizability of the handlers. That was really cool. I've seen these sonifications for decades . . . but I've never seen anything nice in a dashboard like this that lets you add and remove multiple ones and tweak the parameters of each.''*

Participant feedback helped learn about the accessibility of the dashboard. Both participants were able to use the dashboard but one requested that the dashboard use more HTML headings, and found the Speech Output too noisy. Before the second participant session, we annotated each

sensor's name with an HTML heading (to improve efficiency using a screen reader) and added the throttle handler to reduce the frequency of spoken output. One participant also wanted the ability to spatialize speech, an interesting opportunity to consider in the future.

Addressing Design Goals

Participants demonstrated that we addressed design goals D1-D3 and D5 (summarized in table 4.1) successfully. Below we provide some examples of participant feedback which illustrates this.

Learning how sensors respond to stimuli (D1) P2 was interested to understand the sensitivity of different light sensors.

''Maybe you'd want to know how fast it responds to change, or something like that. Because I've seen that with sensitive light sensors, you're able to identify, say, the flickering of an electric bulb, compared with the constancy of sunlight, which I think is quite fascinating. Wagering whether you have a sensitive light sensor—I don't know—that could be something someone wanted to know. And honestly, I've never experienced that, I've just heard that.'' (P2)

When we asked our participants to examine and tell us how a temperature and a humidity sensor responds when they place the sensor at the mouth of a bottle with hot water in it, our participants noticed that the humidity sensor's value was increasing at a slower rate than the temperature sensor. Both participants added two note handlers; one for each sensor, mapped to the left and right speaker. One participant commented that the humidity sensor's value was increasing in steps, whereas the temperature sensor's value was increasing more linearly.

Identifying appropriate thresholds for sensors (D2) When asked to locate the light sensor on a micro:bit and report the minimum value of the sensor when it is covered, our participants made observations that the light sensor was very small in size and was noisier than they expected it to be. After adding a note handler to hear how the sensor value was changing and an extrema handler to

hear the new minima as they occurred when the user covered the light sensor, one participant also commented on the sensitivity of the sensor, noting that the sensor was much more sensitive than they expected it to be.

Learn about unexpected sensor readings (D3) Our participants derived many insights about the accelerometer using PSST. To explore the accelerometer, we asked participants to tell us the axis that changes the most when the accelerometer is rotated in each of the three directions. One participant noted how, as the accelerometer was moved, the values from all three axes tended to change, although at any point in time, one of the axes was typically changing the most. We asked our participants to design a highlight that plays when the sensor is placed flat on the table. One participant specifically questioned his prior understanding of an accelerometer. *''the accelerometer doesn't behave as I expected... and this is to do with a blind guy's 3D geometry''* (P2). The same participant, for example, expected the accelerometer values to be 0 when the sensor is placed on a table. Additionally, he learned that the sensor was noisy when he heard different values being announced when the sensor was placed flat on the table.

Display multiple concurrent streams (D5) Throughout the study, participants configured displays that streamed data from multiple sources. For tasks that required comparing data from multiple axes of the accelerometer, participants configured displays with note and speech outputs to sonify the different axes. For tasks that required participants to compare the temperature and humidity sensor, they configured multiple note handlers and outputs mapped to the left and right audio channels. Finally, our participants configured multiple handlers to the same sensor to understand *how* the sensor was responding, and *what* the value was. For example, our participant used both a note and extrema handler when trying to understand the light sensor; using information from the extrema handler he reported the change in value, and using information from the note handler he observed the sensors' sensitivity because its value was changing even when he had his finger on it.

In summary, our findings show that PSST can increase interest in physical computing or sensor-based programming among BVI developers. Additionally, we find that our experimental tasks demonstrate that PSST meets the design goals initially identified to support nonvisual debugging of sensor data.

4.5 Towards Real-World Adoption of PSST

Our evaluation of PSST with two BVI developers shows that PSST can help with understanding sensor data. The ability to easily and accessibly customize how data is presented differentiates our toolkit from other sonification libraries. We present limitations of PSST and our evaluation, and discuss the feasibility of integrating BVI-customizable dashboards into mainstream data programming tools. We close with a discussion on the feasibility of using large language models to drive toolkits such as PSST to create a dynamic data display from a plain language question about data.

4.5.1 Limitations

Though PSST provides a tool and foundation for BVI developers to customize their access to data, we do not yet know the full extent to which PSST can be used in an end-to-end physical computing programming scenario. We only included two developers in our study, making any quantitative analysis of impact impossible. However, the change in attitude reported in the participants before and after the study gives a strong indication that the toolkit is meeting real needs among at least some subset of BVI developers.

We did not validate or evaluate use of the toolkit API with BVI developers. Consequently, we were not able to assess things like the difficulty of adding novel handlers, outputs, and statistics.

Finally, PSST's library of handlers and outputs is still relatively small and would benefit from additional outputs drawn from the rich sonification literature (*e.g.,* (Holloway et al., 2022)) to provide well-studied interpretable ways of presenting data in audio. For example, live panning

and stereo output of all types of audio, including speech, would be helpful additions to PSST. Furthermore, PSST does not support the ability to go back in time, or replay values after a pause.

4.5.2 PSST Beyond Physical Computing

We designed PSST around anticipated needs to understand streaming physical computing sensor data by allowing BVI developers to customize how they want to access this data. As noted by our participants, however, PSST could be applied to a broad set of programming contexts such as telemetry, mobile computing, machine learning, and data science. Extending PSST in this way would require similar attention to integration that we demonstrated with the PSST dashboard and the micro:bit. Prior work suggests that assembling an accessible programming workflow requires careful balance of social factors such as being in sync with tools collaborators might use (Branham and Kane, 2015), technical factors such as accessibility of the tool itself (Albusays et al., 2017), and nuanced informational needs to perform a particular task (Pandey et al., 2021). Understanding the specific needs of other domains will improve the potential for PSST to support BVI programmers. An important next step would be to study these domains and properly customize PSST to interoperate with existing programming tools and processes within them. Recent systems such as the ChartReader (Thompson et al., 2023), Azimuth (Srinivasan et al., 2023), STRAUSS (Trayford and Harrison, 2023) and umwelt (Zong et al., 2024) show the promise of this future.

4.5.3 Driving PSST with Large Language Models

The PSST dashboard and tools provided by similar toolkits require a BVI developer to perform several interactions and take several decisions to create a useful data display to understand a dataset. Though this customizability is powerful, this may require some expertise and comfort with creating such configurations. Additionally, This added complexity could deter beginner use of end-user customizable data displays. To further simplify PSST use, we assessed the feasibility of using

function calling capabilities of large language models (capabilities with which these models can generate a list of functions and arguments from a schema) to create a data display from a natural language question. We however observed that general purpose models such as GPT4 were unable to generate coherent function calls to create meaningful data displays to explore even a simple data series. Though we were able to increase the probability of adherence to a spec by the use of tools such as the TypeChat library, the lack of standards or a large example set (both necessary to increase function calling accuracy) made the use of large language models to drive PSST infeasible with a potential for miminal to no utility to a BVI developer.

Future efforts could explore the creation of BVI-created examples of useful data displays to fine-tune large language models to support a text to interactive data-display interaction paradigm for data exploration and analysis.

4.6 Contributions

To the best of my knowledge, there was no toolkit prior to PSST configurable by BVI developers supporting access to live streamed data that allows developers to explore sensors and program them. We contribute such a toolkit, PSST. Our study shows that with the tools offered by PSST, BVI users may gain interest to engage in domains that they have been excluded from due to the lack of accessible tool support. Though PSST's design goals and our validation study focus on physical computing, it fundamentally makes both streaming and static data accessible to BVI developers, making the toolkit's capabilities applicable to a larger set of programming domains where the exploration and analysis of data is a critical skill.

In connection to my thesis statement, PSST contributes an *interactive* developer experience to provide access to static and real-time data, which is predominantly *communicated through visual artifacts such as graphs and charts* to equip BVI developers to understand analyze data, critical to perform tasks in several professional domains such as data science, machine learning, and the

design of human-centered machine learning systems.

The work described in this chapter explores the accessibility of physical computing and concludes with the contribution of PSST, a data sonification toolkit that makes streaming data accessible to BVI developers. This work led to a UIST 2022 publication (Potluri et al., 2022b). In the next chapter, I present work to examine the accessibility of Jupyter notebooks, a popular environment used by data scientists to share data, code and results. The results presented in this chapter will focus on the accessibility of data representations in these notebooks while offering evidence on a larger set of factors that impact their accessibility.

Chapter 5

DATA DRIVEN UNDERSTANDING OF COMPUTATIONAL NOTEBOOK ACCESSIBILITY

Computational notebooks such as Jupyter (Pérez et al., 2014) combine code, natural language, and rich representations of data providing a ubiquitous literate programming experience (Knuth, 1984). These notebooks are used through computational notebook systems and programming environments such as Jupyter, Google Colab, Datalore, and Noteable (among others), which abstract software setup, computational infrastructure management and access to resources. These computational notebooks have risen in popularity since the inception of Jupyter in 2014, impacting many domains within and outside of computer science such as data science, machine learning, and astronomy. Computational notebooks are widely used by data scientists as interactive mechanisms to process, understand and express data making it easier for them to collaborate, share code, convey stories and narratives through data visualizations and text, while keeping the reproducibility of results in mind (Rule et al., 2018; Wang et al., 2020). The popularity of Jupyter notebooks, as the go-to tool for data science can be seen in the rapid increase in published notebooks, 2.5 Million public notebooks hosted on GitHub in September 2018 (Rule et al., 2018), increasing by 10x since 2015 (Wang et al., 2020) and eventually increasing to over 10 Million (Guzharina, 2020).

The profound impact of Jupyter notebooks across domains has been recognized by the Association of Computing Machinery (ACM) in 2017 with a *prestigious software systems award* (Association for Computing Machinery, 2017). Despite computational notebooks being popular tools, we know very little about the accessibility of these tools for developers and data scientists who

are blind or visually impaired (BVI). An early 2023 analysis of the accessibility score for Jupyter Hub, a popular notebook offering, graded it as a fail (**F**) (Man From Jupyter, 2020). To address the lack of understanding of accessibility of notebooks, we present a data-driven investigation of the accessibility of computational notebooks. Our investigation focuses on accessibility for BVI notebook *authors* and *consumers* (hereby referred to as BVI users or BVI notebook users).

We conduct a large-scale analysis on 100,000 notebooks from an in-the-wild dataset of computational notebooks (Guzharina, 2020). Our dataset includes notebooks that may have been used for anything from exploratory data analysis to documents produced for public distribution. These notebooks could be written and consumed by users from a variety of backgrounds, including students, coders, or data scientists. All of these activities that support the different stages of notebook authoring should be accessible to any BVI notebook user . Thus, we chose to assess accessibility at scale without narrowing to a specific use task or professional domain. Our first of its kind large-scale analysis answers the question of *whether data artifacts, authoring experiences, and infrastructure to work with computational notebooks, are accessible to BVI users*.

5.1 Background on Notebook Accessibility

Limited efforts have been made by the Jupyter community to address or study the various accessibility challenges of data science tools. Most notable of these efforts is Astronomy Notebooks for All (Space Telescope Science Institute, 2023), an effort to perform accessibility audit of the Jupyter Lab interface and contribute changes to the upstream Jupyter open source community. Further, Microsoft's VisualStudio Code, a popular IDE among the BVI developer community, is actively improving the notebook authoring experience by building on the existing standardized format of notebooks through improvements in interactions with keyboard navigation and audio cues. These improvements however, do not contribute to our understanding of accessibility issues that can arise from the variety of ways in which computational notebooks are authored, consumed, and published.

Given the dearth in understanding the requirements for notebook accessibility, we draw knowledge from prior efforts making data visualizations (§2.3), web, and developer tools (§2.2.2) accessible - establishing an optimistic baseline understanding of notebook accessibility.

A critical aspect of using computational notebooks is to create, consume, and collaborate on visual, tabular, and other representations of data. These representations are often generated by computations performed in a Jupyter notebook. Therefore, in addition to enabling consumption, must support the creation of data visualizations for BVI developers and enable data driven story telling, therefore surfacing the need for these tools to offer capabilities to provide accessible data visualizations. Prior attempts to make data representations accessible to BVI developers, *e.g.,* (Potluri et al., 2022b; Sharif et al., 2022; Thompson et al., 2023; Zong et al., 2024) resulted in libraries with very limited functionality, leaving much to be discovered about their accessibility in the context of notebooks or the data representations shared in them. Prior efforts to provide guidance to describe graphs and make data visualizations accessible (Lundgard et al., 2019; Elavsky et al., 2022), provide actionable measures to assess the accessibility of data representations in notebooks.

Notebooks are authored in a web-based IDE such as Jupyter Lab and Jupyter book, or through hosted and managed alternatives such as Google Colab, and Datalore. Since their invention, millions of such notebooks have been authored for data analysis and related tasks, and it is important that we analyze these phenomena (Wang et al., 2020) in the context of accessibility. Although many notebooks are used primarily by their authors, some notebooks are published. Publishing a notebook leverages tools built into the notebook IDE to generate a webpage, or sometimes a PDF or LaTeX document. In the case of web pages, these tools use web semantics such as headings and tables to structure content, and allow notebooks to be themed or otherwise decorated, making Web semantics and styling central to the notebook experience. Thus, coupling the findings from a domain specific tool leveraging web as a platform like Jupyter, with web accessibility helps us identify potential accessibility concerns that may be unique to notebooks.

Some prior studies use web accessibility guidelines to examine and improve the accessibility of other domains. For example, (Elavsky et al., 2022) extend web accessibility guidelines to make data visualizations accessible. Similarly, (Li et al., 2021) use IBM's accessibility checklist—a set of accessibility guidelines derived from web accessibility guidelines, to compensate for the lack of industry standards to examine the accessibility of high-fidelity prototyping tools. Web accessibility analysis of Jupyter notebooks would contribute to this body of work.

Finally, glanceability has been known to pose access barriers to BVI developer tool users. With a web-based developer and consumption experience provided by Jupyter, the notion of glanceability could manifest as proper use of web semantics. Combining web accessibility with requirements to make notebooks glanceable results in a possibility to assess notebook glanceability.

Our literature review highlights several areas in which accessibility problems might arise, including Visualization and data table access, both of which come up frequently in notebooks; and navigation and glanceability during coding, which would be relevant to the notebook authoring experience. We now describe our analysis pipeline to understand accessibility concerns with notebooks in the areas highlighted in prior literature and present our results.

5.2 Studying Notebook Accessibility

Our study focuses on an at-scale assessment of the accessibility of the consumer and authoring experiences. We choose this approach because of its high ecological validity: while a user study must narrow scope to a very small set of notebooks, possibly hand curated to be semi-accessible so that the study is not a waste of participants' time, a large scale study can explore a much broader range of notebooks. Below we introduce our study approach and sampling strategy. We also discuss the metadata that we extract from notebooks to prepare for our analysis of results.

While there are several datasets of Jupyter notebooks available (Rule et al., 2018; Quaranta et al., 2021), with metadata about where they come from and how they are created, they do not fully

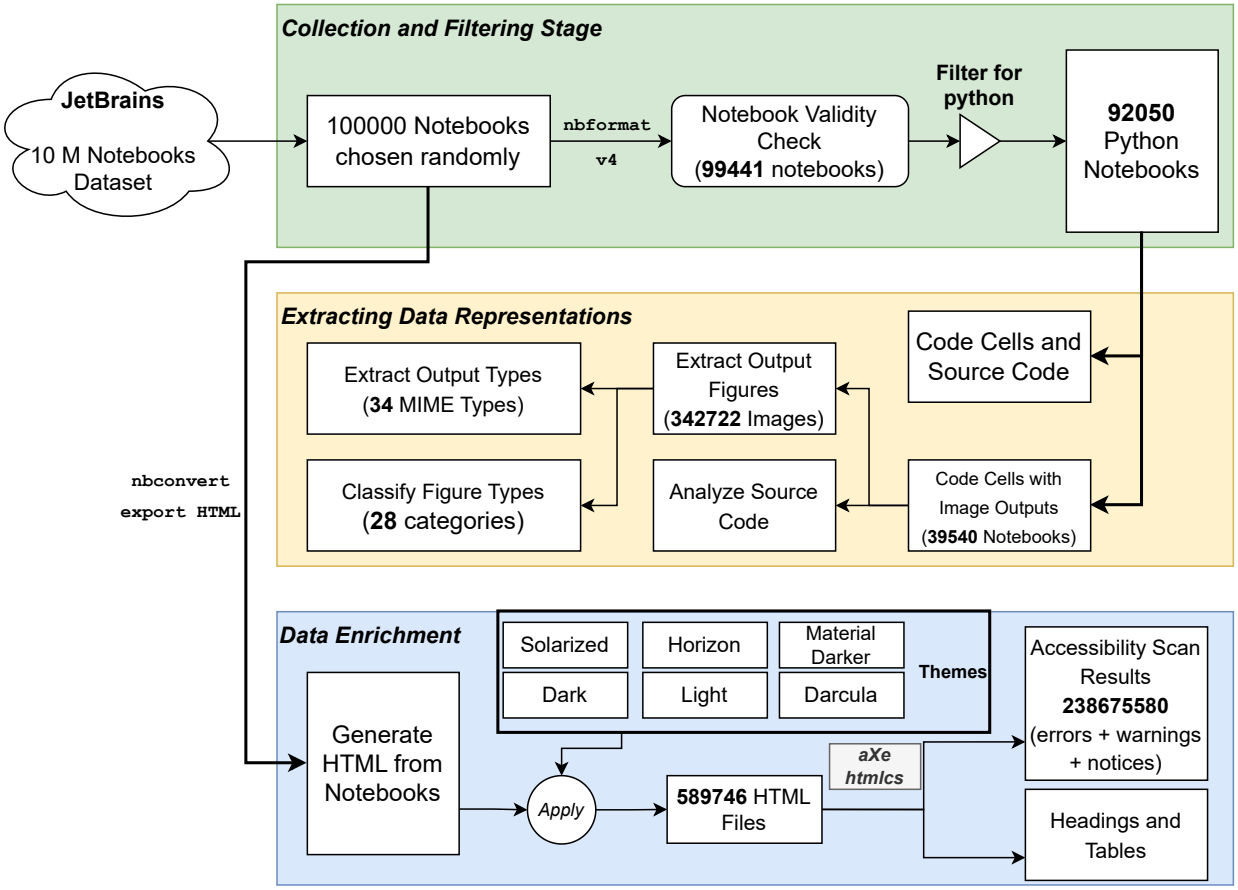


Figure 5.1: Flowchart Diagram indicating the Data Processing Pipeline presented in section 5.2.1, section 5.2.2, and section 5.2.3

capture the variety of contexts that computational notebooks could be used in. Further, accessibility issues can occur in notebooks irrespective of context, and tools should inherently support the creation, consumption, and distribution of accessible notebooks.

