ASSETS: U: Constructing Agency and Usability Through Community-Driven Assistive Technology Design

Thomas B. McHugh
mchugh@u.northwestern.edu
Northwestern University
Evanston, Illinois

ABSTRACT

Despite assistive technologies existing to help people with disabilities interact with software, these tools are often hard to use and not personalized to what a user individually needs. In response to these challenges, new work in the maker community has sought to empower people with disabilities to build their own assitive technologies and to create agency over how one can interact with technology. We discuss current challenges to building assistive technology and overcome these obstacles by presenting V11, a simple cross-platform programming interface to enable the creation of community-driven assistive technology. To evaluate the effectiveness of V11 in environments with novice programmers, we conducted design workshops with undergraduate CS students, who brainstormed solutions to a real accessibility problem and then used V11 to prototype their idea. We found that V11 can help students with limited programming experience build complex assistive technologies and we discuss next steps to engage communities in the creation of these personalized assistive tools.

CCS CONCEPTS

• **Human-centered computing** → **Accessibility**; • **Social and professional topics** → **Computing education**.

KEYWORDS

accessibility; assistive technology; making; computer science education; allyship; inclusive design

1 PROBLEM & MOTIVATION

Through the transformation of computing into a tool for social connection and creativity, the opportunity to democratize creative expression and civic engagement [25] is inherently an exhilarating mission that we see has grasped much of the rhetoric emerging from the tech industry [34]. And yet, whether it be the CS4All movement [51] or digital media's participatory culture [9], the moniker that these new systems have been built "for all" seems to leave out the differences and restrictions that many people with disabilities face when trying to access and participate in digital ecosystems [15]. While the tools and design methods to make one's applications and websites accessible certainly exist, the responsibility to uphold social values through computing [15], such as accessibility and inclusion, is often left to others, if not entirely forgotten. This is apparent in both a quantitative sense, where 98.1% of the top 1 million website homepages have at least one accessibility guideline failure [53], and in more nuanced accessible design challenges, such as creating inclusive collaborative awareness in real-time editing [13] and building accessible interfaces for professional audio editors [44]. One quickly realizes that there is no universal accessibility and that customized solutions are required to truly build for everyone [15].

Given this responsibility to create access through computing, it is critical to include accessibility curriculum into computer science education. This has spurred work to integrate inclusive design skills such as design for user empowerment [29], empathy building [39], and the creation of accessible web [43] and native applications [10] into coursework. However, despite these new ways of introducing accessibility, students are often not taught effectively and even when they are, do not later apply accessibility skills they have learned in practice [55]. Additional barriers to promoting accessibility through education, such as the difficulty for students without a disability to understand how people with disabilities interact with technology [17], means that the tools and curricula used to teach accessibility need to support both students and community members who engage in the design process of new accessible technologies.

We use this background to motivate the design of *V11*, a programming interface for building assistive technology in the JavaScript language. Due to accessibility needing to be individualized and the difficulties for even many experienced programmers to build accessible software, we designed V11 to empower community-driven assistive technology design that could help anyone with a little programming background to create agency over the way they can interact with a computer. We accomplish this by abstracting platform-dependent accessibility interfaces into a DOM-like structure for querying and modifying the native accessibility tree. Additionally, V11 exposes procedures to present data to users through both audio and visual modalities. Due to its similarity with web programming concepts and its abstraction of advanced programming systems, such as audio processors and interface trees, both web and systems programming classes can integrate V11 into existing curricula.

V11 was evaluated by novice CS students who participated in a design workshop to brainstorm and prototype an assistive technology solution to a real accessibility challenge. Participants highlighted the effectiveness of V11 as an easy-to-learn platform for exploring accessibility through programming and found it fun and simple to go from ideation to implementation of an assistive technology. These results affirm that accessibility can be created even with limited programming knowledge. We then discuss our next steps to grow V11 and broaden access to customizable and personalized assistive technology.

