# Assistive Technology Design as a Computer Science Learning Experience

THOMAS B. MCHUGH, Northwestern University
COOPER BARTH, Northwestern University

As many applications remain inaccessible to people with disabilities, work to integrate inclusive design into computer science (CS) curriculum has gained traction. However, there remains obstacles to integrate accessibility into introductory CS coursework. In this paper, we discuss current challenges to building assistive technology and the findings of a formative study exploring the role of accessibility in an undergraduate CS program. We respond to the observed obstacles by presenting V11, a cross-platform programming interface to empower novice CS students to build assistive technology. To evaluate the effectiveness of V11 as a CS and accessibility learning tool, we conducted design workshops with ten undergraduate CS students, who brainstormed solutions to a real accessibility problem and then used V11 to prototype their solution. Post-workshop evaluations showed a 28% average increase in student interest in building accessible technology, and V11 was rated easier to use than other accesibility programming tools. Participant Reflections indicate that V11 can be used to learn about accessibility, while also teaching CS. Finally, we detail plans on growing the V11 to foster a community of accessibility allies who build and share assistive technologies.

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

Additional Key Words and Phrases: accessibility; assistive technology; computer science education; allyship; inclusive design

**ACM Reference Format:**
Thomas B. McHugh and Cooper Barth. 2018. Assistive Technology Design as a Computer Science Learning Experience. In *ASSETS '20: ACM SIGACCESS Conference on Computers and Accessibility, October 26–28, 2020, Athens, Greece.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

## 1 INTRODUCTION

Work to improve the accessibility of software has focused on partnerships, accreditation , and standard guidelines [1, 13]. However, accessibility is far more than a list of best practices, and many technologies continue to be inaccessible to people with disabilities. The CS community must change how students are educated if accessibility is to become a core part of the design process, not just an after thought. These concerns have spurred work to integrate accessibility design skills, such as designing for user empowerment [8] and empathy building [9], into coursework, as well as integrating accessible web [10] and native application [3] lessons into curriculum.

Even so, there remains obstacles to introduce accessibility in introductory CS courses. While web programming is often used to teach accessibility and is taught in CS departments, some computer scientists feel that web programming is not appropriate for an introductory CS course [10]. However, accessibility pedagogy must seamlessly integrate within any CS learning experience, regardless of underlying computational platform, thus acknowledging that people with disabilities face accessibility challenges with technology irregardless of any convenience, or lack thereof, to teaching inclusive design with that technology. Additionally, it is often difficult for students without a disability to understand how people with disabilities interact with assistive technology [4]. While we need to foster a culture of allyship in our

CS learning environments, activities that try to simulate a disability, such as wearing a blindfold to simulate a visual impairment, disenfranchises the daily challenges that a person with a disability faces.

We use this background to motivate the design of *V11*, a programming interface for building assistive technology in the JavaScript language. To empower novice CS students to design new assisitve technology, we abstract 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 undergraduate CS students ($n$ = 10) who participated in a design workshop to brainstorm and prototype an assistive technology solution to a real accessibility challenge. In addition to a 28% average increase in student interest in building accessible technology, participants highlighted the effectiveness of V11 and the design workshop as an easy to learn platform for exploring accessibility and allyship through programming, as well as a platform to learn new computer science programming concepts. These results affirm that accessibility can, and should, be integrated into CS coursework and we discuss our plans to grow V11 to broaden access to assistive technology.

## 2 NEEDFINDING STUDY

To understand students' exposure to accessibility, we surveyed 16 undergraduate CS students about accessibility. We recruited participants through university mailing lists. Participants included four freshmen, three sophomores, five juniors, and four seniors. 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 out of all participants had completed a course that included lessons on disability studies.

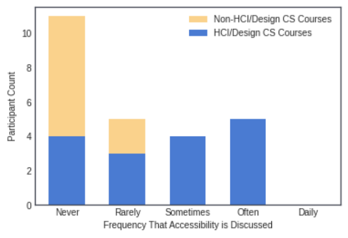

Fig. 1. Frequency of accessibility discussions in HCI/Design vs. Non-HCI/Design CS courses.

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 that included accessibility topics 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 student exposure to accessibility in CS education, this needfinding identifies that there is a lack of non-HCID CS curriculum that incorporates accessibility and that HCID-only CS courses incorporate more accessibility-related content. While this finding is quite disappointing given the applicability of accessibility in systems [5, 14], programming language [6, 12, 15], and machine learning [2, 16] courses, the current literature and critique of accessibility in current CS curricula reinforce these findings.