We begin by detailing our data processing pipeline, including our approach to sampling. We then explain our method of measuring accessibility. We defined our accessibility metrics to prioritize an optimistic upper bound (meaning metrics that have high sensitivity but low precision), because there does not currently exist a validated measure of automatically measuring true accessibility. As we will see, this approach ultimately allows us to confidently say that most in-the-wild notebooks are inaccessible (hinting at the fact that the situation is potentially even worse than we estimate). Put differently, our approach has lower *construct validity* than a user study might have. To complement this automated measurement and analysis of accessibility, we also conducted experiments to manually verify notebook glanceability through screen reader testing.

We build our data processing pipeline (detailed in fig. 5.1) to (1) collect and filter a randomized subset of the notebooks from the larger JetBrains dataset (§5.2.1), (2) extract the required data representations from the filtered notebooks (§5.2.2), and (3) enrich the notebook data through analysis of transformed representations, used in the notebook distribution process (§5.2.3).

5.2.1 Data Sampling and Filtering

We start with the dataset provided by JetBrains that contains 10 million Jupyter notebooks (Guzharina, 2020). Because of the computational and time costs of analyzing a data set of this size, we began by analyzing a sample of 10000 randomly chosen notebooks from the 10 Million notebook dataset. We start with this random sample of 10000 notebooks to test our analysis pipelines, and gain an understanding of the characterization of the dataset. After establishing the required analysis pipelines, we scaled our analysis by 10x and obtained a new random subset of notebooks resulting in 100000 notebooks. We observed that the results from our analysis pipeline returned similar

observations in both the 10000 and 100000 notebook analysis, giving us the confidence that 100000 was a sufficient sample size to draw conclusions from. Therefore, we stopped our analysis without further scaling the number of notebooks in our dataset. By chance, there was overlap of 87 notebooks between these two data sets, which we considered small enough to be inconsequential and retained all 87 overlapping notebooks for our analysis.

It is likely that some of the notebooks in our dataset may be intended for scratch use, or exploratory data analysis which might be inaccessible compared to the presentation-ready notebooks. Since our work intends to explore accessibility for BVI authors, as well as consumers, including these notebooks in our study is intentional. Only studying notebooks that are presentation-ready assumes BVI people's involvement only as consumers of these notebooks and limits discovery of the extent of notebook accessibility problems.

Notebook Validity Check

The first step of the pipeline involves coercing the computational notebook files to the latest v4 specification of the Jupyter Notebook format using the `nbformat` tool to ensure validity (Jupyter Development Team, 2023). A notebook obtained in the dataset is considered as valid if the file has correctly formatted JSON content according to the specified Jupyter notebook format. This conversion resulted in a total of 99441 notebooks filtering out 559 notebooks in the process due to conversion failures which typically occurred because of older and incompatible notebook versions.

Filtering for Python Notebooks

Building on previous works on notebook analysis that have established Python as the most popular language used in computational notebooks (Guzharina, 2020; Quaranta et al., 2021), and filter for notebooks written in Python in our first step. As a result, we obtain 94722 notebooks, and further remove 2672 which do not contain the language metadata, resulting in 92050 notebooks.

5.2.2 Extracting Required Data Representations

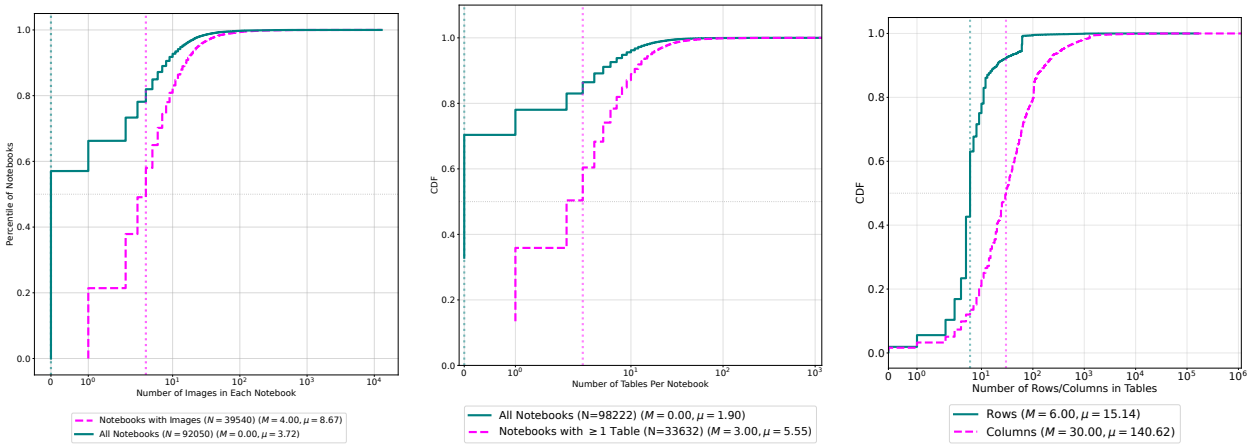
Computational notebooks store source code in 'code cells' and the outputs from the execution of the code as its children formatted as 'output cells'. Additionally notebooks also support the usage of markdown to display text which is formatted as a 'markdown cell'. These cells cannot contain child attributes related to outputs as per the notebook format specification. We process each notebook and extract information about (1) source code, and (2) outputs.

Code Cells from Notebooks

The next stage of our data processing pipeline extracts information about the source code from 'code cells' in the 92050 python notebooks. We extract the source code and markdown text, in addition to computing the number of code and markdown lines and cells in a notebook. Our analysis identifies 39540 notebooks where at least one source code cell generates a graphical output into its corresponding output cell. We removed 52509 (57.04%) notebooks that only contained source code, and no accompanying outputs for the code segments from our analysis, leaving 39540 notebooks for the next stage of the pipeline.

Output Figures

The Jupyter notebook format stores figures generated as a part of the code output as `base64` encoded strings with the metadata information of the corresponding Multipurpose Internet Mail Extensions (MIME Types) to identify the type of media output. By default, the notebooks support five image media mime types indicated by `image/bmp`, `image/gif`, `image/jpeg`, `image/png`, and `image/svg+xml`. Other media output types such as PDF generated from code are immediately converted to display as PNG format in a notebook with no additional extensions installed. We parse through all the 39540 notebooks since these are the python notebooks containing at least one programmatically generated graphic in an output cell corresponding to a code cell.



(a) CDF of number of plots in each notebook.

(b) CDF of number of tables in notebooks.

(c) CDF of number of table rows and columns.

Figure 5.2: Characteristics of the Notebooks in the Analysis. Plots show a cumulative distribution function (CDFs). The teal line in the CDF indicates the relationship between the 92050 valid python notebooks in our study and the number of plots they contain. The magenta line indicates the distribution of 39540 notebooks which contain at least one image. The step nature of the curve compared to the continuous distribution is because of the discrete number of plots which could be included in a notebook. The vertical length of the line at each x value indicates the percent of those notebooks in the dataset.

We convert the encoded base64 strings into image files and store them for further analysis resulting in 342722 total images. As summarized in fig. 5.2a, about 42.95% notebooks have at least one programmatically generated figure. For notebooks that have such figures, the median number of images in them is 4.0 . 10% of the notebooks contain over 16 images per notebook, with a long tail where 1% of the notebooks contain over 77 images. The notebook with the most images contains 12858 images.

Analysis of Code Syntax

The next step in our data processing pipeline targets the contents of the source code in the 39540 python notebooks that contain at least one programmatically generated graphical output. We chose Python because it is the most popular language used in notebooks and is studied in other large scale analyses (Rule et al., 2018; Choetkiertikul et al., 2023). This allows us to perform language specific code analysis by delving deep into the library and module ecosystem of the programming language, to understand accessibility gaps in popular tools, and make actionable recommendations to improve the accessibility of these notebooks.

We use the abstract syntax tree (`ast`) module in python to parse the contents of the source code in these 39540 notebooks. We extract information about modules and functions being imported by notebook authors, and the functions being invoked within various cells of the notebook. This gives us information about the different libraries, and function calls frequently used by notebook authors.

Constructing the abstract syntax trees however is not as trivial as combining the code cells in a notebook before running a parser, and requires additional processing of the code in the notebooks. Notebook systems support Jupyter *magics*—a functionality provided by the kernel, that allow developers to call functions that would simplify some Jupyter operations. For example, the IPython kernel used by Jupyter, allows developers to use the `%% latex` command to render a cell in LaTeX and the `%% bash script magic` command to run a cell in bash as a subprocess.

Table 5.1: Top 5 Output Types in Notebook Account for 98.67% of outputs

Rank	Category	Output Type	Total	Percent	Cumulative
1	Text	Plain	977328	61.72	61.72
2	Image	PNG	339589	21.45	83.17
3	Text	HTML	208170	13.15	96.32
4	Application	Javascript	22842	1.44	97.76
5	Application	Jupyter JSON Widget	14532	0.91	98.67
Total Number of Outputs			**1583255**		

While these are valid operations in a Jupyter environment, they are not a valid part of the syntax of the python programming language and result in errors when parsing the syntax to construct an abstract syntax tree, even if the rest of the code in the cell is valid syntax. Therefore, constructing abstract syntax trees requires the source code snippets to be processed to remove any magic lines. We removed source code lines which begin with the `%` (percentage) special character, in addition to lines starting with `!` indicating the execution of a shell command, or ending with the help `?` operator. We run our parsers to construct the AST over the resulting code and extract information about the functions, and modules imported in the notebooks.

Output Types

The extensibility of Jupyter notebooks allows notebook authors to customize the story telling and consumption experience of the notebooks. To understand how these customizations impact accessibility, we extracted three high level categories (*application*, *image*, and *text*) based on MIME types used in various cells of the notebooks. The top 5 output types presented in table 5.1 account for 98.67% of the outputs in the notebooks. Notebooks included 24 different application types; 6 text output types (*e.g.,* HTML, LaTeX, markdown, etc.,), and 4 image output types. Portable network graphics (PNG) images are the most commonly used image output formats making up 21.45% of the outputs in the notebooks.

Figure Types

To understand what type of figures are created in the notebooks, we classify the figures into 28 different image type categories including Line Chart, Histogram, Box plot, Confusion matrix, Scatter plot and others (the full list can be found in fig. 5.3b). We classify the 342722 images obtained from our figure extraction phase of the pipeline, using a Fully Connected - Convolutional Neural Network (FC-CNN) combined with a Fisher-Vector Convolutional Neural Networks (FV-CNN) (Song et al., 2017) pretrained on the DocFigure dataset (Jobin et al., 2019). We run this inference on an AWS `p2.16xlarge` virtual machine configured with 16 NVIDIA K80 GPUs, 64 vCPUs, and 732 GB of memory.

5.2.3 Data Enrichment

The user experience to consume and author Jupyter notebooks often takes place through web interfaces. Developers customize their development environment experiences through themes, and many IDEs in addition to their defaults, provide a wide variety of themes—catering to developer preferences. Often these same themes are carried to published notebooks.

Generating HTML from notebooks

Since different themes can vary in their accessibility, we selected 6 popular themes including the defaults provided by Jupyter for publishing HTML exports. Our selections include *solarized*—a theme (originally written for Vim and made available now by various IDEs) (Schoonover, 2023), *darcula* (for ZSH originally, and used extensively by JetBrains) (Rocha, 2013), *horizon* (default by VSCode) (Mohirio, 2022), the *material darker* theme (Mirosa, 2022), and the default *dark* and *light* theme options supported by Jupyter (Pérez et al., 2014). To assess the accessibility of notebooks in these web interfaces, we generate HTML exports of notebooks in our dataset with these six popular themes applied using the standardized `nbconvert` tool.

We use the open source `nbconvert` tool that supports converting notebooks into publishable HTML and other formats. `nbconvert` allows users to select themes and specify other parameters to control the generation of HTML output and is the de-facto tool integrated into computational notebooks providing various export format capabilities. Once the conversion is complete, we export the notebooks into standalone HTML files which we then serve through a web server. All 100000 notebooks were exported as HTML, producing 589746 HTML files, 98291 per theme.

Accessibility scans

HTML user interfaces are typically evaluated using accessibility testing and evaluation engines which are software programs that evaluate the content, design of the interface, and check their ability to satisfy various established accessibility guidelines. We perform accessibility scans using aXe and HTML Code Sniffer (HTMLCS) together by deploying a self hosted version of the *pa11y* accessibility scanning infrastructure configured to use both engines (Manning et al., 2023).

For all 589746 HTML files across all themes, we extracted the type of violation (*error*, *warning* or *notice*), the accessibility engine that detected it (Axe or HTMLCS), the specific code corresponding to the violation provided by the engines, and the selector (HTML node where the violation was detected). We enrich our dataset by attaching this information to the name of the notebook and theme being tested, identifying 238675580 combined errors, warnings and notices across notebooks.

Web Semantics

While accessibility scanners provide information about standardized accessibility errors, they do not provide more nuanced information such as the size of tables and the presence of headings that can be critical to understand the screen reader glanceability and navigability of notebooks. To gain this understanding, we use LXML, a library that enables efficient parsing of the HTML DOM tree, to process the HTML outputs of notebooks. We picked the light theme and used the 98291

HTML files corresponding to it for this processing. We extracted information about the type of cell, presence of images, alternative text (using the alt attribute of the `<img>` tags), information about tables (using the `<table>`, `th`, and `td` tags), presence of links (using the `<a>`), and various heading levels (using the `<h1 – h6>` tags) along with the file name, and cell location in the file.

Our observations indicate that 65.9% of the notebooks do not contain any tables in their outputs. Among the remaining, a notebook contains 3.0 (median) / 5.55 (mean); however the distribution has a long tail with the maximum at 1181 tables and 1% of the notebooks containing over 37.0 tables. Among the 34.1% of notebooks in our dataset which contain tables, we extract additional metadata to identify the structural shape of the tables. The median table rendered in the output cells of the notebooks contains 6 (median) / 15 (mean) rows, and 30 (median) / 140 (mean) columns. Figure 5.2c indicates the distribution of the number of rows and columns for all tables identified in our dataset. The largest table in the notebooks in our dataset contains 162736 rows and 1139145 columns indicating a maximum of 185379900720 cell values.

5.2.4 Manual Screen Reader Testing

To investigate a sample of accessibility issues identified by our automated testing, and to verify if notebooks were glanceable, two research team members experienced with using a screen reader opened a selection of notebooks in screen reader and browser combinations, and verified if a screen reader user will be able to get to all headings, images, and tables present in those notebooks. As we noticed the BVI researcher's screen reader crash several times during the exploratory phase of our research, we hypothesized size of the notebooks to play a role in causing these crashes. We identify a list of notebooks meeting different size criteria based on the percentiles of the file sizes in our dataset. While one researcher (also a BVI screen reader user) performed the tests and reported the counts in each notebook, the second researcher visually verified the reporting, noting the observations. We performed these tests with Microsoft Edge, Google Chrome, and Firefox

Nightly with their accessibility cache improvements enabled (Dotzler, 2022) with JAWS and NVDA 2023 on a Windows computer running Windows 10. We performed our VoiceOver tests on a Mac with VoiceOver using both Safari, and Chrome. We did not test our notebooks with Firefox on the Mac since the accessibility cache improvements for Firefox were not yet available for the Mac OS. Similarly, we did not test using Safari on Windows since Apple ended support for it in 2015.