2 BACKGROUND & RELATED WORK

This work builds on insights into the design of APIs and programming languages that has been applied to assistive technology design, do-it-yourself (DIY) accessibility, and inclusive making.

2.1 API & Programming Language Design

There is a rich history of applying design methods to the creation of APIs and programming languages with a goal to increase these tools' usability for education, creativity, and social impact. Pane et al. introduce programming languages as user interfaces between a computer and a programmer, and that standard design methods can be leveraged to create more usable languages [37]. More formal methods for designing programming languages have been introduced [35], but these continue to be based on human-centered approaches to designing a range of digital interfaces. These methods have been applied to projects that have taken many different views to what a programming language is [27]. Expressive languages like Scratch [40] are designed to engage young child in the creation of digital media, while languages like Gidget [30] empower computer science students who might be intimidated by debugging and error messages.

As frameworks and toolkits, also known as application programming interfaces (APIs), become common layers on top of programming languages for functionality and reusability [36], the methods for designing programming languages have been adapted to the creation of APIs. API designers must take into account a range of factors when creating these interfaces, including performance, power, and usability [50]. Myers and Stylos survey the effective techniques for designing APIs for usability [36] and find many shared principles from programming language design. Using these methods, our work builds on this field to consider how current accessibility API design patterns work against enabling a low-floor to building assistive technologies, and how more usable and familiar APIs can be created to assist in developing community-driven assistive technologies.

2.2 Assistive Technology Design

While there are many types of physical and digital devices that help people with disabilities interact with computers and the world, our work begins by discovering opportunities for assistive technology that provide accessible interfaces to digital devices. Common tools that currently exist include screen readers, magnifiers, braille displays, conversation interfaces, and single-button controllers. These tools are often built and depend on a specific operating system. For instance, when users rely on a screen reader to control their computer through auditory descriptions of a user interface, macOS users will use VoiceOver [3], while Windows users may use Narrator [31], NVDA [1], or JAWS [47].

In addition to these built in tools, software developers are provided with interfaces to build their own digital assistive technologies. Most operating systems expose low-level APIs [4, 14, 16, 18, 20, 32, 33] to retrieve accessible representations of computer programs, termed the accessibility tree, and these interfaces allows the tree to be presented through assistive technologies. While these interfaces can enable advanced technologies, like screenreaders, they often require a depth of knowledge in systems programming and platform-related frameworks to use. For instance, the Apple AX APIs require the knowledge of both the C programming language [42] and the Core Foundation framework [5]. These APIs are also almost always dependent on the platform, i.e. the Android, Windows, Mac, and Linux APIs are all different. In response to some of the difficulties to observing and manipulating the accessibility tree, higher-level scripting languages like AppleScript [12] and JavaScript for Automation [2] have been created, which provide access to the accessibility tree's data without the requirement to master low-level systems frameworks. Despite this, these abstracted APIs face other restrictions, such as continuing to be platform-dependent and providing a limited syntax that does not take advantage of the expressive nature of these scripting languages.

V11 takes a "write once run anywhere" approach to accessibility and we synthesize these APIs into a design that allows for new comers to understand accessibility trees and to create advanced applications that would be difficult to write with existing accessibility APIs. Rather than creating a long list of prerequisite knowledge to then begin understanding how accessibility on digital devices works, we take the stance that accessibility is in itself a nuanced topic that should be introduced with as little overhead as possible. While some accessibility APIs often seem forgotten or bare-bones, V11 is purposefully built to be opinionated and understandable for our project's stakeholders.

2.3 DIY Accessibility & Inclusive Making

While assistive technologies to interact with computers do exist, the limited usefulness and satisfaction of these tools has spurred work to empower people with disabilities to be actively involved as inventors and collaborators of assitive technologies. Philips and Zhao note that 29.3% of assistive technologies are abandoned due to poor performance. They advise that including users' needs into the design process is required to create effective technologies [38]. Bigham highlights that new technologies, particularly artificial intelligence, are often placed onto people with disabilities, making them become early adopters [6]. To prevent these technologies from limiting the agency of people with disabilities, he argues that we must build on the lived experiences that these users can provide. Ladner also introduces the design for user empowerment methodology [29], which works to increase self-determination for people with disabilities by including them in all components of the design process.