## 3 V11 DESIGN & FEATURES

Our formative work identified a clear need to simplify assistive technology development tools. Thus, we began designing an interface to simplify the creation of assistive technology that would be available across all major platforms, be easy to learn for a new CS student, and could be integrated within existing introductory curriculum. 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 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 course start with 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 click buttons, both important components of assistive technologies.

Therefore, we built a Node.JS JavaScript wrapper for the assistive core library. JavaScript was used because of its use on the web, so both systems and web courses have the potential to incorporate V11. Furthermore, the library can build on JavaScript's ability to query and manipulate the Document Object Model (DOM), which is similar to the native accessibility tree. This provides a familiarity to the programming interface, as many API design decisions were based off of equivalent APIs for JavaScript's web 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 accessibiltiy tree, and presenting information to users in both visual and auditory modalities.

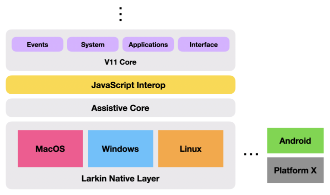

Fig. 2. A native layer of core assistive features from each platform. These are passed through a JavaScript wrapper. Finally, this is used to implement simple JavaScript components that support assistive technology design.

## 4 STUDENT DESIGN WORKSHOP

We then designed a workshop for students to brainstorm a solution to a real accessibility problem and to prototype that solution using V11. We recruited participants from our needfinding study ($n$ = 10). Our workshop modified the Google Design Sprint method [7]. 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 [11]. 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.

Afterwords, 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.

## 5 RESULTS

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, conversational interfaces, and scripting languages. These different interfaces were used to retrieve different information, such as volume levels, mute status, effects, and track type. Task automation interfaces (TAIs) were the most common type of design by participants. Many kinds of interfaces were used that were similar to IRIs, but they were reapplied to automate

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

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 are specifically declared within a terminal, and they use a *command + arguments* format.

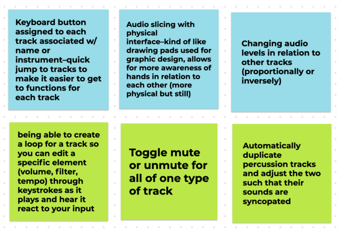
Keyboard button assigned to each track associated w/ name or instrument–quick jump to tracks to make it easier to get to functions for each track
Audio slicing with physical interface–kind of like drawing pads used for graphic design, allows for more awareness of hands in relation to each other (more physical but still)
Changing audio levels in relation to other tracks (proportionally or inversely)
being able to create a loop for a track so you can edit a specific element (volume, filter, tempo) through keystrokes as it plays and hear it react to your input
Toggle mute or unmute for all of one type of track
Automatically duplicate percussion tracks and adjust the two such that their sounds are syncopated

Fig. 3. Designs from brainstorming.

Participants rated the effectiveness of the workshop for teaching accessibility quite high, with an average of 4.6/5. Additionally, students rated their interest in accessibility technology an average of 2.7/5 before the study and an average of 4.1/5 after the study, a 28% increase in interest (t=-3.21; p=0.0024). Finally, 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. One student, when asked how they would build their tool without V11, wrote: *"I honestly would not know where to start."* Many comments built on this by identifying 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 (way back), so this was not a super new concept to me - it felt familiar."*

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 have much more interest than I did before. I think it's not only a fun technology but it also is a great way to help those in need."* Many students 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."*

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 still 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."*

## 6 DISCUSSION & FUTURE WORK

Through our analysis of V11, it is clear that CS coursework should integrate accessibility. While this work provides exciting results, there is more that can be done to continue developing the platform. Participants gave areas of improvement that need to be addressed when building new features for V11. Additionally, a study that evaluates V11's usage in a real academic learning environment will further develop the role of accessibility within CS education. Finally, 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 using tools that do not require programming. By embracing this responsibility within future designs of V11, we hope to foster a community of self-empowered users and allies who build and share assistive software to create more equitable experiences when using digital technology.

## ACKNOWLEDGMENTS

We thank all the students who participated in our study and Anne Marie Piper and Abir Saha.

## REFERENCES

[1] 2018. Web Content Accessibility Guidelines (WCAG) 2.1. https://www.w3.org/TR/WCAG21/

[2] Jeffrey P Bigham and Patrick Carrington. 2018. Learning from the Front: People with Disabilities as Early Adopters of AI.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

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

[13] Kristen Shinohara, Saba Kawas, Andrew J Ko, and Richard E Ladner. 2018. Who teaches accessibility? A survey of US computing faculty. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 197–202.

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

[15] 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.

[16] 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.