5.3 Results

Recall that the goal of our investigation is *to understand the accessibility of data artifacts, authoring experiences, and infrastructure to work with computational notebooks to BVI users*. Given the overall concerns about visualization and data accessibility discussed in section 2.3, we must understand how accessible these are in notebooks. Thus, we begin our analysis by exploring how accessible the data artifacts in notebooks are to BVI users. Relatedly, given the importance of navigation, and information seeking/glanceability to IDE accessibility, it is critical that we assess this in notebooks as well. Thus, our analysis explores the proper use of headers and other landmarks that improve navigation and glanceability for screen reader users.

Notebooks are not just a programming tool, they are also a communication platform that can be customized and themed to generate final ‘‘documents’’ in various formats. If these documents are not accessible, the consumer experience will be impacted. Thus, our analysis investigates the impact of the tools used to customize and distribute notebooks on their accessibility. to develop an understanding of notebook accessibility, we developed metrics which represent optimistic *upper bounds*, meaning that the presence of accessibility in notebooks is likely much smaller than our findings in this paper indicate.

5.3.1 Accessibility of Data Artifacts

At the heart of data analysis is the data, and that is typically explored through a combination of text summaries, graphical representations, and tables. These latter two artifacts -- *graphics* and *tables*, that are essential to the data story telling process used by notebook authors, may have important implications for accessibility. A truly accessible notebook should support advanced, accessible and dynamic visualization techniques (Kim et al., 2021). However there are common, basic requirements for making static images and charts accessible (Elavsky et al., 2022), which guided the development of our optimistic metrics for evaluating artifact accessibility.

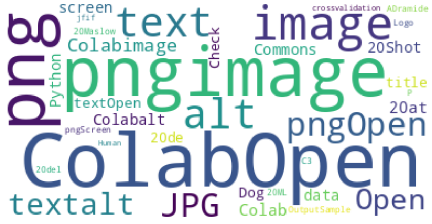
Presence of ALT text : A meaningful *ALT* text attribute should accompany visualizations. We measure the presence or absence of ALT text in programmatically generated images and analyzed the alt-texts found in images that may have been manually added by notebook authors. This is an optimistic measure because the presence of ALT text does not mean it is descriptive or helpful for accessibility. Only .19% of images in our data have ALT text whose word frequency we measure and present detailed results of in Section 5.3.1. We do not measure ALT text quality.

Figures followed by tables : Without ALT text, a figure can still be somewhat accessible if it is followed by a table with the equivalent data that is visualized in the figure. This is optimistic because tables after figures, though present, may not be accessible, or may not be related to the figure. We present the detailed results in Section 5.3.1.

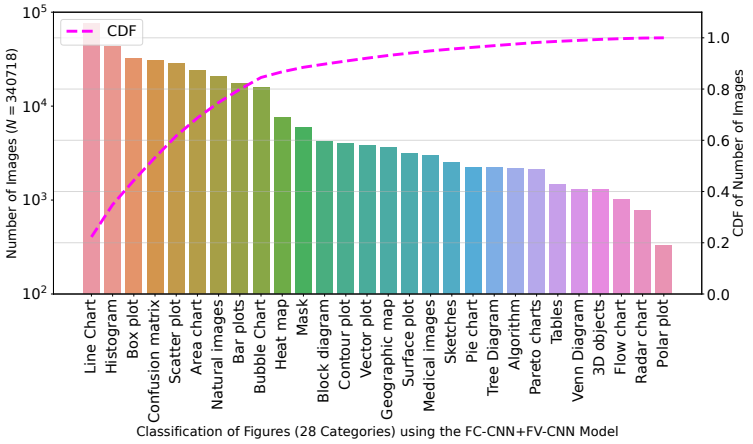
ALT text for Static Images and Charts

Static images and charts are present in 42.95% (39540) of the 92050 python based notebooks, most of which are PNG files (Table 5.1). Notebooks with these artifacts contain a median of 4.0 figures as shown in Figure 5.2a. We consider a figure to be programmatically generated if the code cell in the notebook contains a mapped output section indicating the result of the execution of the cell and

Figure 5.3: Presence of images in notebooks



(a) Word cloud of Image ALT-Texts in Notebooks.



(b) Classification of 342722 images used in 39540 notebooks into 28 categories.

Figure 5.4: Types of visualizations used in notebooks and analysis of most frequent words used in alternate text descriptions.

containing an image type. The vast majority of the programmatically generated images (N=342102 (99.81%)) do not have associated alternative text. Of the 609 containing the alt text information in the notebooks, only 1 image is programmatically generated from code. We investigate the one image with alt text attribute generated programmatically in the notebook but can not reproduce the result.608 images have alt texts specified through the `description` attribute in markdown images, included using the syntax '`![description](path_to_image)`'. In Figure 5.3a, we present a word cloud of the alt descriptions we found indicating that they were mostly meaningless in their current use. The most dominant words 'Open' and 'Colab' come from the markdown included by the interaction button for opening notebooks with Google Colab, which is included by some notebook authors when publishing their notebooks. The word cloud also indicates poor usage of alt text whenever it is used in markdown with most descriptions lackadaisically referring to the graphic included as 'image', 'png', or 'alt'.

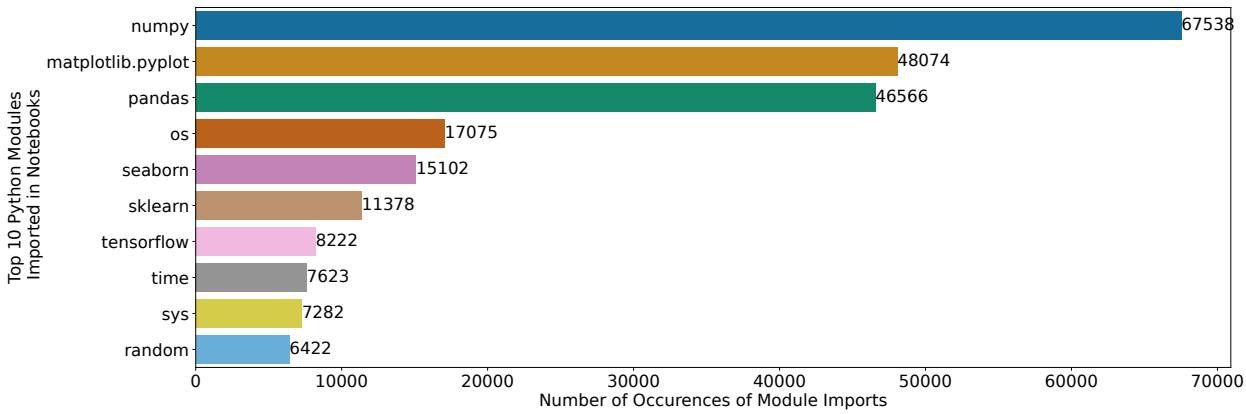


Figure 5.5: Top 10 popular python modules imported in notebooks ranked by their usage frequency

Table 5.2: Top 10 Most Invoked Function calls in Python Notebooks with Figure Outputs (excluding built in functions)

Rank	**Rank (incl. built ins)**	**Function Targets**	**Module**	**Function Call**	**Count**
1	4	Visualization	`Matplotlib`	`plt.plot`	70575
2	5	Visualization	`Matplotlib`	`plt.show`	70454
3	6	Visualization	`Matplotlib`	`plt.title`	49998
4	8	Data	`numpy`	`np.array`	46360
5	9	Visualization	`Matplotlib`	`plt.figure`	45797
6	10	Visualization	`Matplotlib`	`plt.xlabel`	42430
7	11	Visualization	`Matplotlib`	`plt.ylabel`	41693
8	12	Data	`pandas`	`pd.DataFrame`	32144
9	13	Data	`pandas`	`pd.read_csv`	30858
10	14	Visualization	`Matplotlib`	`plt.legend`	23228

Types of charts and libraries used Understanding the types of charts and figures most used by notebook authors can help guide and prioritize research in accessible visualization for data analysis. The usage of the pre-trained FC-CNN+FV-CNN classification model (Section 5.2.2) over the programmatically generated images found in our data (N=342722) reveals line charts as the most common type of programmatically generated visualization (22.26%), followed by histograms

(12.61%), and boxplots (9.45%). Figure fig. 5.3b contains a histogram of the most popular types of figures found in the analyzed notebooks ranked from most widely used to the least widely used. Seven types of charts make up roughly 75% of the figures in the dataset: Line chart, histogram, box plot, confusion matrix, scatter plot, area chart, and natural images. Some of these are understudied in the accessible visualization literature, which primarily focuses on bar, line, and pie charts, and scatter plots (Kim et al., 2021). Histograms, box plots, confusion matrices, and area charts represent important areas to make more accessible in future research.

We also analyzed the python `import` statements and function call invocations used in the 39540 notebooks with images, to identify the most popular charting library used by notebook authors (`Matplotlib`, followed by `seaborn` as shown in Figure 5.5). The seaborn library is an enhanced wrapper over the underlying Matplotlib libraries. Additionally, through code analysis and tracing the function calls we identified the most popular functions used by notebook authors and present them in Table 5.2. Seven of these 10 popular functions produce visualizations through the Matplotlib module, while the others target data processing through modules like `NumPy` and `Pandas`. Given this, we investigated Matplotlib's capabilities to understand support for alt text. However, despite its importance in the python community and extensive use in computational notebooks, scientific publishing, and other media, Matplotlib does not support embedding alt text descriptions for the generated graphics. Even with static figures, it is possible to include ALT text -- PNG, the most popular image format standard used by authors (Table 5.1), supports embedding image descriptions in metadata.

Comparative Ordering of Tables and Figures

Unless a chart's summary, context, and alt text convey all relevant information contained in the chart, which our investigation of ALT text shows is clearly lacking in this data (Section 5.3.1), a data table or data summary is a necessary accompaniment to that chart (Elavsky et al., 2022). With

the use of data processing libraries like `pandas`, it is relatively straightforward to also render a slice of the dataset represented in a chart as an HTML formatted table, or to provide a description in the surrounding code or markdown cells. To estimate the prevalence of these accessibility best practice guidelines, we look for code cells that programmatically output an image and filter those which (1) do not contain a heading immediately after the current output cell -- since they typically indicate a change in the context (Choetkiertikul et al., 2023), (2) have a markdown cell before or after the current cell potentially indicating necessary context, or (3) have a cell with a table (either programmatically generated or via markdown) immediately before or after the cell with the image.

Of the 208427 code cells across all notebooks that contain images, we find that 159341 (76.44%) cells meet our criteria for possibly being related to the image. Of these, 59166 (37.13%) cells contain markdown text (indicating possible explanations of the image) and 13037 (8.18%) cells contain a table. We further evaluate these results to understand their implications for accessibility.

The 13037 cells surrounded by a data table in the neighboring cells are found in 7109 *notebooks* in our dataset, while the 59166 cells containing markdown text after an image are present in 19028 notebooks. The existence of markdown cells in the neighboring cells but no heading in the cell immediately after however does not immediately imply relevance to the figure generated in the current code cell since they could also contain markdown included images. The existence of both markdown content, and supporting tables indicate the most accessible representations of the image -- containing both the relevant tables and a description.

Only 2566 (1.23%) cells in 1795 (4.53%) notebooks with figure outputs meet this criteria and contain both markdown and tables in their neighboring cells making those cells relatively more accessible when navigating and attempting to understand the notebook.

Simply including tables after charts is a low bar. It is also important that those tables are usable with a screen reader. Despite the inclusion of tables in the notebooks, too many rows and columns can affect screen reader navigation, resulting in users losing context or requiring too many key

strokes to skip the tables or interact with elements in the table. While there is no defined threshold for the maximum number of rows or columns, screen reader users typically lose context when navigating large tables (Wang et al., 2022). The median row count (6 rows) is identical to the number of default rows printed by `pandas.head()` (one header row, and 5 data rows). The number of columns however grows rapidly as shown in Figure 5.2c. For example, the largest table in our dataset contains 1139145 columns and 162736 rows and may be impossible to glance with a screen reader. An average table present in the notebooks in our study contains 140 columns and 15 rows resulting in over 2100 cells indicating the very high number of keyboard interactions needed by BVI users to understand and navigate the tables.

In summary, we find that images and tables–data artifacts found in notebooks–may not contain the necessary information for them to be accessible to BVI notebook users. The presence of tables and text descriptions indicate the possibility for these data artifacts to be accessible and open up avenues for future investigations. However, notebook authors following this accessible practice need to also provide descriptions and use tables with sizes considering screen reader accessibility.

5.3.2 Navigability and Accessibility for Authors

An understanding of the practices of notebook authors can guide our approach to making notebooks more accessible. Current understanding of code glanceability challenges for BVI people (*e.g.,* (Potluri et al., 2018)) does not account for code representations which are interwoven with rich representations of data and web semantic structures -- such as headings and tables supported by computational notebooks. Since Jupyter notebooks support markdown, a markup that is rendered using web semantics, authors have a great deal of control over how structural information is conveyed in the notebooks. For example, notebook authors control presence of semantic elements such as headings, tables, links, and figures. Screen readers typically support single key navigation among headings of different levels, links, tables, and many others (NVAccess, 2023).

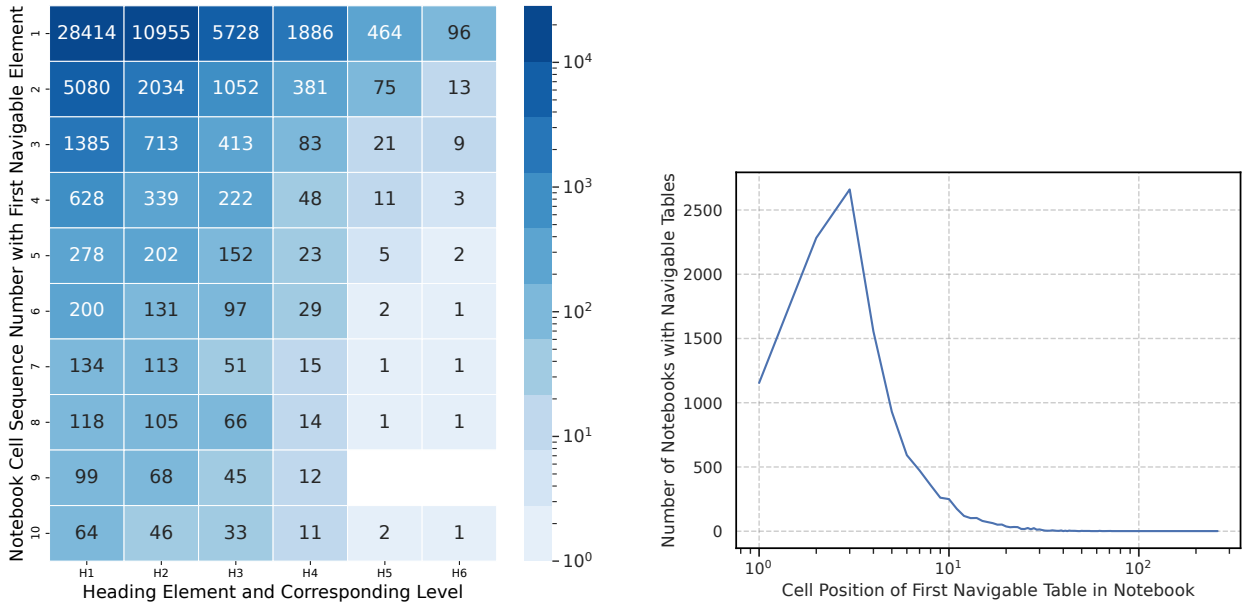
This method of navigation supported by screenreaders allow BVI screen reader users to skim through a notebook, and quickly understand its structure. To explore notebook glanceability through web semantics , we define the following optimistic estimates of accessibility:

Navigability of Notebooks: One simple but important aspect of proper heading use, is to use a level 1 heading (H1) in the first cell of every notebook, enabling screen reader users to easily find the start of notebook content (Fast, 2023b). This is an optimistic estimate because the heading should contain text that is relevant to highlighting the topic of the notebook, and it is only one of several rules that should be followed to properly support navigability with headers. We present detailed findings in Section 5.3.2.

Finding landmarks: *Tables* are an important and accessible way of representing data to a screen reader user, especially given the lack of ALT text uncovered in visualizations. They also make it easy to quickly gain some insights about a notebook, because screen reader users can jump through tables using the single-key navigation similar to headings. Thus, a second simple and optimistic metric is whether a notebook has a table in it. Only 34.1% notebooks have data tables and we present detailed results in Section 5.3.2.