In parallel, the "maker" community has created tools and environments that foster personal expression and powerful ideas through one's lived experiences. Blickstein introduces expressive technology as an agent of emancipation and humanization that can harness a student's creativity and agency in their own communities [7]. Later, he also highlights the history of the maker movement, and argues that students can use computational tools to construct culturally meaningful artifacts that solve personal problems [8]. Kuznetsov and Paulos describe the low barrier DIY communities provide to fully expressing individual creativity through solving problems in these environments [28]. Vossoughi et al. push back against stereotypes of the maker community, and brings to light the importance of integrating students' cultural histories into making activities [52].

Research at this intersection of making and accessibility has enabled opportunities for people with disabilities to create more personalized and usable assistive technologies, while also developing agency and control over the technologies that they use. Hurst and Kane present new accessible making tools as an opportunity

to empower individuals to create their own personalized assistive technologies [23]. Hurst and Tobias build on these findings through design recommendations [24] for DIY assistive technologies that showcase many people's excitement at designing new assistive technologies and that custom built assistive technology can be less expensive and work better than traditional technologies. Hamidi et al. also introduces Sensebox [22], a DIY platform for creating audio interfaces for therapy, and argue that providing maker tools that are simple, customizable, affordable, and accessible, allows audio therapists to create more personalized experiences with their clients. Worsley et al. reflects on the creation of MultiCAD and Tangicraft, and contend that the development of multimodal AI technologies can democratize the opportunities for people with disabilities to build their own assistive technologies [54].

V11 builds on the ideas and technologies from the DIY accessibility and maker communities through providing simple and usable APIs for building assistive technologies. Similar to tools like Arduino and Makey-Makey [11], the creative experiences that making can initiate are enabled by easy to learn technologies with a low-floor of entry, a high-ceiling of opportunity, and a wide variety of applications [41]. While it is important to note that all areas of making should be accessible to people with disabilities, and our work is not trying to define assistive technology creation as the singular opportunity for people with disabilities in the maker community, the life experiences and limited usability of current assistive technologies create an opportunity for maker culture to spur a self-determined future for assitive technologies where community-driven development can create highly meaningful and individualized accessible digital experiences.

3 UNIQUENESS OF THE APPROACH

Our work is composed of three parts: a formative work study to explore the opportunities in designing tools for accessibility, the prototyping of an accessibility API, and an evaluation of our API design through a workshop with students.

3.1 Formative Work

To better understand the educational landscape of accessibility in computer science, we surveyed 16 undergraduate CS students about their experiences and tool usage when designing in classes for accessibility. We recruited these students through university mailing lists. Participants included four freshmen, three sophomores, five juniors, and four seniors.

We separated the survey into three sections: computer science experience, accessibility experience, and knowledge of content creation applications. Computer science experience questions contextualized our study participants by asking them about how many CS courses they had taken, what languages they were familiar with, and what their experience with human-computer interaction and design was. Accessibility experience questions explored whether, and where, any of our participants had thought about, or engaged in the usage, of accessibility tools and design practices and what specific tools they had previously used to create accessible designs. Finally, we asked questions about the participant's knowledge on a range of professional content creation tools, such as GarageBand, iMovie, and the Adobe Creative Cloud suite, to better understand what professional applications the study participants could engage with in the design of an assistive technology to evaluate our work.

3.2 API Design

Building on the existing assistive frameworks and APIs that exist and the feedback for opportunities in accessibility from our formative work, we began iteratively designing an API for building assistive technology. We wanted to mix what students already knew with the existing platforms for assitive technology that current existed, so that everyone from community members, students, and professionals could engage in the assistive technology design process. To have small iterative tests of new prototypes, we worked with a number of researchers who were building their own assistive technologies to implement and evaluate features using our prototype APIs. Using the feedback from our research peers, we then adapted the API.

