Assistive Technology Design as a Computer Science Learning Experience

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

Cooper Barth
cfbarth@u.northwestern.edu
Northwestern University
Evanston, Illinois

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 an 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 accessibility programming tools. Student reflections indicate that V11 can be an accessibility learning tool, while also teaching fundamental Computer Science concepts.

CCS CONCEPTS

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

KEYWORDS

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

ACM Reference Format:
Thomas B. McHugh and Cooper Barth. 2020. Assistive Technology Design as a Computer Science Learning Experience. In *The 22nd International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '20), October 26–28, 2020, Virtual Event, Greece.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3373625.3417081

1 INTRODUCTION

Work to improve the accessibility of software has focused on partnerships, accreditation, and the improvement of standards and guidelines [1, 13]. 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.

ASSETS '20, October 26–28, 2020, Virtual Event, Greece

ACM ISBN 978-1-4503-7103-2/20/10…$15.00
https://doi.org/10.1145/3373625.3417081

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.

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

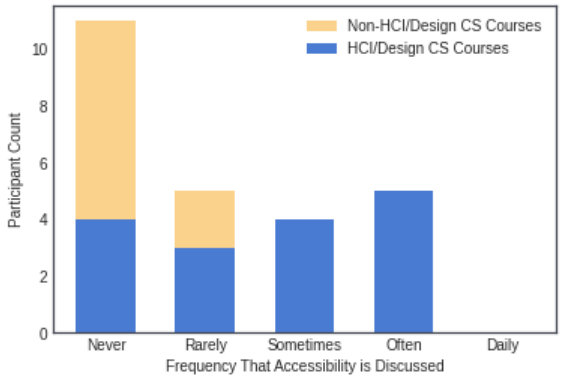

Figure 1: Accessibility discussion frequency in CS courses.

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

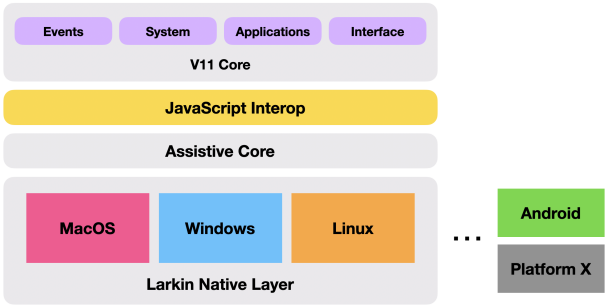

Figure 2: V11 platform software architecture.

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 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 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

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

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 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.

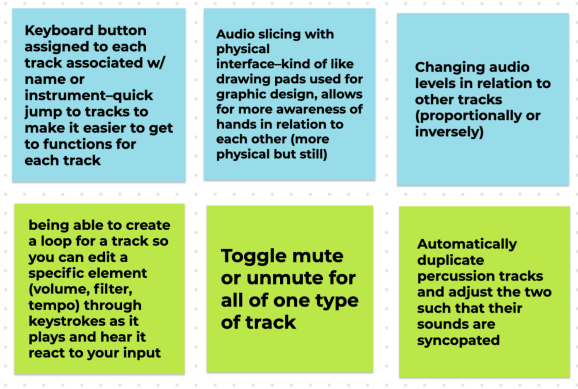

Figure 3: Designs from student brainstorming.

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

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

Our work shows that there is a need to integrate accessibility into CS curricula and that V11 could make this adoption valuable and enjoyable for students. While our study provides exciting results, there is more that can be done. Participants indicated areas of improvement that need to be addressed and a study that evaluates V11's usage in a real learning environment will resolve questions around the study's impact from experimental observation. 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 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 technology to create more equitable digital experiences.

ACKNOWLEDGMENTS

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

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.