Navigability of Notebooks

To assess navigability, we calculate where the *first* heading element lies in each notebook, as well as what type of heading it is (H1 through H6). In Figure 5.6a, the left most column (from top to bottom) shows the number of notebooks with an H1 in cell 1, cell 2, and so on down to cell 10. A majority (28414 (28.90%) notebooks) have a heading level 1 (`H1`) in the first cell. We speculate IDE integrations, JupyterLab, and Google Colab's default behavior, which adds an H1 in the first cell automatically, may have resulted in this accessibility advantage. As the first occurrence of a heading moves further away from the first cell indicated in Figure 5.6a, it is likely that screen reader



(a) First Navigable Heading Element

(b) Position of First Navigable Table in Notebooks

Figure 5.6: Presence of first navigable heading and table elements in the Notebooks. **Left (fig. 5.6a):** shows the cell position of the first heading element and its level (x axis) from the 1st cell in the notebook to the 10th (y axis). **Right (fig. 5.6b):** shows the cell position (x axis) of the first table present in notebooks (y axis) with a navigable table (excluding notebooks with 0 tables).

users will skip a number of useful cells in the notebook. Screen reader users may also be confused by or skip cells when authors use a different heading level for a first heading in cell 1, breaking the typically expected structure of the notebook or web based interactions. In Figure 5.6a, we see that 10955 (11.14%) notebooks start with an H2 (row 1, column 2), 5728 (5.82%) with H3 and so on down to H6 (.09%). 47543 (48.36%) notebooks contain a heading of *any* level in the first cell of the notebook and 59.67% (28414) of them correctly match our expectations to have a heading level 1 in the first cell. The high percentage of notebooks containing a heading indicate the high potential for improving accessibility of notebooks through improved navigability for screen reader users. The subsequent rows indicate the number of useful code cells potentially skipped by the screen reader navigation and show the location and frequency of occurrences of the first heading.

Finding Landmarks: Tables

We found that 33517 (34.1%) of the notebooks have tables in them. Among these, a notebook contains 5.55 tables on average, with 3.0 tables at the median, and the top 1 percentile of notebooks contain at least 37.0 tables as shown in fig. 5.2b. The lack of tables in 64774 (65.9%) of notebooks, consequently reduces the accessibility of notebooks and the possibility of glancing at data.

The first occurrence of tables is typically at the third cell in the notebook as shown in fig. 5.6b, perhaps because the first two cells are typically used for importing the necessary modules, and loading the required datasets for further analysis. This raises a further optimistic aspect of our metric -- such tables may be primarily present to check that data loaded correctly, rather than highlighting results after analysis is complete, limiting their utility for screen reader users.

5.3.3 Tooling Impacts on Notebook Accessibility

The accessibility of published notebooks on BVI consumers is impacted not only by authorship, but also by the tools used to export them. Jupyter notebooks allow the developers to export the notebook into various formats such as PDF, LaTeX or HTML, among many others. The HTML format is widely used by notebook authors when releasing their notebooks because of the ease of sharing content over the web. Many popular code hosting repository tools like GitHub or GitLab use the submitted raw `ipynb` notebook format files and convert them by exporting them into HTML when navigating to the file using a web browser. Thus we focus our analysis on HTML renderings. We use a mix of manual testing and automated accessibility testing tools to assess this. We measure:

Reachability of structural and graphical information: As mentioned in Section 5.3.3, the ability to navigate to structural elements is important to accessibility. While our previous analysis looked at whether semantic information is properly included in notebooks by authors (Section 5.3.3 and Section 5.3.2), here we test the same question once notebooks have been rendered. To assess this,

we performed manual screen reader testing, the only metric that we tested at small scale (only for 10 notebooks). This is an optimistic estimate because we did not test whether the landmarks were useful, only whether a screen reader user could get to them. We find significant concerns with the scalability of notebooks for screen reader users affecting navigability in 6/10 cases and present the detailed results in Section 5.3.3.

Impact of Theme Choice on Accessibility (Section 5.3.3): This metric captures the accessibility impact due to the choice of theme when generating a notebook. Many themes are also used during authoring, so this may also impact authorship. This metric is optimistic since it may not capture all errors introduced by a theme due to the use of automated testing, which is known to be an optimistic measure of website accessibility (Mankoff et al., 2005). The best theme we tested differed from the worst by 84.95% overall; however we found that different themes performed differently on different accessibility testers and raised different types of errors within those testers.

Reachability of Structural and Graphical Information

Table 5.3 shows the results of manual testing of the accessibility of notebooks. We sampled notebooks of a representative range of sizes as we observed that large notebooks were crashing browsers during the exploratory phase of this research. Through the sampling based manual analysis, we identified that notebooks of large sizes can cause accessibility breakdowns, crashing screen readers and browsers on windows, indicated by $\perp$ in Table 5.3. The results suggest that at least 1% of all notebooks are fully inaccessible due to their large sizes and cause screen readers or web browser tabs to completely crash. We speculate that the difference in how JAWS and NVDA read webpages through a virtual buffer *vs* VoiceOver's ability to directly interact with the browser causes the difference in accessibility breakdowns visible in the table (Zehe, 2017). Addressing these breakdowns in windows screen readers is critical due to their popularity among the BVI community.

Table 5.3: Accessibility evaluation on randomly chosen notebooks of variable sizes (in bytes) ordered by percentile rank (P10-P100). For consistency we chose the *light* theme in our evaluations. Only VoiceOver is shown for Safari, since the browser is no longer actively supported on Windows. √ indicates notebooks which pass the accessibility evaluation for glanceability, ⊖ represents those which are functional, but not fully glanceable, ✗ represents those which fail glanceability evaluation for multiple reasons, and ⊥ represent the notebook which cause complete crashes of screen readers or browser tabs.

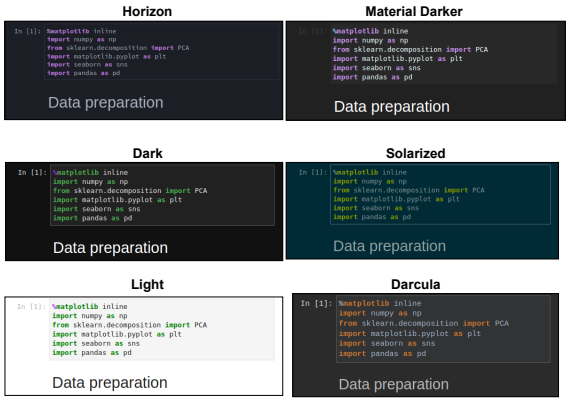
	Rank	**Size (Bytes)**	**Microsoft Edge**		**Google Chrome**			**Firefox Nightly**		**Apple Safari**
			NVDA	JAWS	NVDA	JAWS	VoiceOver	NVDA	JAWS	VoiceOver
N1	1 (P10)	630352	√	√	√	√	√	√	√	√
N2	2 (P25)	634340	√	√	√	√	√	√	√	√
N3	3 (P50)	721056	⊖	⊖	⊖	⊖	√	⊖	⊖	√
N4	4 (P75)	997861	⊖	⊖	⊖	⊖	√	⊖	⊖	√
N5	5 (P85)	1362512	√	√	√	√	√	⊖	⊖	√
N6	6 (P90)	1900465	⊖	⊖	⊖	⊖	√	⊖	⊖	√
N7	7 (P95)	1915381	⊖	⊖	⊖	⊖	√	⊖	⊖	√
N8	8 (P99)	10955553	✗	✗	✗	✗	√	⊥	⊥	√
N9	9 (P100)	103790428	⊥	⊥	⊥	⊥	√	⊥	⊥	√

Notebook N5 is an interesting case. Although other notebooks smaller than it fail, it passes for both NVDA and JAWS on Edge and Chrome. Firefox Nightly however was not able to find all the required headers or tables present in the notebook therefore marking its status as a functional but not fully glanceable notebook. We looked more deeply into this and found that one possible cause is that N5 did not contain any programmatically generated graphics.

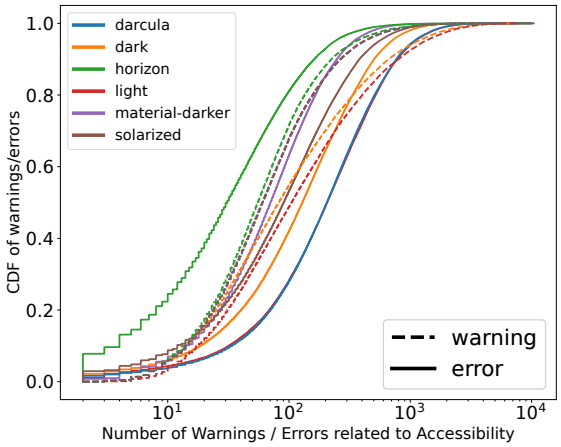
Additionally, our manual accessibility checks demonstrate that for *any notebook size*, both JAWS and NVDA with Chrome, Edge and Firefox only detect the first programmatically generated graphic despite the existence of more than one in the subsequent cells. While navigating by graphic will jump focus to other images that are manually added to markdown cells, both JAWS and NVDA do not jump to any graphics that are encoded as base64 strings that occur in subsequent code cells, a significant accessibility problem (indicated by ⊖ in Table 5.3). We further tested this behavior by adding synthetically generated base64 encoded images of different sizes into HTML outputs randomly chosen from our data, and to new notebooks with just two cells and observed similar behavior. It is important for screen reader users to be informed of the presence of these images and

to be able to navigate to them. Current attempts at navigation using popular NVDA and JAWS screen readers completely skip images. VoiceOver on the Mac performs the best with no crashes and detects all images, headings, and tables -- satisfying the requirements for notebook glanceability we set forth (denoted by $\surd$), and enables screen reader users to navigate to them using the VoiceOver rotor -- a single-key navigation equivalent for VoiceOver. Though the notebooks are glanceable using VoiceOver on the Mac, it is critical that Windows based screen readers be able to do the same due to their wide adoption among BVI users (WebAIM Surveys, 2021).

Impact of Theme Choice on Accessibility



(a) Examples of notebooks rendered in different Themes



(b) CDF of accessibility warnings and errors based on theme

Figure 5.7: Customizability of Notebook experiences and its accessibility implications. **Left (fig. 5.7a)**: shows the visual difference in notebooks when applying different themes in our evaluation.**Right (fig. 5.7b)**: shows the distribution of the number of accessibility errors and warnings reported by accessibility engines on the same set of notebooks exported into multiple themes.

To understand the impact of the choice of theme to accessibility of the notebook, we evaluate each notebooks' accessibility using six themes, two of which are the default (light and dark), and the four others which are popular defaults used by various IDEs. The use of color schemes to modify

the visual style of the editor interface is a common practice and allows developers to improve their productivity due to their ability to make code easier to visually read and understand. These visual differences are summarized for the six themes we selected in Figure 5.7a. Applying color schemes such as high contrast themes on IDEs has been widely adopted by developers for both accessibility (low vision developers), and aesthetic reasons. Although Jupyter notebooks by default are exported to HTML using the *light* theme, notebook authors can specify a different theme to use and export the notebook into. As summarized in fig. 5.7, these themes also differ significantly by the amount of warnings and errors related to accessibility that we found when we used automated testing on them.

We found that the horizon theme, which is the default for the popular VSCode IDE, performs the best, with the fewest accessibility errors (μ=67.52 (σ=138.97) errors). It was 84.95% better than the default light theme provided by the Jupyter IDE, which had a mean of 335.40 (σ=393.72) errors. This difference is statistically significant according to the paired-samples t-test ($t(95101)=198.05$, $p < .001$). Figure 5.7b summarizes all six themes using a CDF showing the distribution of the number of errors (solid lines) and warnings (dashed lines) reported by the aXe and HTMLCS accessibility engines on the exported HTML versions of the notebooks.

In summary, the findings presented here give us an optimistic baseline of the accessibility of notebooks. Specifically, we find that data presented in notebooks is inaccessible, due to issues stemming from lack of tool support and poor authoring practices. We also find that the tools used to export notebooks can have a significant impact on accessibility, glanceability and navigability.

5.4 Towards Real-world Accessibility of Computational Notebooks

Our work is the first large scale analysis of computational notebooks that was conducted to study their accessibility. Our study of 100,000 notebooks provides insights not only about the current state of accessibility of notebooks, but also suggests directions for future research. We note that our analysis does not directly address the fact that web-based notebook editors are not accessible to

BLV authors. We do not say more about this as it was not a focus of our data analysis, but it is an important domain for future work to address (Man From Jupyter, 2020).

5.4.1 Improving Artifact Accessibility

We found that notebook images almost universally lack ALT text, and are usually created with `Matplotlib` or a library built on top of it (`seaborn`). We also found a lack of accessible, tabular information associated with figures. Many of these concerns could be improved upon by providing additional options to developers using the libraries or minor changes to the tool defaults.

Including Alternate Text in Images

During the programmatic creation of tables and images, there is a great deal of available knowledge that could be used to improve their accessibility. First, it is possible to check for the presence of axis labels, validate color contrast, and even generate basic ALT text that would be significantly better than what we found (fig. 5.3a), all based on information available when the chart is instantiated. For example, `Matplotlib` during the generation of a line chart, could include alt-text information indicating the type of chart, the color and number of lines, and the labels along various axes. Proprietary statistics software such as SAS/STAT include such metadata information in the graphics that are generated based on the context such as the function call, or the data being visualized (SAS Help Center, 2023). Second, notebook interactions could be designed to help users encode image descriptions according to the four-level semantic model proposed by Lundgard and Satyanarayan (2021). Third, it is very feasible to modify a tool such as `Matplotlib`, the most used plotting tool as observed in our analysis (presented in table 5.2), and provide programmatic methods to embed ALT text in PNG images (the most common image type it produces in our data (table 5.1)). ALT text could be provided manually by the notebook authors, similar to the ability for web developers to customize the `alt` attribute in HTML `<img>` tags to describe images.

To demonstrate the feasibility of our suggestion, we updated the `Matplotlib` backend for `PNG` formats, by modifying the modules' source code (`image.py`, `backend_ (png|pdf).py`), to use the standardized Exchangeable Image File Format (`EXIF`) and include alt text information which could be passed as image metadata by making less than 10 lines of code changes to the open source code. Such image description information can be encoded in a serialized byte format against the `0x010e (270)` byte delimiter following the EXIF standards. Following the established and existing standards would allow consumers of notebook formats, such as Jupyter, VSCode, and various IDEs to embed ALT text into the figures and make it available to screen reader users. We further verified that the descriptions included using our proposed alt argument can be included by existing tools like `nbconvert` and included in the HTML conversions of the notebook.

Our artifacts released as a part of this work utilize these changes and include ALT metadata description in the resulting images. We leave their ability to be automatically recognized by notebook software for future open source efforts motivated by this research.

Leveraging Interactivity on the Web

Visualizations on web pages (using libraries like d3.js, Highcharts, jQuery charts), are often generated during the page load event establishing a separation between the data and the functions responsible for creating and rendering the visualizations, enabling interactivity, and making it possible for extensions to build additional accessibility features (Sharif et al., 2022). Future work could explore comprehensive API designs similar to those that are provided by Apple to sonify charts (Apple, 2022), to replace static base64 encoded images returned from the Jupyter python kernels' message passing interface with a representation of only the data required offloading the visualization capabilities to the Jupyter front end. This could allow web exports of notebooks to separate the data and functions responsible for visualizations, thus making it feasible for including *accessible*, and interactive graphs.

Presenting Multiple Data Representations

Similar innovations could be added to the tools used in notebooks to improve table prevalence and accessibility. For example, we found that tables don't always accompany charts, despite this being a best practice (Elavsky et al., 2022). It may be possible to extend modules such as Matplotlib, or seaborn to return a table representing the various data points in the image being created, or for notebook tools to perform code analysis of popular data processing libraries like pandas (table 5.2) and automatically insert the intermediate data representations passed to the visualization functions as a table. For example, notebooks with the understanding of a code cell using a `pandas` dataframe in a variable '`df`' calling the `lineplot(data=df, x='label', y='data')` could create a snapshot of the data corresponding to the 'label' and 'data' columns in its metadata and present a table along with the resulting figure. Future accessibility efforts to improve this understanding could develop heuristics to assess the relevance of tables and other data representations in notebooks.

Addressing Large Tables

Our characterization of tables used in computational notebooks shows that these tables are typically very wide (presented in fig. 5.2c). Additionally, Wang et al. (2022) highlight table navigation challenges experienced by blind web users, finding that screen reader users have difficulty keeping track of context when navigating long tables. Programmatic support could be designed to ensure that the defaults for representing tabular data are as accessible as possible, though additional research is needed to understand the best way to accessibly represent tables for data sets with large numbers of columns. Extensive work has been done by a few contributors to the Notebooks4All effort to provide guidance on making table outputs accessible (Fast, 2023a).

Wang et al. (2022) also find that screen reader users encounter incorrectly marked up tables. Though not significant, our accessibility scan results do indicate the presence of table related errors, possibly generated by data processing libraries such as pandas when specific constants to improve

their accessibility are not set (Fast, 2023a). Code analyses extending the one described in our pipeline (section 5.2.2), combined with accessibility scans of programmatically generated tables with exhaustive combinations of parameters that have an impact on table output, could help identify root causes and target accessibility improvements.