3.3 Student Design Workshop

Once satisfied with our initial version of the API, we then designed a workshop for students to brainstorm a solution to a real accessibility problem and to prototype that solution using the V11 API. We recruited participants from our needfinding study (n = 10). Our workshop utilized a modified version of the Google Design Sprint method [26]. Students used the *Map, Sketch, Decide, Prototype, Test* structure, but the session was conducted individually for scheduling flexibility and we condensed the workshop to 90 minutes. 10 minutes were spent exploring V11 through a demonstration.

Then, participants read a brief that described the accessibility challenge they would design for. It was critical that V11 was evaluated within the context of solving a *real* accessibility challenge. Therefore, the workshop's design brief synthesized Saha and Piper's exposition of challenges for visually impaired audio engineers who use desktop audio editors [45]. Specifically, the brief focused on one challenge, that working with multiple tracks or streams of audio is difficult within audio editing applications. Participants' goal during the workshop was to use the structured design methodology to ideate and implement a solution to one aspect of this design problem for the GarageBand application.

Afterwards, 15 minutes were spent brainstorming solutions. Participants would start by writing many ideas and finish by refining them into 1-2 insights. The remaining time would be used to implement one insight using V11. While instructors could answer questions and give suggestions to a stuck participant, they were not allowed to write any code. Finally, participants filled out a reflection about V11 and their creation.

4 RESULTS & CONTRIBUTIONS

Through our formative work, we showcase the need to expose students to opportunities in accessibility across the multiple sub-fields of computer science. Given the lessons and insights we found, we then argue for and introduce V11, a design for a shared accessibility API that can empower students and communities to embed their lived experiences in the creation of personalized assistive technology. We evaluate V11 through a design workshop where introductory computer science students worked on building real assistive technology using V11. Finally, we discuss the findings of

our initial work with V11 and opportunities that we continue to explore in its design and applications.

4.1 Formative Work

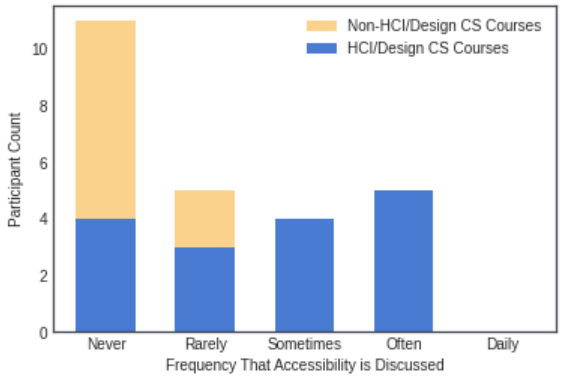

Figure 1: Accessibility discussion frequency in CS courses.

Five students noted familiarity with an accessibility standard such as WCAG or WAI. When reflecting on these standards, four students noted learning these concepts through a university class, while two students noted learning through self-exploration and two students noted learning through an internship. Nine students had completed an HCI course. Six of those students' courses included accessibility programming lessons. Finally, five students had completed a course that included lessons on disability studies.

When asked to rank the frequency of classes that included accessibility topics in HCI/Design (HCID) versus non-HCID CS courses, eleven students stated that discussions including accessibility topics are never brought up in non-HCID CS courses (figure 1). On average, students ranked that discussions surrounding accessibility occurred 26.25% more frequently in HCID CS courses than non-HCID CS courses (t=-4.05; p=0.0001). While larger studies will give better insights into accessibility exposure in CS education, this study identifies that there is a lack of non-HCID CS curriculum that incorporates accessibility. While this is quite disappointing given the applicability of accessibility in systems [19, 48], programming language [21, 46, 49], and machine learning [6, 54] courses, the current literature and critiques of accessibility in CS curricula reinforce these findings.

