Assistive Technology Design as a Computer Science Learning Experience

A Preprint

Thomas B. McHugh
Northwestern University
Evanston, IL 60201
`mchugh@u.northwestern.edu`

Cooper Barth
Northwestern University
Evanston, IL 60201
`cfbarth@u.northwestern.edu`

Abstract

As awareness surrounding the importance of developing accessible applications has grown, work to integrate inclusive design into computer science (CS) curriculum has gained traction. However, there remain obstacles to integrating 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 undergraduate CS curriculum. 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 Computer Science concepts.

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

1 Introduction

Work to improve the accessibility of software has focused on partnerships, accreditation, and the improvement of standards and guidelines [13, 1]. However, the presentation of accessibility simply as a list of standards and best practices undercuts the importance of thoughtful and inclusive design, relegating accessibility to an afterthought during software development. This is especially apparent in computer science education; the CS community must change how students are educated if accessibility is to become a core part of the design process. These concerns have spurred work to integrate inclusive design skills such as design for user empowerment [8], empathy building [9], and the creation of accessible web [10] and native applications [3] into curriculum.

Even so, there remains obstacles to introducing accessibility into computer science curricula. While web programming is often used to introduce students to accessibility, 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. 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, disenfranchise 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 assistive technologies, 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 as an easy-to-learn platform for exploring accessibility and allyship through programming and learning new programming concepts. These results affirm that accessibility can and should be integrated into CS coursework. We then 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, recruited 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 had completed a course that included lessons on disability studies.

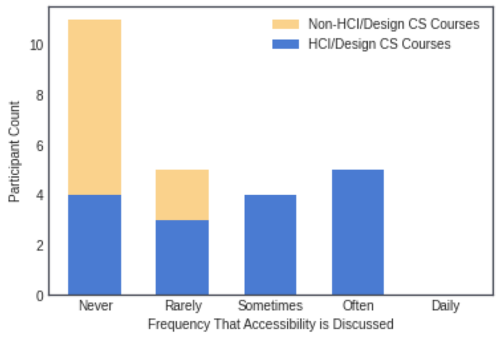


Figure 1: Frequency of accessibility discussions in HCID vs. Non-HCID CS courses.

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 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 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 is quite disappointing given the applicability of accessibility in systems [5, 14], programming language [15, 12, 6], and machine learning [2, 16] courses, the current literature and critiques of accessibility in CS curricula reinforce these findings.

3 V11 Design & Features

Our formative work identified a clear need to develop accessibility-focused curricula in a wider range of computer science courses and disciplines. 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 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 curricula. 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, 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.

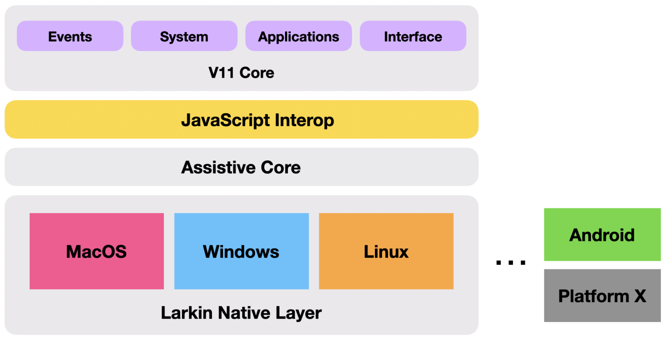


Figure 2: A native layer of assistive APIs are passed through a JavaScript wrapper. These are used to implement simple JavaScript components that support assistive technology design.

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 utilized a modified version of 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.

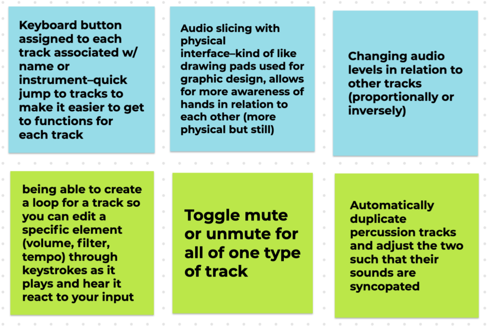
Then, participants read a brief that described the accessibility challenge they would design for. It was critical that V11 was evaluated within the context of solving a *real* accessibility challenge. Therefore, the workshop's design brief synthesized Saha and Piper's exposition of challenges for visually 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.

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.

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 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.

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).


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

Figure 3: Design ideas from brainstorming.

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, 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 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, it is clear that there is a need to integrate accessibility into range of CS curricula and that tools such as V11 can make this adoption valuable and enjoyable for students. While this work provides exciting results, there is more that can be done to expand upon this work. Participants indicated 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.

7 Acknowledgements

We thank Anne Marie Piper, Abir Saha, and all the students who participated in our study. This work was supported by NSF grant IIS-1901456.

References

[1] Web content accessibility guidelines (wcag) 2.1, Jun 2018.

[2] BIGHAM, J. P., AND CARRINGTON, P. Learning from the front: People with disabilities as early adopters of ai, 2018.

[3] COHEN, R. F., FAIRLEY, A. V., GERRY, D., AND LIMA, G. R. Accessibility in introductory computer science. *ACM SIGCSE Bulletin 37*, 1 (2005), 17–21.

[4] FREIRE, A. P., DE MATTOS FORTES, R. P., BARROSO PAIVA, D. M., AND SANTOS TURINE, M. A. Using screen readers to reinforce web accessibility education. *ACM SIGCSE Bulletin 39*, 3 (2007), 82–86.

[5] GONZALEZ, A., AND REID, L. G. Platform-independent accessibility api: Accessible document object model. In *Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)* (2005), pp. 63–71.

[6] HADWEN-BENNETT, A., SENTANCE, S., AND MORRISON, C. Making programming accessible to learners with visual impairments: A literature review. *International Journal of Computer Science Education in Schools 2*, 2 (2018), n2.

[7] KNAPP, J., ZERATSKY, J., AND KOWITZ, B. *Sprint: How to solve big problems and test new ideas in just five days*. Simon and Schuster, 2016.

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

[9] PUTNAM, C., DAHMAN, M., ROSE, E., CHENG, J., AND BRADFORD, G. Teaching accessibility, learning empathy. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility* (2015), pp. 333–334.

[10] ROSMAITA, B. J. Accessibility first! a new approach to web design. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education* (2006), pp. 270–274.

[11] SAHA, A., AND PIPER, A. M. Understanding audio production practices of people with vision impairments. In *The 22nd International ACM SIGACCESS Conference on Computers and Accessibility* (New York, NY, USA, 2020), ASSETS '20, Association for Computing Machinery. To appear.

[12] SÁNCHEZ, J., AND AGUAYO, F. Blind learners programming through audio. In *CHI'05 extended abstracts on Human factors in computing systems* (2005), pp. 1769–1772.

[13] SHINOHARA, K., KAWAS, S., KO, A. J., AND LADNER, R. E. Who teaches accessibility? a survey of us computing faculty. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (2018), pp. 197–202.

[14] SINCLAIR, R., WAGONER, P. M., AND MCKEON, B. Accessibility system and method, Oct. 7 2008. US Patent 7,434,167.

[15] STEFIK, A., AND LADNER, R. The quorum programming language. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (2017), pp. 641–641.

[16] WORSLEY, M., BAREL, D., DAVISON, L., LARGE, T., AND MWITI, T. Multimodal interfaces for inclusive learning. In *International Conference on Artificial Intelligence in Education* (2018), Springer, pp. 389–393.