5.4.2 Accessible Notebook Authoring

We found that many notebooks have an H1 in their first cell, but consistent and proper use of headers is by no means universal. Further, only a third of notebooks have even one table in them. Improving these authoring practices could improve notebook accessibility.

Computational notebook tools could be updated to provide notices to the authors about incorrect usage of headings when violating the heading order or existence in the WCAG2 guidelines. Similarly, they could suggest places where authors may want to summarize what has happened so far in a table. This could be valuable for all notebook authors, regardless of disability, since tables convey valuable information about the structure of the data being processed in the subsequent cells of the notebook. These practices, along with increased figure accessibility, can improve notebook comprehensibility and glanceability. Future efforts could focus on understanding and improve notebook glanceability to further explore what glanceability means to a BVI user based on the task at hand (authoring *vs* consuming) and incorporate these best practices into tools such as linters for Jupyter Notebooks, to support accessible notebook authoring.

5.4.3 Accessible Notebook Consumption

Notebooks that are converted to HTML may be accessible to consumers if that HTML is accessible. When we used the most accessible theme (Horizon, which is not the default that most authors are likely to use), we found a mean of 68 accessibility errors, when testing notebooks with automated tools. This was significantly better than the worst theme, which had an average of 335 errors. These

results are particularly striking since they imply that changing the default theme of the notebook software during export, or enabling the ability to toggle themes in the exported notebooks, could significantly improve the overall accessibility. While changing the theme is not the remedy to all accessibility issues in notebooks, it is an important factor to consider when improving and addressing the challenges of notebook accessibility.

As a first step, the Jupyter community could explore conducting an accessibility evaluation of their theme(s), address existing accessibility issues, and provide improved accessible defaults. Future explorations could investigate the feasibility of integrating accessibility checkers such as aXe, or HTMLCS into the `nbconvert` process or within the notebook programming environments. Additionally, notebook authors hosting their notebooks on source code repository systems such as GitHub could also consider integrating accessibility engines to test the export of notebooks into their continuous integration and deployment pipelines (CI/CD) which could highlight and prevent accessibility issues during notebook creation (Sutton, 2023).

5.5 Contributions

Prior to this work, the research community had very little knowledge about how the authoring practices, infrastructures, and tools to author and share notebooks impacted their accessibility. We conducted the first at-scale analysis of computational notebooks geared towards investigating their accessibility. Further, we draw upon prior understanding of notebook use, web accessibility literature, and accessibility best practices to share data representations to develop an optimistic baseline understanding of the accessibility of these notebooks.

In connection to my thesis statement, This at-scale analysis contributes a *data-driven study* to investigate the accessibility of notebooks, and the *visual data representations* in them. This work led to an ASSETS 2023 paper Potluri et al. (2023). The code and datasets are publicly available for future efforts to help us gain a deeper, nuanced understanding about the accessibility of notebooks.

Chapter 6

A DISCOURSE DRIVEN DISCUSSION ON AI FOR ACCESSIBLE PROGRAMMING

The work presented so far makes visual code outputs accessible and considers current-day programming practices. The recent adoption of generative AI tools such as OpenAI's ChatGPT, Midjourney, and GitHub Copilot, however, have claimed to revolutionize how we interact with the technology around us, and the creation process of software. Tools like GitHub Copilot, Amazon Code Whisperer, and Tabnine offer productivity benefits (Kalliamvakou, 2022; Yang et al., 2024). While these tools assist developers within their integrated development environment (IDE), a different class of developer interfaces, commonly referred to as *no-code* interfaces, equip developers to train AI models and create AI-powered experiences.

The fast pace of innovation and the potential change caused by these tools to *how we program* calls for a investigation of the impact on accessibility of programming for BVI developers. Additionally, this shift creates a unique opportunity to examine the accessibility gaps experienced by BVI developers as the human-like capability promises of these AI tools may have caused a shift in expectations from computer assistance. Emerging work is beginning to explore the utility of generative AI to increase accessibility (Glazko et al., 2023; Huh et al., 2023). The impact of these tools on the developer experience for BVI developers, remains largely unexplored. (Glazko et al., 2023) describe, through a vignette, the shortcomings of tools such as GitHub copilot to assist BVI developers to create understandable data visualizations. Similarly, (Huh et al., 2023) contribute a system that assists BVI users with creating sharable AI-generated images.

The only exploration of the use of generative AI to improve the BVI developer experience (Saben and Chandrasekar, 2024) attempts to simplify trace messages. They however neither seek input from, nor validate their system with BVI developers.

No-code interfaces to train AI models, agentic systems to perform end-to-end software engineering tasks, and multi-modal capabilities of AI models to understand and reason about text, images, and audio, introduce the potential of capability-agnostic tools and interfaces to program. For example, Versal.ai uses large language models to return UI components on-the-fly, and agent-based systems such as GitHub Copilot Workspaces (Dohmke, 2024), AutoGen (Wu et al., 2023), and devin.ai (Devin (AI), 2024) handle end-to-end software engineering tasks such as fixing GitHub issues and raising pull requests. We argue that such a transformation in how we create software warrants (1) a holistic understanding of the accessibility of programming and software development in the context of AI assistance, and (2) designing born-accessible AI-assisted developer experiences as general-purpose AI tools do not provide assistance that meets the high-quality bar desired by BVI developers (Glazko et al., 2023). We discuss the potential paradigm shift in accessible programming through an exploratory analysis of data from a public archive of the program-l list (a mailing list focused on BVI developers and programming), and prior work in accessible programming.

6.1 Discourse Driven Examination

Prior work examines the accessibility of the developer experience for BVI people through surveys and semi-structured interviews (Mealin and Murphy-Hill, 2012; Potluri et al., 2018; Pandey et al., 2021), observational studies (Albusays et al., 2017), and literature surveys (Mountapmbeme et al., 2022). Our preliminary study adds to this body of literature by examining the state of accessibility of various programming tasks and activities from BVI developers' conversations about programming.

6.1.1 Method

Data Sources

For our analysis, we retrieved email conversations sent to program-l, an email list for BVI developers containing about 62000 posts from 2000-2024. For this exploratory analysis, we reached out to the list moderators and used a dataset located at (Program-l, 2024).

Data Processing Pipeline

Our pre-processing was focused on minimizing the amount of personally identifiable information and repetitive content. To minimize the amount of personally identifiable information, we drop the partially anonymized email addresses provided by the public archive for our analysis. Additionally, we clean up possible name mentions and patterns that might resemble email addresses.

To reduce repetitive content, we drop email signatures and repetitive content that might be included as a part of the email thread. We then filtered for emails contained from 2014 to May 3, 2024, which added up to a total of 34035 emails. For our analysis, we randomly sampled for 5% of emails from each year to ensure that our feasibility assessment (training) dataset had conversations across multiple years. We then similarly sample for 80% from this sample (our generation set) to avoid running into GPT token limits and generate topics.

Topic Generation

Unlike traditional topic modeling approaches, we explored the use of GPT models to generate and assign topics. This approach claims to produce human-interpretable topics with benchmarks comparable to traditional topic modeling approaches (Pham et al., 2023). To ensure data privacy of the email list members, we modified the pipeline provided by (Pham et al., 2023) to work with a private instance of GPT 3.5 and 4 models hosted on Azure OpenAI.

As a first step of generating topics, we curated a total of nine seed topics. These are informed by prior literature (Potluri et al., 2018; Huff et al., 2020) and my experience as a BVI developer, researcher, and a member of the program-l community. They include discoverability, navigability, alertability (Potluri et al., 2018), employment and education (inspired by (Huff et al., 2020)), and artificial intelligence and feature requests (motivated by current trends and personal experiences of the research team members). These topics were selected in consultation with another team member, who is also a BVI researcher, developer, and member of the program-l community. We used our generation set along with these seed topics to generate first level, and second level generation topics. We retained most of the parameters suggested by (Pham et al., 2023). We modified the threshold for duplicate topics to account for the larger corpus, and the models to use our instances on Azure OpenAI. The first level topic generation spanned for approximately 17 hours.

Topic Assignment

We used GPT 3.5 to assign topics as recommended by (Pham et al., 2023). We randomly sampled for 5% of emails from each year between 2014-2024 and assigned either a primary or secondary topic. We also assigned topics for emails that contained mention of research artifacts such as CodeTalk, Quorum, and StructJumper, and emails that had a mention of generative AI tools such as ChatGPT, Midjourney, GitHub Copilot and transformers (we included transformers due to the relevance to generative AI). Our goals with this exploratory analysis is to develop a data (discourse) driven understanding of the accessibility of programming, but is not to evaluate the performance of the topic modelling approaches. We do not check for overlaps in the assignment sets in this step.

Post-Processing and Analysis

We clean up the assignment results and tabulate them for easy qualitative analysis. We randomly analyze a 10 % sample to evaluate topic relevance and to check for hallucinations. We present

results focused on conversations around programming domains (relevant to my thesis topic), and AI (relevant to current-day development).

Methodological Limitations and Considerations

The use of large language models to perform qualitative analysis is an emerging, yet sort-after method (Lam et al., 2024). The descriptive, human-like outputs of these models provide an opportunity to evaluate their reasoning and bias using qualitative methods similar to those that are use on human-generated qualitative data (Glazko et al., 2024). These large language models claim to offer comparable results with crowd workers (Gilardi et al., 2023). At the same time, they do not capture the nuances that are essential to design to include minoritized identities such as gender and disability, and contradict with core values of user centered research of understanding the subjects and representation (Agnew et al., 2024).

These models have sparked a renewed interest in topic modeling -- a method that has been previously explored to understand the experiences of BVI computer scientists (Seo, 2019; Park et al., 2023). The earlier efforts were limited by the interpretability of the topics generated by traditional topic modeling approaches, which our chosen approach claims to offer (Pham et al., 2023).

In summary, the use of large language models to perform topic modeling on a corpus of information about a minoritized identity -- developers who are blind or visually impaired -- has been largely unexplored, and our preliminary exploration leaves several pertinent questions for the future. Based on our observations, we discuss limitations around output consistency, computational time and lack of nuance, and future plans to extend our preliminary analysis.

The use of GPT-4 was time consuming with an approximate response time for each document (an email in our data sample) taking about 60 seconds. This prevented us from being able to conduct multiple iterations of topic generations, on a larger chunk of the corpus. Multiple iterations on topic generation could allow us to validate the consistency in generated topics, and systematically identify

topic saturation. We however consider our results to be useful due to the benchmarked performance of this approach by (Pham et al., 2023). We plan to modify the approach suggested by (Pham et al., 2023) to work with OpenAI's batch processing API (OpenAI, 2024) to reduce costs and increase processing time.

Additionally, at least one email conversation about a user frustration triggered the content moderation policy, and a few requests resulted in errors that were hard to debug due to the opaqueness in these models. For example, a few responses did not result in topics being generated due to rate limits despite throttling our queries, reducing tokens, and introducing retries. Additionally, the topic assignment approach suggested by (Pham et al., 2023) resulted in data loss to perform descriptive statistical analysis (this data loss was approximately 4.5% of assignments). The model generated outputs in highly inconsistent formats that made reliable post-processing infeasible. We will explore approaches to mitigate such inconsistencies in the future.

Finally, though the topics generated were useful to gain a high-level understanding of the conversation, they occasionally lacked nuance that an expert qualitative researcher might provide. to further validate the topics and the classification of the email samples into these topics, we will incorporate an inter-rater reliability check and refine the results. Additionally, we will explore the feasibility of emerging methods *e.g.,* proposed by (Lam et al., 2024).

Understanding Preliminary Results

We treat the topic assignments similar to a light-weight thematic analysis and present results based on those themes and our research objectives. Quotes are email responses, slightly modified for readability, and to protect the anonymity of the original poster.

6.1.2 Preliminary Results

We summarize findings that inform us about the expert programming domains of interest for BVI developers, and expectations and the state of AI for accessible software development.

Expert Programming Domains

We observe that members of the program-l list were experts of, or were interested in Web Development, Mobile Development and developing using artificial intelligence. We also encountered occurrences of interest in game development, although the evidence that pointed to this was not of sufficient quality to understand and report further nuance.

Conversations around web development included accessibility focused help to perform web development related tasks such as configuration of front-ends and hosting. *''I am a web designer. . . have my own business. With my clients, I mostly use go daddy, because it is accessible, really cheap. If I am not mistaken, there I believe is an area to install drupal on that platform.''* Here, the poster is offering another beginner web developer advice on an accessible hosting platform (Johnson et al., 2022). Similarly, developers also sought help to create and configure front-ends. WordPress appears to be the most popular hosting platform, possibly due to its abstractions that separate layout and content (Li et al., 2019; Potluri et al., 2021; Pandey et al., 2022). In addition to guidance about platform choices, list members also appear to share knowledge about the possibility of developing web front-ends. One such post snippet highlights both the possibility, and challenge of developing aesthetically good-looking front-ends:

> *''I have good enough vision that I could create great user interfaces myself, but my UIs always look very simple old - fashioned. I know CSS, least the syntax level, but I'm no designer. Recognizing this, my boss brought our company's graphic designer into my department to help. . . has taken my super - simple UI, and turned it into something*

> *my company could show off. So there definitely is a certain art to it, vision is not the issue. But it does put blind VI programmers at a disadvantage, because unless we already know how every CSS attribute works… odds are it won't look right. In one of my apps work I had buttons where the text ran over the border, other little things like that which took some figuring out. So I'm not about to say no, it's impossible, but it's a lot harder.''*

Similar to web development, conversations about mobile app development were also focused on frameworks to develop front-ends (Xamarin, Flutter, and React were used), and debugging assistance related to developer tools such as Android Studio.

Despite several efforts to improve accessibility of UI developer tools (Li et al., 2019; Potluri et al., 2019b; Li et al., 2022), the lack of commercially available developer tools continues to pose barriers to programming opportunities with front-ends as a central component to the end-product, making the need for accessible UI design tools urgent (§3.5.3). We observe that BVI list members expect AI tools such as ChatGPT and GitHub Copilot to offer assistance, but realize the limitations of these models to generate high-quality outputs. One list member summarizes by saying:

> *''Yes, ChatGPT can do that, but requires human supervision. And sometimes, the code generated is very ugly. but you can ask for even more complex UIs. I used it even for openScad! Open Scad is not a popular language, I think. so getting results for openScad surprised me. I use this to get an initial idea of the think that I want, but then I implement my own solution because most of times gpt does not follow good design patterns.''*

In summary, the ability to design user interfaces remains to be a challenge for aspiring blind and visually impaired (BVI) web developers.

AI Assistance for Software Development

We observe that BVI developers used AI assistance to increase productivity, accessibility, and knowledge. We report findings related to tool use and provide user perspectives on the current strengths and limitations of AI assistance.

BVI developers used both general-purpose AI tools (*e.g.,* ChatGPT), and generative AI tools for code completion (GitHub Copilot was the most popular). Our approach surfaced the prominence of GitHub Copilot despite the presence of other similar tools such as cursor.ai (Cursor, 2023). The accessibility of Visual Studio and VS Code may have been a reason for this adoption. A deeper investigation into the different AI tools used by BVI developers could provide more nuance behind tool choices. Though BVI developers found productivity benefits in using these tools, they often encountered accessibility barriers due to the complexity in code suggestions that the tools offered. A conversation about GitHub Copilot and attempts to improve its screen reader experiences highlights this challenge.

> *‘‘problem we’re running into is how to communicate the suggestion to the user. Effectively, the visual user sees their code, the code that [GitHub Copilot] is suggesting, but the suggested code is lighter grey, making it ‘easy’ to see what is and (isn’t) part of their code. A sighted user also doesn’t need to navigate through the suggestion to investigate it, whereas the screen reader user does. For instance, we’ve got the following scenario: Type ‘C’ in a C # file. IntelliSense comes up, suggests the methods you might want to use. Arrow down until you find Console. . . screen reader says ‘console’ as it should, but we’ve got one system that wants to give you the information about the command (e.g., class System. Console). . . What should we be reading? Is the parameter information more (or less) valuable than the suggestion? Should reading the suggestion be a separate keystroke? ’’*

Traditional completion tools such as IntelliSense offer suggestions that are one word or one code line long. These suggestions are easy to verbalize using a screen reader and BVI developers can quickly choose to accept a suggestion or ignore it. Suggestions with GitHub Copilot, however, can span an entire method. Waiting to hear the entire suggestion to decide to accept or reject it could be time consuming and may require BVI developers to memorize a huge chunk of code. Auditory alerts have been suggested by BVI developers as a way to indicate the presence of a Copilot suggestion, and dialog boxes have been suggested as a way to interactively explore these suggestions.

While BVI developers are working to adopt AI assistance into their developer workflows, they are also skeptical about the accuracy of these tools, and would prefer to have alternative interfaces that did not require them to use ChatGPT's web interface.