4.2 V11 Design & Features

Our formative work identified a clear need to develop accessibility that were similar to introductory tools found in the computer science classroom. However, there remains a high level of complexity in tools used to create accessible applications in non-web programming environments. To address this barrier, we designed a programming interface to simplify the creation of assistive technology that is available across all major platforms, easy to learn for a new CS student, and can be integrated within existing introductory curricula. Given these design requirements, we chose to abstract core assistive services from MacOS's AXUIElement, Windows' IUIAutomation, and Linux's ATK into a platform-agnostic C++ library for accessibility (figure 2). While this is useful in solving

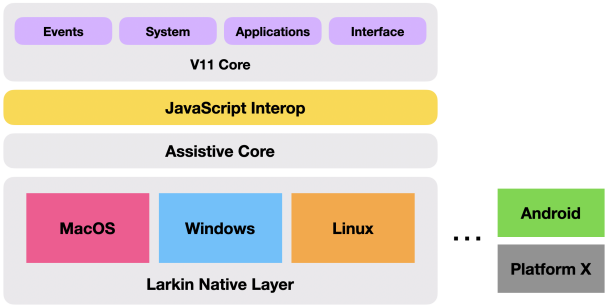

Figure 2: V11 platform software architecture.

our first requirement, there remains constraints that prevent many new CS students from engaging with the tool. C++ is a difficult language to learn, and many courses start with high-level languages such as JavaScript, Python, or Java when teaching introductory CS concepts. Additionally, C++ was not designed for querying and manipulating tree-like user interfaces and it has no native event system for performing actions when users open applications or interact with interface elements, both important components of assistive technologies.

Therefore, we built a Node.JS JavaScript wrapper for the assistive core library. Because of JavaScript's use on the web, both systems and web courses have the potential to incorporate this tool into their curriculum. Furthermore, it builds upon JavaScript's capabilities for querying and manipulating the Document Object Model (DOM), which has a similar structure to the native accessibility tree. This parallel provides a familiarity to the programming interface, as many API design decisions were based off of equivalent APIs for JavaScript's DOM interface. The resulting programming interface is V11[1], a native JavaScript library that provides a core set of components for creating assistive technology: listening for keyboard and application events, retrieving system information, querying and modifying an application's accessibility tree, and presenting information to users in both visual and auditory modalities.

4.3 Student Design Workshops

During the workshops, each participant generated an average of 4 designs and combined total of 42 (figure 3). We coded the ideas resulting in three types of projects. Information retrieval interfaces (IRIs) are systems that retrieve the state of multiple tracks without using the GUI. Example interfaces included new keyboard shortcuts and conversational interfaces. These interfaces were used to retrieve different information, such as volume levels, mute status, and track type. Task automation interfaces (TAIs) were the most common creation. TAIs reapplied IRI interfaces to automate complex tasks, such as applying effects to tracks, toggling mute, adjusting the volume, and providing shortcuts for actions. Command line interfaces (CLIs) were used as IRIs and TAIs. These systems were declared within a terminal, and they use a *command + arguments* format.

Participants rated the effectiveness of the workshop for teaching accessibility highly, with an average of 4.6/5. Additionally, before

[1]Source code and documentation can be found at: https://github.com/InclusiveTechNU/v11

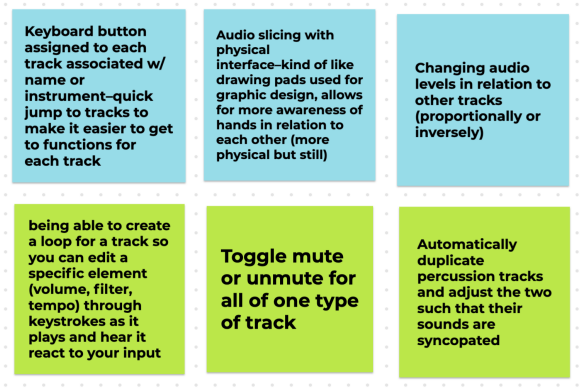

Figure 3: Designs from student brainstorming.

the study, students rated their interest in accessibility technology an average of 2.7/5. After the study, this increased to 4.1/5; a 28% increase in interest (t=-3.21; p=0.0024). The ease of learning prior-used accessibility tools was rated on average 2.7/5 and participants rated the ease of learning V11 on average 4.3/5; a 32% increase in ease of learning (t=-3.379; p=0.0016).