> *''I then go back and ask chatgpt to do the same challenge. So far it can do it all but two languages . . . I was being pretty picky about how I wanted it done and it was pretty [bad]. . . . I will say most cases it takes someone who knows these languages to fix errors or just weird [things] chatgpt does or at least point the errors out to the chat bot. . . . It has been a fun experiment so far I have done it with over 20 languages. I currently have 40 of the 67 languages running on my computer from exercism challenges and have been doing all the tracks without the AI's help so it is fun to go back and see how smart the silly thing is.''*

Here, the developer explains using ChatGPT to test its capabilities and limitations as a fun exercise, and highlights the need for user expertise to trust its output. But as observed earlier, developers also may desire to use AI assistance for tasks that are inherently inaccessible such as UI design and data visualizations (Glazko et al., 2023). We were unable to gain a deeper understanding of the trust developers are willing to place in generative AI systems, especially when these systems offer the promise to increase accessibility to a previously inaccessible task.

However, our findings signal the desire to use these tools, and expectation among BVI developers that they would aid in increasing accessibility of software development.

> *''I decided to take the plunge and got a ChatGPT+ subscription. And I am completely mind blown and now think of my abilities as a software engineer a completely different way. I mean it. After finishing the API key creation, I just requested ChatGPT's web browser version to do the following: 1. write me, in swiftUI, a messaging app with rows white font and black background 2. write me a swift ChatGPTAPI class that will send requests to the API 3. write me a JSON encoder decoder struct to convert the data for the return value of the request function... I then created a new xcode project, added the files, did one change to the code which did not take into account a parameter, and converted a variety of objects into Any so that the compiler won't complain... and I got a local full ChatGPT service. The web version gets cluttered as a session runs longer, but on my simple interface I can access each row of the answers much quicker... I am mind blown. This took me less than an hour to do.''*

In summary, we find that BVI developers may want to participate in professional domains that involve web and mobile development, and the development of artificial intelligence. BVI developers have begun to adopt AI assistance into their developer workflow to gain knowledge, gain similar productivity benefits promised to sighted developers, and to increase accessibility to previously inaccessible visual programming tasks. Accessibility of AI suggestions, accuracy of AI outputs, and accessibility of interfaces to interact with these AI tools themselves may be immediate hurdles to truly assess the impact that these class of tools can have on the BVI developer.

6.2 The Future of Nonvisual Programming

Recall that the work described in this thesis answers the following overarching research question: *How can we approach the design of both domain-specific and general-purpose development*

environments to make specialized programming domains accessible to BVI developers? The recent adoption of generative AI tools in the software development process blurs the gap between general purpose programming tools and domain specific ones, and increases user expectation about the capabilities of AI assistance in software development. We discuss the future of new interaction techniques, access to data representations, and data-driven studies to ensure that software development and programming continue to remain accessible to BVI developers.

6.2.1 New Interaction Techniques

The work presented in this dissertation demonstrates the importance and impact of new interaction techniques to increase accessibility of user interface design -- a task that has different information needs from traditional programming tasks, and that has been inaccessible to BVI developers. Further, we also discuss the impact of extending new interaction techniques to screen readers on access to visual information. Relatedly, AI assisted developer tools powered by language models surface the need for new interactions.

A key difference in traditional programming tools *vs* AI assisted ones that provide functionality similar to auto complete is that AI assisted developer tools provide large volumes of information (code, comments or explanations) as suggestions which often need to be audited for accuracy and correctness. Current programming tools such as GitHub Copilot provide this information using affordances optimized for a sighted developer to glance, and decide to accept or reject a suggestion quickly. Though GitHub Copilot provides information about the presence of an AI-assisted suggestion accessibly, the ability to quickly audit it remains inaccessible. The default behavior results in a sound effect signaling the presence of a suggestion. A screen reader user can then insert the suggestion to review it, or review it in a separate window -- requiring additional interactions and time. This experience causes additional challenges to verify the correctness of the code change suggestions spanning multiple files, or multiple non-consecutive lines within a file.

Additionally, a different class of developer assistance tools such as Devin.ai Devin (AI) and GitHub Copilot Workspaces (Dohmke, 2024) generate plans and fixes for entire work items or issues. These code changes can span multiple files, and can be verbose. Though these outputs are not quickly glanceable, the large amounts of information, if not optimized for screen reader consumption, may be overwhelming to a BVI developer (Potluri et al., 2018).

Finally, tools that offer AI assistance with visual tasks, or other inaccessible programming tasks may generate outputs that may be inaccurate or incorrect. BVI developers may not have access to the tools to verify the correctness of outputs due to the tasks being inherently inaccessible. The lack of mechanisms in AI assisted tools for visual content creation, and the verification of its correctness, may further exacerbate accessibility gaps for BVI developers (Glazko et al., 2023; Huh et al., 2023).

Future efforts to explore new interaction techniques could investigate the use of large language models to reduce verbosity of large outputs produced by AI assistance tools to offer quick glanceability. Current verbosity settings for screen readers are limited to very specific attributes of screen reader speech, such as punctuations and whether or not to announce tutor or help messages. The notion of verbosity in screen readers may need to be re-visited to meet the needs of AI-generated outputs, which may be human-like and ambiguous in terms of length and level of detail. Additionally, future studies could explore designs to integrate AI-generated content into screen reader feedback. Early features such as Picture Smart in JAWS for Windows (Jones, 2024) demonstrate early possibilities of AI assistance in screen reading. These efforts however, are limited to images. Efforts like VoiceOver's screen recognition could explore the use of AI assistance to answer questions about screen content that may be inaccessible, automate tasks, and other use cases informed by the BVI developer community.

6.2.2 Access to Data Representations

Multimodal interactive exploration, BVI-customizability and dialog-based interfaces have shown to increase access to data visualizations (Sharif et al., 2022; Potluri et al., 2022b; Zong et al., 2024). The work presented in this dissertation makes one overlooked types of data visualizations -- of streaming sensor data -- accessible through an interactive sonofication toolkit. The experience of accessing data through interactive sonification toolkits such as PSST is driven by a user configuring the desired parameters for a data display. The ecosystem of generating these interfaces to explore and query data remains fragmented, and requires user expertise. AI assistance can have the potential to generate a starter representation depending on the dataset and context, or based on an exploration goal communicated by the user.

Our attempts to drive PSST with a large language model were unsuccessful, possibly due to the models not understanding nuances that are needed to generate a meaningful data display. Future efforts could explore the use of custom GPTs with sonification best practices and examples to drive data sonification toolkits as such instruction can improve accessibility (Glazko et al., 2024). Additionally, additional datasets, possibly created with BVI experts or crowd workers could help fine-tune AI models to generate meaningful sonifications and interactive data displays.

6.2.3 Data Driven Studies

Our analysis of Jupyter Notebooks, and the program-l mailing list shows that we can develop a baseline understanding of the accessibility of widely used tools. Given such a large footprint of information about accessibility, future research efforts could consider ethically looking to such data sources prior to engaging with BVI developers. Developing a baseline understanding could assist with crafting research questions that could inform us of deeper nuances around the accessibility of the BVI developer experience.

Further, given the variety of capabilities promised by AI tools, developing a holistic understanding of successful, and failed attempts of AI assistance in the wild could reflect real-world expectations and use of AI assistance. Future augmentations to opensource screen readers such as NVDA, and a diary study combined with data collections about the interactions with AI models may be a viable approach to develop a baseline for AI assistance.

Chapter 7

CONCLUSION

Recall that the work described in this thesis establishes that *access to visual code outputs is critical for BVI experts to contribute expertise in high-skilled programming work.* New interaction techniques, access to data representations, and data-driven studies are critical to make this visual information in widely used programming tools accessible.

7.1 Contributions and Reflections

My work to make user interface design demonstrates a touch plus keyboard developer experience on commodity hardware (an iPad) and an accessible general-purpose development environment familiar to BVI developers (VSCode). In addition to contributing a new interaction technique, UITap demonstrates the importance of computer aided assistance to identify and fix visual bugs. We discuss recommendations that could result in a real-world adoption of the multi-modal developer experience. Similarly, in demonstrating PSST, a toolkit to make streaming data accessible, we build capabilities suitable to examine sensor data -- from Jacdac sensors -- a very specialized domain and platform while keeping the design of the toolkit relevant to make streaming, real-time data at large accessible to BVI developers. Finally, we analyze Jupyter notebooks -- widely used tools in a variety of domains such as data science, machine learning, and astronomy to gain insights into their accessibility and the tools, infrastructure, and practices that make them inaccessible.

While the work described in this dissertation may have caused a shift in how we address accessibility for BVI developers, the developer experience is expected to undergo a paradigm shift

caused by the adoption of AI assistance. This dissertation offers recommendations, grounded in prior work and BVI developer experiences, to ensure that this redefined developer experience is accessible from the get-go. considering *new interaction techniques* that enable BVI developers to efficiently navigate and audit AI-generated outputs, including the correctness of AI-generated visual code outputs become of paramount importance. Additionally, re-visiting *accessible data representations*, and interactions therein, could ensure that BVI developers are involved in the development of data-driven user experiences. Finally, the need for ethical, privacy-focused data collection and *data-driven investigations* about in-the-wild experiences of using AI assistance may be critical to truly understanding the strengths, and gaps of AI systems to be of use to BVI developers, and people with disabilities at large.

7.2 Closing Remarks

My work described in this dissertation alleviates accessibility barriers posed by the design of technical systems -- developer tools to equip BVI developers to realize their full potential. While the longitudinal and true impact of AI-assistance remains to be assessed, the attempts of BVI developers to use these systems, and work with imperfect tools to perform various tasks hints at their expertise and signals their desire to be experts. It is imperative that we, as responsible technology designers, meet these expectations!

BIBLIOGRAPHY

H. Abukadah, M. Fereidouni, and A. Siddique. Mapping natural language intents to user interfaces through vision-language models. In *2024 IEEE 18th International Conference on Semantic Computing (ICSC)*, pages 237--244, 2024. doi: 10.1109/ICSC59802.2024.00045.

W. Agnew, A. S. Bergman, J. Chien, M. Díaz, S. El-Sayed, J. Pittman, S. Mohamed, and K. R. McKee. The illusion of artificial inclusion. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3642703.

K. Albusays and S. Ludi. Eliciting programming challenges faced by developers with visual impairments: Exploratory study. In *Proceedings of 9th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 82--85, Austin, TX, 2016. ACM. ISBN 9781450341554. doi: 10.1145/2897586.2897616.

K. Albusays, S. Ludi, and M. Huenerfauth. Interviews and observation of blind software developers at work to understand code navigation challenges. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '17, page 91–100, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349260. doi: 10.1145/3132525.3132550.

American Foundation for the Blind. Screen magnification systems, 2021. URL `https://www.afb.org/node/16207/screen-magnification-systems`.

Apple. Voiceover, 2019. URL https://www.apple.com/accessibility/mac/vision/.

Apple. Apple developer documentation: Audio graphs, 2022. URL https://developer.apple.com/documentation/accessibility/audio_graphs.

Association for Computing Machinery. Software system award goes to three for pioneering open source initiatives, 2017. URL https://awards.acm.org/software-system.

J. Atherton and P. Blikstein. Sonification blocks: A block-based programming environment for embodied data sonification. In *IDC 2017 - Proceedings of the 2017 ACM Conference on Interaction Design and Children*, 2017. doi: 10.1145/3078072.3091992.

J. Austin, H. Baker, T. Ball, J. Devine, J. Finney, P. De Halleux, S. Hodges, M. Moskal, and G. Stockdale. The BBC micro:bit: From the uk to the world. *Communications of the ACM*, 63 (3):62--69, 2020.

A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 712--721, May 2013. doi: 10.1109/ICSE.2013.6606617.

C. M. Baker, L. R. Milne, and R. E. Ladner. Structjumper: A tool to help blind programmers navigate and understand the structure of code. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 3043–3052, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450331456. doi: 10.1145/2702123.2702589.

M. S. Baldwin, G. R. Hayes, O. L. Haimson, J. Mankoff, and S. E. Hudson. The tangible desktop: A multimodal approach to nonvisual computing. *ACM Trans. Access. Comput.*, 10(3), 08 2017. ISSN 1936-7228. doi: 10.1145/3075222.

F. Beijers. How to get a developer job when you're blind: Advice from a blind developer who works alongside a sighted team, 2019. URL https://www.freecodecamp.org/news/blind-developer-sighted-team/.

K. Beilharz and S. Ferguson. An interface and framework design for interactive aesthetic sonification. In *Proceedings of the 15th International Conference on Auditory Display (ICAD 2009)*, 2009.

C. L. Bennett. A toolkit for facilitating accessible design with blind people. *SIGACCESS Access. Comput.*, page 16–19, 01 2018. ISSN 1558-2337. doi: 10.1145/3178412.3178415.

C. L. Bennett, K. Shinohara, B. Blaser, A. Davidson, and K. M. Steele. Using a design workshop to explore accessible ideation. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '16, page 303–304, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341240. doi: 10.1145/2982142.2982209.

C. L. Bennett, A. Stangl, A. F. Siu, and J. A. Miele. Making nonvisually: Lessons from the field. In *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '19, page 279–285, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366762. doi: 10.1145/3308561.3355619.

C. L. Bennett, C. Gleason, M. K. Scheuerman, J. P. Bigham, A. Guo, and A. To. "It's complicated": Negotiating accessibility and (mis)representation in image descriptions of race, gender, and disability. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445498.

J. P. Bigham, A. C. Cavender, J. T. Brudvik, J. O. Wobbrock, and R. E. Ladner. Webinsitu: A comparative analysis of blind and sighted browsing behavior. In *Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility*, Assets '07, page 51–58, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595935731. doi: 10.1145/1296843.1296854.

J. P. Bigham, C. M. Prince, and R. E. Ladner. Webanywhere: A screen reader on-the-go. In *Proceedings of the 2008 international cross-disciplinary conference on Web accessibility (W4A)*, pages 73--82, 2008.

J. P. Bigham, T. Lau, and J. Nichols. Trailblazer: Enabling blind users to blaze trails through the web. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*, IUI '09, page 177–186, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605581682. doi: 10.1145/1502650.1502677.

J. P. Bigham, C. Jayant, H. Ji, G. Little, A. Miller, R. C. Miller, R. Miller, A. Tatarowicz, B. White, S. White, and T. Yeh. Vizwiz: Nearly real-time answers to visual questions. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, pages 333--342, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0271-5. doi: 10.1145/1866029.1866080.

T. Booth, S. Stumpf, J. Bird, and S. Jones. Crossed wires: Investigating the problems of end-user developers in a physical computing task. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 3485–3497, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450333627. doi: 10.1145/2858036.2858533.

Y. Borodin, J. P. Bigham, G. Dausch, and I. V. Ramakrishnan. More than meets the eye: A survey of screen-reader browsing strategies. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450300452. doi: 10.1145/1805986.1806005.

S. M. Branham and S. K. Kane. The invisible work of accessibility: How blind employees manage accessibility in mixed-ability workplaces. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*, ASSETS '15, page 163–171, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450334006. doi: 10.1145/2700648.2809864.

V. Braun and V. Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77--101, 2006.

R. N. Brewer. Facilitating discussion and shared meaning: Rethinking co-design sessions with people with vision impairments. In *Proceedings of the 12th EAI International Conference on Pervasive Computing Technologies for Healthcare*, PervasiveHealth '18, page 258–262, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450364508. doi: 10.1145/3240925.3240981.

J. W. Bruce and N. T. Palmer. Sift: Sonification integrable flexible toolkit. In *ICAD 05-Eleventh Meeting of the International Conference on Auditory Display*, 2005.

L. Buechley and B. M. Hill. Lilypad in the wild: How hardware's long tail is supporting new engineering and design communities. In *Proceedings of the 8th ACM Conference on Designing Interactive Systems*, DIS '10, page 199–207, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450301039. doi: 10.1145/1858171.1858206.

Bureau of Labor Statistics, U.S. Department of Labor. Occupational outlook handbook, data scientists. Online, 2024. URL https://www.bls.gov/ooh/math/data-scientists.htm. Visited on April 17, 2024.

J. Cherston and J. A. Paradiso. Rotator: Flexible distribution of data across sensory channels. In *International Conference on Auditory Display (ICAD)*, 2017.

M. Choetkiertikul, A. Hoonlor, C. Ragkhitwetsagul, S. Pongpaichet, T. Sunetnanta, T. Settewong, V. Jiravatvanich, and U. Kaewpichai. Mining the characteristics of jupyter notebooks in data science projects, 2023.

C.-C. Chung and S.-J. Lou. Physical computing strategy to support students' coding literacy: An educational experiment with Arduino boards. *Applied Sciences*, 11(4):1830, 2021.

Cursor. Cursor - the ai-first code editor. Website, 2023. URL https://cursor.sh. Accessed on May 21, 2024.

T. H. Davenport and D. Patil. Data scientist. *Harvard business review*, 90(5):70--76, 2012.

J. U. Davis, T.-Y. Wu, B. Shi, H. Lu, A. Panotopoulou, E. Whiting, and X.-D. Yang. Tangiblecircuits: An interactive 3d printed circuit education tool for people with visual impairments. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367080. doi: 10.1145/3313831.3376513.

A. De Campo. Toward a data sonification design space map. In *Proceedings of the International Conference on Auditory Display*, pages 342--347, 2007.

B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, page 845–854, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349819. doi: 10.1145/3126594.3126651.

Devin (AI). Devin the developer - ai coding assistant. Website, 2024. URL `https://preview.devin.ai`. Accessed on May 22, 2024.

L. Dishman. What it's like to be a blind software engineer at Amazon, 2020. URL `https://www.fastcompany.com/40555815/what-its-like-to-be-a-blind-software-engineer-at-amazon`.

T. Dohmke. GitHub Copilot Workspace: Welcome to the copilot-native developer environment. GitHub Blog, 04 2024. URL `https://github.blog/2024-04-29-github-copilot-workspace/`. Accessed on April 25, 2024.

A. Dotzler. Cache the world opt in preview - Mozilla Accessibility. `https://blog.mozilla.org/accessibility/cache-the-world-opt-in-preview/`, 08 2022. (Accessed on 05/03/2023).

D. Drew, J. L. Newcomb, W. McGrath, F. Maksimovic, D. Mellis, and B. Hartmann. The toastboard: Ubiquitous instrumentation and automated checking of breadboarded circuits. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, page 677–686, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341899. doi: 10.1145/2984511.2984566.

G. Dubus and R. Bresin. Sonification of physical quantities throughout history: A meta-study of previous mapping strategies. In *the 17th International Conference on Auditory Display (ICAD 2011)*. OPAKFI Egyesület, 2011.

M. Ehtesham-Ul-Haque, S. M. Monsur, and S. M. Billah. Grid-coding: An accessible, efficient, and structured coding paradigm for blind and low-vision programmers. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393201. doi: 10.1145/3526113.3545620.

F. Elavsky, C. Bennett, and D. Moritz. How accessible is my visualization? Evaluating visualization accessibility with chartability. *Computer Graphics Forum*, 41(3):57--70, 2022. doi: https://doi.org/10.1111/cgf.14522. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14522.

K. Enge, A. Rind, M. Iber, R. Höldrich, and W. Aigner. It's about time: Adopting theoretical constructs from visualization for sonification. In *Proceedings of the 16th International Audio Mostly Conference*, AM '21, page 64–71, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385695. doi: 10.1145/3478384.3478415.

T. Fast. Rendering DataFrames for screen readers with pandas, 2023a. URL https://tonyfast.github.io/tonyfast/xxiii/2023-01-02-accessible-dataframes-basic-indexes.html.

T. Fast. Notebook authoring accessibility checklist. https://github.com/Iota-School/notebooks-for-all/blob/main/resources/event-hackathon/notebook-authoring-checklist.md, 03 2023b. (Accessed on 05/03/2023).

C. Fleet. How tactile graphics can help end image poverty. MIT Technology Review, 05 2023. URL https://www.technologyreview.com/2023/06/15/1074036/ending-image-poverty/. Accessed on May 5, 2024.

Freedom Scientific. Jaws for windows, 2021. URL https://www.freedomscientific.com/products/software/jaws/.

B. J. Fry. *Computational information design*. PhD thesis, Massachusetts Institute of Technology, 2004.

A. A. Galadima. Arduino as a learning tool. In *2014 11th International Conference on Electronics, Computer and Computation (ICECCO)*, pages 1--4. IEEE, 2014.

B. Garcia, W. Diaz-Merced, J. Casado, and A. Cancio. Evolving from xSonify: A new digital platform for sonorization. In *EPJ Web of Conferences*, volume 200, page 01013. EDP Sciences, 2019.

A. Gendreau Chakarov, Q. Biddy, C. Hennessy Elliott, and M. Recker. The data sensor hub (DaSH): A physical computing system to support middle school inquiry science instruction. *Sensors*, 21(18):6243, 2021. doi: 10.3390/s21186243.

W. Gerrey. The Smith-Kettlewell technical file. A Quarterly Publication of The Smith-Kettlewell Eye Research Institute's Rehabilitation Engineering Research Center., 1980. https://www.ski.org/smith-kettlewell-technical-file.

Ghost. Screen readers should announce where collaborators are editing like in google docs. GitHub Issue, April 2020. URL https://github.com/microsoft/live-share/issues/3456. Accessed on July 23, 2024.

F. Gilardi, M. Alizadeh, and M. Kubli. Chatgpt outperforms crowd workers for text-annotation tasks. *Proceedings of the National Academy of Sciences*, 120(30):e2305016120, 2023.

K. Glazko, Y. Mohammed, B. Kosa, V. Potluri, and J. Mankoff. Identifying and improving disability bias in GAI-based resume screening. *arXiv preprint arXiv:2402.01732*, 2024.

K. S. Glazko, M. Yamagami, A. Desai, K. A. Mack, V. Potluri, X. Xu, and J. Mankoff. An autoethnographic case study of generative artificial intelligence's utility for accessibility. In *Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702204. doi: 10.1145/3597638.3614548.

Google. Talkback, 2019. URL `https://support.google.com/accessibility/android/answer/6283677?hl=en`.

J. L. Gorlewicz, J. L. Tennison, P. M. Uesbeck, M. E. Richard, H. P. Palani, A. Stefik, D. W. Smith, and N. A. Giudice. Design guidelines and recommendations for multimodal, touchscreen-based graphics. *ACM Transactions on Accessible Computing (TACCESS)*, 13(3):1--30, 2020. doi: 10.1145/3403933.

A. Guzharina. We downloaded 10,000,000 Jupyter Notebooks from Github – this is what we learned — The JetBrains Datalore Blog. `https://blog.jetbrains.com/datalore/2020/12/17/we-downloaded-10-000-000-jupyter-notebooks-from-github-this-is-what-we-le` 12 2020. (Accessed on 01/18/2023).

W. Haider and Y. Yesilada. Tables on the web accessible? Unfortunately not! In *Proceedings of the 17th International Web for All Conference*, W4A '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370561. doi: 10.1145/3371300.3383349.

T. Hermann, A. Hunt, and J. G. Neuhoff. *The sonification handbook*. Logos Verlag Berlin, 2011.

T. Hildebrandt and S. Rinderle-Ma. Toward a sonification concept for business process monitoring. In *19th International Conference on Auditory Display (ICAD 2013)*, pages 323--330, Lodz, Poland, July 2013. Lodz University of Technology Press. URL `https://eprints.cs.univie.ac.at/3729/`.

S. Hodges, J. Scott, S. Sentance, C. Miller, N. Villar, S. Schwiderski-Grosche, K. Hammil, and S. Johnston. .NET Gadgeteer: A new platform for k-12 computer science education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, page 391–396, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450318686. doi: 10.1145/2445196.2445315.

S. Hodges, S. Sentance, J. Finney, and T. Ball. Physical computing: A key element of modern computer science education. *Computer*, 53(04):20--30, 04 2020. ISSN 1558-0814. doi: 10.1109/MC.2019.2935058.

L. Holloway, C. Goncu, A. Ilsar, M. Butler, and K. Marriott. Infosonics: Accessible infographics for people who are blind using sonification and voice. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022. doi: 10.1145/3491102.3517465.

E. W. Huff, K. Boateng, M. Moster, P. Rodeghero, and J. Brinkley. Examining the work experience of programmers with visual impairments. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 707--711, Online, 2020. IEEE. doi: 10.1109/ICSME46990.2020.00077.

M. Huh, Y.-H. Peng, and A. Pavel. GenAssist: Making image generation accessible. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701320. doi: 10.1145/3586183.3606735.

K. Hussein, E. Tilevich, I. I. Bukvic, and S. B. Kim. Sonification design guidelines to enhance program comprehension. In *IEEE International Conference on Program Comprehension*, 2009. doi: 10.1109/ICPC.2009.5090035.

A. Issa, J. Sillito, and V. Garousi. Visual testing of graphical user interfaces: An exploratory study towards systematic definitions and approaches. In *2012 14th IEEE International Symposium on Web Systems Evolution (WSE)*, pages 11--15, 2012. doi: 10.1109/WSE.2012.6320526.

K. K. Jain. *Making Makerspaces more accessible for people with visual impairment: Understanding user needs to reimagine solutions.* PhD thesis, Massachusetts Institute of Technology, 2021.

P. Janata and E. Childs. Marketbuzz: Sonification of real-time financial data. In *International Conference on Auditory Display (ICAD)*, 2004.

K. V. Jobin, A. Mondal, and C. V. Jawahar. DocFigure: A dataset for scientific document figure classification. In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, volume 1, pages 74--79, 2019. doi: 10.1109/ICDARW.2019.00018.

J. Johnson, A. Begel, R. Ladner, and D. Ford. Program-L: Online help seeking behaviors by blind and low vision programmers. In *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1--6, 2022. doi: 10.1109/VL/HCC53370.2022.9833106.

R. Jones. Picture smart AI: How we got here. Freedom Scientific Blog, 03 2024. URL https://blog.freedomscientific.com/picture-smart-ai-how-we-got-here/. Accessed on April 25, 2024.

Jupyter Development Team. nbformat, 2023. URL https://github.com/jupyter/nbformat.

E. Kalliamvakou. Research: Quantifying github copilot's impact on developer productivity and happiness. GitHub Blog, 09 2022. URL https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-produ Accessed on May 22, 2024.

T. Kaluarachchi. Human-Centered Machine Learning. Encyclopedia, 2021. URL https://encyclopedia.pub/entry/8717. Accessed on April 20, 2024.

S. K. Kane, J. P. Bigham, and J. O. Wobbrock. Slide rule: Making mobile touch screens accessible to blind people using multi-touch interaction techniques. In *Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '08, pages 73--80, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-976-0. doi: 10.1145/1414471.1414487.

S. K. Kane, M. R. Morris, A. Z. Perkins, D. Wigdor, R. E. Ladner, and J. O. Wobbrock. Access overlays: Improving non-visual access to large touch screens for blind users. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 273--282, New York, NY, USA, 2011a. ACM. ISBN 978-1-4503-0716-1. doi: 10.1145/2047196.2047232.

S. K. Kane, J. O. Wobbrock, and R. E. Ladner. Usable gestures for blind people: Understanding preference and performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 413--422, New York, NY, USA, 2011b. ACM. ISBN 978-1-4503-0228-9. doi: 10.1145/1978942.1979001.

C. Kearney-Volpe and A. Hurst. Accessible web development: Opportunities to improve the education and practice of web development with a screen reader. *ACM Trans. Access. Comput.*, 14(2), 06 2021. ISSN 1936-7228. doi: 10.1145/3458024.

R. Khurana, D. McIsaac, E. Lockerman, and J. Mankoff. Nonvisual interaction techniques at the keyboard surface. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 11:1--11:12, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5620-6. doi: 10.1145/3173574.3173585.

N. Kim, S. Joyner, A. Riegelhuth, and Y. Kim. Accessible visualization: Design space, opportunities, and challenges. In *Computer Graphics Forum*, volume 40, pages 173--188. Wiley Online Library, 2021.

D. E. Knuth. Literate programming. *The computer journal*, 27(2):97--111, 1984.

A. J. Ko. Six learning barriers in end-user programming systems. In *2004 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 199--206. IEEE Computer Society, 2004. doi: 10.1109/VLHCC.2004.47.

J. Kubelka. Artifact driven communication to improve program comprehension. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 457--460, Buenos Aires, Argentina, 2017. IEEE/ACM. doi: 10.1109/ICSE-C.2017.47.

R. Kumar, A. Satyanarayan, C. Torres, M. Lim, S. Ahmad, S. R. Klemmer, and J. O. Talton. Webzeitgeist: Design mining the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 3083–3092, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450318990. doi: 10.1145/2470654.2466420.

M. S. Lam, J. Teoh, J. A. Landay, J. Heer, and M. S. Bernstein. Concept induction: Analyzing unstructured text with high-level concepts using LLooM. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3642830.

J. A. Landay and B. A. Myers. Interactive sketching for the early stages of user interface design. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 43--50, 1995.

E. Lawrence. Fiddler-free web debugging proxy, 2020. URL `https://www.telerik.com/fiddler`.

J. Li, S. Kim, J. A. Miele, M. Agrawala, and S. Follmer. Editing spatial layouts through tactile templates for people with visual impairments. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 206:1--206:11, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-5970-2. doi: 10.1145/3290605.3300436.

J. Li, G. W. Tigwell, and K. Shinohara. Accessibility of high-fidelity prototyping tools. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445520.

J. Li, Z. Yan, E. H. Jarjue, A. Shetty, and H. Peng. TangibleGrid: Tangible web layout design for blind users. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393201. doi: 10.1145/3526113.3545627.

G. Lindgaard, G. Fernandes, C. Dudek, and J. Brown. Attention web designers: You have 50 milliseconds to make a good first impression! *Behaviour & Information Technology*, 25(2): 115--126, 2006. doi: 10.1080/01449290500330448.

S. K. Lodha, C. M. Wilson, and R. E. Sheehan. Listen: Sounding uncertainty visualization. In *Proceedings of Seventh Annual IEEE Visualization'96*, pages 189--195. IEEE, 1996.

S. K. Lodha, J. Beahan, T. Heppe, A. Joseph, and B. Zane-Ulman. Muse: A musical data sonification toolkit. In *International Conference on Auditory Display (ICAD)*, 1997.

C. Low, E. McCamey, C. Gleason, P. Carrington, J. P. Bigham, and A. Pavel. Twitter A11y: A browser extension to describe images. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '19, page 551–553, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366762. doi: 10.1145/3308561.3354629.

S. Ludi. Position paper: Towards making block-based programming accessible for blind users. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, pages 67--69. IEEE, 2015.

A. Lundgard and A. Satyanarayan. Accessible visualization via natural language descriptions: A four-level model of semantic content. *IEEE transactions on visualization and computer graphics*, 28(1):1073--1083, 2021.

A. Lundgard, C. Lee, and A. Satyanarayan. Sociotechnical considerations for accessible visualization design. In *2019 IEEE Visualization Conference (VIS)*, pages 16--20, 2019. doi: 10.1109/VISUAL.2019.8933762.

K. Mack, E. Cutrell, B. Lee, and M. R. Morris. Designing tools for high-quality alt text authoring. In *Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383066. doi: 10.1145/3441852.3471207. URL https://doi.org/10.1145/3441852.3471207.

K. Mack, E. McDonnell, V. Potluri, M. Xu, J. Zabala, J. Bigham, J. Mankoff, and C. Bennett. Anticipate and adjust: Cultivating access in human-centered methods. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391573. doi: 10.1145/3491102.3501882.

F. Macklon, M. R. Taesiri, M. Viggiato, S. Antoszko, N. Romanova, D. Paas, and C.-P. Bezemer. Automatically detecting visual bugs in HTML5 canvas games. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394758. doi: 10.1145/3551349.3556913.

T. M. Madhyastha. A portable system for data sonification. Master's thesis, University of Illinois at Urbana-Champaign, 1992.

Man From Jupyter. Accessibility issues needing addressing for WCAG 2.1 compliance (as of version 2.2.6) · issue #9399 · jupyterlab/jupyterlab. https://github.com/jupyterlab/jupyterlab/issues/9399, 11 2020. (Accessed on 01/18/2023).

J. Mankoff, H. Fait, and T. Tran. Is your web page accessible? A comparative study of methods for assessing web page accessibility for the blind. In G. C. van der Veer and C. Gale, editors, *Proceedings of the 2005 Conference on Human Factors in Computing Systems, CHI 2005, Portland, Oregon, USA, April 2-7, 2005*, pages 41--50, New York, NY, USA, 2005. ACM. doi: 10.1145/1054972.1054979.

R. Manning, H. Kay, J. Robinson, G. Phillips, A. Mee, P. Harlock, and A. Kilgour. Pa11y, 2023. URL `https://pa11y.org/`.

W. McGrath, D. Drew, J. Warner, M. Kazemitabaar, M. Karchemsky, D. Mellis, and B. Hartmann. Bifröst: Visualizing and checking behavior of embedded systems across hardware and software. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, page 299–310, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349819. doi: 10.1145/3126594.3126658.

S. Mealin and E. Murphy-Hill. An exploratory study of blind software developers. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 71--74, Innsbruck, Austria, 2012. IEEE. doi: 10.1109/VLHCC.2012.6344485.

W. D. Merced. How a blind astronomer found a way to hear the stars. TED Talk, 2021. URL `https://www.ted.com/talks/wanda_diaz_merced_how_a_blind_astronomer_found_a_way_to_hear_the_stars?language=en`. Accessed on April 28, 2024.

W. L. D. Merced. *Sound for the exploration of space physics data*. PhD thesis, University of Glasgow, 2013.

Microsoft. Enable accessibility features in Visual Studio Code - Live Share. Microsoft Learn, April 2022. URL https://learn.microsoft.com/en-us/visualstudio/liveshare/use/enable-accessibility-features-visual-studio-code. Accessed on April 25, 2024.