In written reflections, students noted that they found V11 exciting because it was familiar and they would be unsure of how to implement their designs without V11. When asked how they would build their tool without V11, one student wrote: *"I honestly would not know where to start."* Many comments similarly identified that V11 was very familiar to them. *"V11 felt very similar to the DOM model of online websites...I happened to have spent some time using plain javascript as well as jQuery, so this was not a super new concept to me - it felt familiar."* Many students also saw connections to their current CS coursework, with one student finding multiple courses that she could connect the workshop back to: *"I actually could see it in an OS class, a web dev class, and an accessibility-focused class. For OS, for example, the idea would be to use V11 to be able to inspect, access, and modify system elements...In web dev, it would be a cool extension to learn about the DOM."*

Workshop participants also found solving accessibility challenges to be very worthwhile. One student described how they would, *"Love to know more about accessibility issues with technologies I take for granted,"* while another student wrote that, *"I do have an interest in building assistive technology, but for a while I wasn't really sure if there were too many frameworks out there...V11 is great because it opened my eyes to the process and convinced me that these technologies do exist!"* While not all students were convinced that they wanted to continue building assistive technology, some did see connections in the framework that would help them support more accessibility features in their own software. One student noted: *"Even if my end goal is not to make [an assistive technology], I can always implement these ideas into projects of my own and constantly think of ways to better my projects."*

While much of the feedback was positive from the reflections, there remains room for improvement. Some participants noted that error messages were not always helpful. Additionally, there remains a learning curve. One participant wrote: *"I wish there were more examples and code snippets,"* while another student described how, *"At first I was confused on how to access certain elements...However, after about an hour or so, I actually got the hang of it and was working much faster."*

4.4 Discussion & Future Work

In the design workshops, students built new assistive technologies from ideation through implementation. Given the novice nature of our participants' programming skills, this highlights an exciting first step at introducing assistive technology design with a very low overhead. Many of these prototypes would have required hundreds of lines of code in a native accessibility API, but were written in less than twenty lines with V11.

While our study provides exciting results, there is far more work to reach our goal of community-driven assistive technology. Participants indicated areas of improvement in the design of V11 that need to be addressed. Additionally, while developing accessibility allyship through CS remains an important moral obligation, it is also critical that we empower non-programmers with disabilities to build solutions to the problems they experience by providing tools that do not require programming or by providing curriculum that targets an introductory CS curriculum towards the low overhead of V11.