Microsoft. Jacdac, 2022. URL https://aka.ms/jacdac.

L. R. Milne, C. M. Baker, and R. E. Ladner. Blocks4All demonstration: a blocks-based programming environment for blind children. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '17, page 313–314, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349260. doi: 10.1145/3132525.3134774.

O. Mirosa. Jupyterlab material darker theme. https://github.com/oriolmirosa/jupyterlab_materialdarker, 12 2022. (Accessed on 07/23/2023).

Mohirio. Jupyterlab horizon theme. https://github.com/mohirio/jupyterlab-horizon-theme, 12 2022. (Accessed on 05/03/2023).

M. R. Morris, J. Johnson, C. L. Bennett, and E. Cutrell. Rich representations of visual content for screen reader users. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 59:1--59:11, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5620-6. doi: 10.1145/3173574.3173633.

C. Morrison, N. Villar, A. Hadwen-Bennett, T. Regan, D. Cletheroe, A. Thieme, and S. Sentance. Physical programming for blind and low vision children at scale. *Human--Computer Interaction*, 36(5-6):535--569, 2021.

A. Mountapmbeme, O. Okafor, and S. Ludi. Addressing accessibility barriers in programming for people with visual impairments: A literature review. *ACM Trans. Access. Comput.*, 15(1), mar 2022. ISSN 1936-7228. doi: 10.1145/3507469.

N. Nagappan, L. Williams, M. Ferzli, E. Wiebe, K. Yang, C. Miller, and S. Balik. Improving the CS1 experience with pair programming. *SIGCSE Bull.*, 35(1):359–362, 2003. ISSN 0097-8418. doi: 10.1145/792548.612006.

M. Nees and B. Walker. Auditory interfaces and sonification. *The Universal Access Handbook*, pages 507--522, 06 2009.

M. A. Nees. Eight components of a design theory of sonification. In *International Conference on Auditory Display (ICAD)*, 2019.

NV Access. Nonvisual desktop access, 2019a. URL `https://www.nvaccess.org/`.

NV Access. Nvda user guide, 2019b. URL `https://www.nvaccess.org/files/nvda/documentation/userGuide.html`.

NVAccess. NVDA 2023.1 user guide. `https://www.nvaccess.org/files/nvda/documentation/userGuide.html`, 2023. (Accessed on 05/04/2023).

Y. Ochiai. Visible breadboard: System for dynamic, programmable, and tangible circuit prototyping with visible electricity. In *International Conference on Virtual, Augmented and Mixed Reality*, pages 73--84. Springer, 2014.

M. Omer. Audio themes, 2019. URL `https://addons.nvda-project.org/addons/AudioThemes.uk.html`.

OpenAI. Batch processing guide. OpenAI Platform Documentation, 2024. URL https://platform.openai.com/docs/guides/batch. Accessed on May 20, 2024.

M. Pandey, V. Kameswaran, H. V. Rao, S. O'Modhrain, and S. Oney. Understanding accessibility and collaboration in programming for people with visual impairments. *Proc. ACM Hum.-Comput. Interact.*, 5, apr 2021. doi: 10.1145/3449203.

M. Pandey, S. Bondre, S. O'Modhrain, and S. Oney. Accessibility of UI frameworks and libraries for programmers with visual impairments. In *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1--10, 2022. doi: 10.1109/VL/HCC53370.2022.9833098.

M. Pandey, S. Oney, and A. Begel. Towards inclusive source code readability based on the preferences of programmers with visual impairments. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3642512.

S. Pareddy, A. Guo, and J. P. Bigham. X-ray: Screenshot accessibility via embedded metadata. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '19, page 389–395, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366762. doi: 10.1145/3308561.3353808.

J. Park, J. Seo, and J. Y. Lee. Exploring an online community of blind programmers by using topic modeling and network analysis. *Proceedings of the Association for Information Science and Technology*, 60(1):1096--1098, 2023. doi: https://doi.org/10.1002/pra2.956. URL https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/pra2.956.

C. Pham, A. Hoyle, S. Sun, and M. Iyyer. TopicGPT: A prompt-based topic modeling framework, 2023.

S. Phillips and A. Cabrera. Sonification workstation. In *International Conference on Auditory Display (ICAD)*, 2019.

V. Potluri, P. Vaithilingam, S. Iyengar, Y. Vidya, M. Swaminathan, and G. Srinivasa. Codetalk: Improving programming environment accessibility for visually impaired developers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–11, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356206. doi: 10.1145/3173574.3174192.

V. Potluri, T. Grindeland, J. E. Froehlich, and J. Mankoff. AI-assisted UI design for blind and low-vision creators. In *ASSETS'19 Workshop: AI Fairness for People with Disabilities*, 2019a.

V. Potluri, L. He, C. Chen, J. E. Froehlich, and J. Mankoff. A multi-modal approach for blind and visually impaired developers to edit webpage designs. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '19, page 612–614, New York, NY, USA, 2019b. Association for Computing Machinery. ISBN 9781450366762. doi: 10.1145/3308561.3354626.

V. Potluri, T. Grindeland, J. E. Froehlich, and J. Mankoff. Examining visual semantic understanding in blind and low-vision technology users. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445040.

V. Potluri, M. Pandey, A. Begel, M. Barnett, and S. Reitherman. CodeWalk: Facilitating shared awareness in mixed-ability collaborative software development. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '22, New York, NY, USA, 2022a. Association for Computing Machinery. ISBN 9781450392587. doi: 10.1145/3517428.3544812.

V. Potluri, J. Thompson, J. Devine, B. Lee, N. Morsi, P. De Halleux, S. Hodges, and J. Mankoff. PSST: Enabling blind or visually impaired developers to author sonifications of streaming sensor data. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022b. Association for Computing Machinery. ISBN 9781450393201. doi: 10.1145/3526113.3545700.

V. Potluri, S. Singanamalla, F. Tieanklin, and J. Mankoff. Notably inaccessible – data driven understanding of data science notebook (in)accessibility. In *Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 979-8-4007-0220-4/23/10. doi: 10.1145/3597638.3608417.

Program-l. Public archive of the program-l mailing list. Website, 2024. URL `https://www.freelists.org/archive/program-l/`. Accessed on May 22, 2024.

F. Pérez, B. Granger, and M. Ragan-Kelley. Jupyter, 2014. URL `https://jupyter.org/`.

L. Quaranta, F. Calefato, and F. Lanubile. Kgtorrent: A dataset of python jupyter notebooks from kaggle. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 550--554. IEEE, 2021.

L. Race, C. Fleet, J. A. Miele, T. Igoe, and A. Hurst. Designing tactile schematics: Improving electronic circuit accessibility. In *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '19, page 581–583, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366762. doi: 10.1145/3308561.3354610.

L. Race, C. Kearney-Volpe, C. Fleet, J. A. Miele, T. Igoe, and A. Hurst. Designing educational materials for a blind Arduino workshop. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI EA '20, page 1–7, New York, NY, USA, 2020a. Association for Computing Machinery. ISBN 9781450368193. doi: 10.1145/3334480.3383055.

L. Race, J. A. Miele, C. Fleet, T. Igoe, and A. Hurst. Putting tools in hands: Designing curriculum for a nonvisual soldering workshop. In *Proceedings of the 22nd International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '20, New York, NY, USA, 2020b. Association for Computing Machinery. ISBN 9781450371032. doi: 10.1145/3373625.3418011.

T. V. Raman. Emacspeak—direct speech access. In *Proceedings of the Second Annual ACM Conference on Assistive Technologies*, Assets '96, page 32–36, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917766. doi: 10.1145/228347.228354.

Z. Rocha. Solarized. `https://ethanschoonover.com/solarized/`, 10 2013. (Accessed on 05/03/2023).

G. Rodríguez-Pérez, R. Nadri, and M. Nagappan. Perceived diversity in software engineering: a systematic literature review. *Empirical Software Engineering*, 26(5):1--38, 2021.

A. Roginska, E. Childs, and M. K. Johnson. Monitoring real-time data: A sonification approach. In *Proceedings of the 12th International Conference on Auditory Display*, pages 176--181, 2006.

A. Rule, A. Tabard, and J. D. Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1--12, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356206. doi: 10.1145/3173574.3173606.

C. Saben and P. Chandrasekar. Enabling blv developers with llm-driven code debugging. *arXiv preprint arXiv:2401.16654*, 2024.

E.-L. Sallnäs. Navigation and control in haptic applications shared by blind and sighted users. In D. McGookin, editor, *Haptic and Audio Interaction Design*, pages 68--80, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-37596-8.

H. Sampath, A. Merrick, and A. MacVean. Accessibility of command line interfaces. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1--10, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445544.

SAS Help Center. Altdesc, noaltdesc graphics options. `https://documentation.sas.com/doc/en/pgmsascdc/v_034/graphref/n1gj4oarw072dxn1bl3pcqbdv644.htm`, 02 2023. (Accessed on 05/03/2023).

E. Schanzer, S. Bahram, and S. Krishnamurthi. Accessible ast-based programming for visually-impaired programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 773–779, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450358903. doi: 10.1145/3287324.3287499.

E. Schoonover. Solarized. `https://ethanschoonover.com/solarized/`, 2023. (Accessed on 05/03/2023).

J. Seo. Discovering informal learning cultures of blind individuals pursuing stem disciplines: A quantitative ethnography using Listserv archives. *First International Conference on Quantitative Ethnography -- Advances in Quantitative Ethnography*, 2019.

S. Shaikh. How a blind developer uses Visual Studio, 2018. URL `https://www.youtube.com/watch?v=94swlF55tVc`.

A. Sharif, S. S. Chintalapati, J. O. Wobbrock, and K. Reinecke. Understanding screen-reader users' experiences with online data visualizations. In *Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383066. doi: 10.1145/3441852.3471202.

A. Sharif, O. H. Wang, A. T. Muongchan, K. Reinecke, and J. O. Wobbrock. Voxlens: Making online data visualizations accessible with an interactive javascript plug-in. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391573. doi: 10.1145/3491102.3517431.

A. Siu, G. S-H Kim, S. O'Modhrain, and S. Follmer. Supporting accessible data visualization through audio data narratives. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391573. doi: 10.1145/3491102.3517678.

A. F. Siu, S. Kim, J. A. Miele, and S. Follmer. ShapeCAD: An accessible 3d modelling workflow for the blind and visually-impaired via 2.5D shape displays. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '19, page 342–354, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366762. doi: 10.1145/3308561.3353782.

D. Socha and K. Sutanto. The ''pair'' as a problematic unit of analysis for pair programming. In *Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '15, page 64–70, Florence, Italy, 2015. IEEE Press.

Y. Song, P. Wang, X. Hong, and I. McLoughlin. Fisher vector based CNN architecture for image classification. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 565--569. IEEE, 2017.

Space Telescope Science Institute. Astronomy notebooks for all, 2023. URL `https://github.com/Iota-School/notebooks-for-all`.

A. Srinivasan, T. Harshbarger, D. Hilliker, and J. Mankoff. Azimuth: Designing accessible dashboards for screen reader users. In *Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702204. doi: 10.1145/3597638.3608405.

Stack Overflow. Stack overflow developer survey 2021, 2021. URL `https://insights.stackoverflow.com/survey/2021#section-demographics-disability-status`.

Stack Overflow. Stack overflow developer survey 2022, 2022. URL `https://survey.stackoverflow.co/2022#section-demographics-disability-status`.

A. Stefik, R. T. Alexander, R. Patterson, and J. Brown. WAD: A feasibility study using the wicked audio debugger. *15th IEEE International Conference on Program Comprehension (ICPC '07)*, pages 69--80, 2007. URL https://api.semanticscholar.org/CorpusID:28098471.

K. M. Storer, H. Sampath, and M. A. Merrick. ''It's just everything outside of the ide that's the problem'': Information seeking by software developers with visual impairments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 05 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445090.

E. Strasnick, M. Agrawala, and S. Follmer. Scanalog: Interactive design and debugging of analog circuits with programmable hardware. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, page 321–330, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349819. doi: 10.1145/3126594.3126618.

M. Sutton. Automating peace of mind withaccessibility testing & continuous integration, 2023. URL https://marcysutton.github.io/a11y-and-ci/#/.

B. J. Tang, A. Boggust, and A. Satyanarayan. VisText: A benchmark for semantically rich chart captioning. *arXiv preprint arXiv:2307.05356*, 2023.

J. R. Thompson, J. J. Martinez, A. Sarikaya, E. Cutrell, and B. Lee. Chart Reader: Accessible visualization experiences designed with screen reader users. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394215. doi: 10.1145/3544548.3581186.

J. W. Trayford and C. M. Harrison. Introducing STRAUSS: A flexible sonification python package. *arXiv preprint arXiv:2311.16847*, 2023.

M. Vigo and S. Harper. Coping tactics employed by visually disabled users on the web. *International Journal of Human-Computer Studies*, 71(11):1013 -- 1025, 2013. ISSN 1071-5819. doi: https://doi.org/10.1016/j.ijhcs.2013.08.002. URL http://www.sciencedirect.com/science/article/pii/S1071581913001006.

B. N. Walker and M. A. Nees. An agenda for research and development of multimodal graphs. In *International Conference on Auditory Display (ICAD)*, 2005.

B. Wang, G. Li, X. Zhou, Z. Chen, T. Grossman, and Y. Li. Screen2Words: Automatic mobile UI summarization with multimodal learning. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, UIST '21, page 498–510, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386357. doi: 10.1145/3472749.3474765.

J. Wang, L. Li, and A. Zeller. Better code, better sharing: On the need of analyzing Jupyter Notebooks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, pages 53--56, 2020.

Y. Wang, R. Wang, C. Jung, and Y.-S. Kim. What makes web data tables accessible? insights and a tool for rendering accessible tables for people with visual impairments. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391573. doi: 10.1145/3491102.3517469.

Web Accessibility Initiative. Understanding success criterion 1.3.1: Info and relationships. url, 1999. Retrieved Jan 16, 2020 from https://www.w3.org/WAI/WCAG21/Understanding/info-and-relationships.html.

Web Accessibility Initiative. Tables consepts, 2019. Retrieved January 16, 2020 from https://www.w3.org/WAI/tutorials/tables/.

WebAIM Surveys. Screen reader user survey #9 results. https://webaim.org/projects/screenreadersurvey9/, note = (Accessed on 05/03/2023), 06 2021.

K. Williams, T. Clarke, S. Gardiner, J. Zimmerman, and A. Tomasic. Find and seek: Assessing the impact of table navigation on information look-up with a screen reader. *ACM Trans. Access. Comput.*, 12(3), 08 2019. ISSN 1936-7228. doi: 10.1145/3342282.

Q. Wu, G. Bansal, J. Zhang, Y. Wu, S. Zhang, E. Zhu, B. Li, L. Jiang, X. Zhang, and C. Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.

T.-Y. Wu, H.-P. Shen, Y.-C. Wu, Y.-A. Chen, P.-S. Ku, M.-W. Hsu, J.-Y. Liu, Y.-C. Lin, and M. Y. Chen. CurrentViz: Sensing and visualizing electric current flows of breadboarded circuits. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, page 343–349, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349819. doi: 10.1145/3126594.3126646.

Z. Yang, Z. Sun, T. Z. Yue, P. Devanbu, and D. Lo. Robustness, security, privacy, explainability, efficiency, and usability of large language models for code. *arXiv preprint arXiv:2403.07506*, 2024.

K. You, H. Zhang, E. Schoop, F. Weers, A. Swearngin, J. Nichols, Y. Yang, and Z. Gan. Ferret-UI: Grounded mobile UI understanding with multimodal LLMs. *arXiv preprint arXiv:2404.05719*, 2024.

A. Zait. 4 simple steps for debugging your Arduino project. Circuito.io Blog, April 2018. URL https://www.circuito.io/blog/arduino-debugging/. Accessed on April 25, 2024.

M. Zehe. Rethinking web accessibility on windows. https://www.marcozehe.de/rethinking-web-accessibility-on-windows/, 09 2017. (Accessed on 05/03/2023).

G. Zhang, M. A. Merrill, Y. Liu, J. Heer, and T. Althoff. Coral: Code representation learning with weakly-supervised transformers for analyzing data analysis. *EPJ Data Science*, 11(1):14, 2022.

J. Zong, C. Lee, A. Lundgard, J. Jang, D. Hajas, and A. Satyanarayan. Rich screen reader experiences for accessible data visualization. *arXiv preprint arXiv:2205.04917*, 2022.

J. Zong, I. P. Pineros, D. Hajas, A. Satyanarayan, et al. Umwelt: Accessible structured editing of multimodal data representations. *arXiv preprint arXiv:2403.00106*, 2024.

VITA

Venkatesh is an Apple Scholar, a Google Lime Scholar, and he holds a PhD in Computer Science from the University of Washington. He investigates accessibility barriers experienced by BVI developers and contributes new interaction techniques and real-world systems to make programming accessible.