REFERENCES

[1] NV Access. 2021. *NV Access.* https://www.nvaccess.org
[2] Apple. 2015. *Introduction to JavaScript for Automation Release Notes.* https://developer.apple.com/library/archive/releasenotes/InterapplicationCommunication/RN-JavaScriptForAutomation/Articles/Introduction.html#//apple_ref/doc/uid/TP40014508
[3] Apple. 2021. *Apple Accessibility - Vision.* https://www.apple.com/accessibility/vision/
[4] Apple. 2021. *AXUIElement.h.* https://developer.apple.com/documentation/applicationservices/axuielement_h?language=objc
[5] Apple. 2021. *Core Foundation.* https://developer.apple.com/documentation/corefoundation?language=objc
[6] Jeffrey P Bigham and Patrick Carrington. 2018. Learning from the Front: People with Disabilities as Early Adopters of AI.
[7] Paulo Blikstein. 2008. Travels in Troy with Freire: Technology as an agent of emancipation. In *Social Justice Education for Teachers.* Brill Sense, 205–235.
[8] Paulo Blikstein. 2013. Digital fabrication and 'making'in education: The democratization of invention. *FabLabs: Of machines, makers and inventors* 4, 1 (2013), 1–21.
[9] Nico Carpentier. 2011. *Media and participation: A site of ideological-democratic struggle.* Intellect.
[10] Robert F Cohen, Alexander V Fairley, David Gerry, and Gustavo R Lima. 2005. Accessibility in introductory computer science. *ACM SIGCSE Bulletin* 37, 1 (2005), 17–21.
[11] Beginner's Mind Collective and David Shaw. 2012. Makey Makey: improvising tangible and nature-based user interfaces. In *Proceedings of the sixth international conference on tangible, embedded and embodied interaction.* 367–370.
[12] William R Cook. 2007. Applescript. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages.* 1–1.
[13] Maitraye Das, Darren Gergle, and Anne Marie Piper. 2019. " It doesn't win you friends" Understanding Accessibility in Collaborative Writing for People with Vision Impairments. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–26.
[14] Free Desktop. 2021. *AT-SPI2.* https://www.freedesktop.org/wiki/Accessibility/AT-SPI2/
[15] Elizabeth Ellcessor. 2016. *Restricted access: Media, disability, and the politics of participation.* Vol. 6. NYU Press.
[16] Linux Foundation. 2021. *IAccessible2 API.* https://accessibility.linuxfoundation.org/a11yspecs/ia2/docs/html
[17] André Pimenta Freire, Renata Pontin de Mattos Fortes, Debora Maria Barroso Paiva, and Marcelo Augusto Santos Turine. 2007. Using screen readers to reinforce web accessibility education. *ACM SIGCSE Bulletin* 39, 3 (2007), 82–86.
[18] Gnome. 2021. *Gnome Accessibility - ATK.* https://wiki.gnome.org/Accessibility

[19] Andres Gonzalez and Loretta Guarino Reid. 2005. Platform-independent accessibility api: Accessible document object model. In *Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*. 63–71.

[20] Google. 2021. *AccessibilityService*. https://developer.android.com/reference/android/accessibilityservice/AccessibilityService

[21] Alex Hadwen-Bennett, Sue Sentance, and Cecily Morrison. 2018. Making Programming Accessible to Learners with Visual Impairments: A Literature Review. *International Journal of Computer Science Education in Schools* 2, 2 (2018), n2.

[22] Foad Hamidi, Sanjay Kumar, Mikhail Dorfman, Fayokemi Ojo, Megha Kottapalli, and Amy Hurst. 2019. SenseBox: A DIY prototyping platform to create audio interfaces for therapy. In *Proceedings of the Thirteenth International Conference on Tangible, Embedded, and Embodied Interaction*. 25–34.

[23] Amy Hurst and Shaun Kane. 2013. Making" making" accessible. In *Proceedings of the 12th international conference on interaction design and children*. 635–638.

[24] Amy Hurst and Jasmine Tobias. 2011. Empowering individuals with do-it-yourself assistive technology. In *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility*. 11–18.

[25] Henry Jenkins. 2009. *Confronting the challenges of participatory culture: Media education for the 21st century*. The MIT Press.

[26] Jake Knapp, John Zeratsky, and Braden Kowitz. 2016. *Sprint: How to solve big problems and test new ideas in just five days*. Simon and Schuster.

[27] Andrew J Ko. 2016. What is a programming language, really?. In *Proceedings of the 7th international workshop on evaluation and usability of programming languages and tools*. 32–33.

[28] Stacey Kuznetsov and Eric Paulos. 2010. Rise of the expert amateur: DIY projects, communities, and cultures. In *Proceedings of the 6th Nordic conference on human-computer interaction: extending boundaries*. 295–304.

[29] Richard E Ladner. 2015. Design for user empowerment. *interactions* 22, 2 (2015), 24–29.

[30] Michael J Lee and Andrew J Ko. 2011. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the seventh international workshop on Computing education research*. 109–116.

[31] Microsoft. 2021. *Complete guide to Narrator*. https://support.microsoft.com/en-us/windows/complete-guide-to-narrator-e4397a0d-ef4f-b386-d8ae-c172f109bdb1

[32] Microsoft. 2021. *Microsoft Active Accessibility*. https://docs.microsoft.com/en-us/windows/win32/winauto/microsoft-active-accessibility

[33] Microsoft. 2021. *Microsoft UI Automation*. https://docs.microsoft.com/en-us/dotnet/framework/ui-automation

[34] Evgeny Morozov. 2013. *To save everything, click here: The folly of technological solutionism*. Public Affairs.

[35] Brad A Myers, Amy J Ko, Thomas D LaToza, and YoungSeok Yoon. 2016. Programmers are users too: Human-centered methods for improving programming tools. *Computer* 49, 7 (2016), 44–52.

[36] Brad A Myers and Jeffrey Stylos. 2016. Improving API usability. *Commun. ACM* 59, 6 (2016), 62–69.

[37] John F Pane, Brad A Myers, and Leah B Miller. 2002. Using HCI techniques to design a more usable programming system. In *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*. IEEE, 198–206.

[38] Betsy Phillips and Hongxin Zhao. 1993. Predictors of assistive technology abandonment. *Assistive technology* 5, 1 (1993), 36–45.

[39] Cynthia Putnam, Maria Dahman, Emma Rose, Jinghui Cheng, and Glenn Bradford. 2015. Teaching accessibility, learning empathy. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*. 333–334.

[40] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

[41] Mitchel Resnick and Ken Robinson. 2017. *Lifelong kindergarten: Cultivating creativity through projects, passion, peers, and play*. MIT press.

[42] Dennis M Ritchie, Brian W Kernighan, and Michael E Lesk. 1988. *The C programming language*. Prentice Hall Englewood Cliffs.

[43] Brian J Rosmaita. 2006. Accessibility first! A new approach to web design. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*. 270–274.

[44] Abir Saha and Anne Marie Piper. 2020. Understanding Audio Production Practices of People with Vision Impairments. In *The 22nd International ACM SIGACCESS Conference on Computers and Accessibility*. 1–13.

[45] Abir Saha and Anne Marie Piper. 2020. Understanding Audio Production Practices of People with Vision Impairments. In *The 22nd International ACM SIGACCESS Conference on Computers and Accessibility* (Athens, Greece) *(ASSETS '20)*. Association for Computing Machinery, New York, NY, USA. To appear.

[46] Jaime Sánchez and Fernando Aguayo. 2005. Blind learners programming through audio. In *CHI'05 extended abstracts on Human factors in computing systems*. 1769–1772.

[47] Freedom Scientific. 2021. *JAWS*. https://www.freedomscientific.com/products/software/jaws/

[48] Robert Sinclair, Patricia M Wagoner, and Brendan McKeon. 2008. Accessibility system and method. US Patent 7,434,167.

[49] Andreas Stefik and Richard Ladner. 2017. The quorum programming language. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 641–641.

[50] Jeffrey Stylos and Brad Myers. 2007. Mapping the space of API design decisions. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*. IEEE, 50–60.

[51] Sara Vogel, Rafi Santo, and Dixie Ching. 2017. Visions of computer science education: Unpacking arguments for and projected impacts of CS4All initiatives. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 609–614.

[52] Shirin Vossoughi, Meg Escudé, Fan Kong, and Paula Hooper. 2013. Tinkering, learning & equity in the after-school setting. In *annual FabLearn conference. Palo Alto, CA: Stanford University*.

[53] WebAIM. 2000. *The WebAIM Million – An annual accessibility analysis of the top 1,000,000 home pages*. https://webaim.org/projects/million/

[54] Marcelo Worsley, David Barel, Lydia Davison, Thomas Large, and Timothy Mwiti. 2018. Multimodal interfaces for inclusive learning. In *International Conference on Artificial Intelligence in Education*. Springer, 389–393.

[55] Qiwen Zhao, Vaishnavi Mande, Paula Conn, Sedeeq Al-khazraji, Kristen Shinohara, Stephanie Ludi, and Matt Huenerfauth. 2020. Comparison of Methods for Teaching Accessibility in University Computing Courses. In *The 22nd International ACM SIGACCESS Conference on Computers and Accessibility*. 1–12.