Computer Science Principles for Teachers of Blind and Visually Impaired Students

Andreas Stefik, Richard E. Ladner, William Allee, and Sean Mealin

Abstract

The College Board's AP Computer Science Principles (CSP) content has become a major new course for introducing K-12 students to the discipline. The course was designed for many reasons, but one major goal was to broaden participation. While significant work has been completed toward equity by many research groups, we know of no systematic analysis of CSP content created by major vendors in relation to accessibility for students with disabilities, especially those who are blind or visually impaired. In this experience report, we discuss two major actions by our team to make CSP more accessible. First, with the help of accessibility experts and teachers, we modified the entire Code.org CSP course to make it accessible. Second, we conducted a one-week professional development workshop in the summer of 2018 for teachers of blind or visually impaired students in order to help them prepare to teach CSP or support those who do. We report here on lessons learned that are useful to teachers who have blind or visually impaired students in their classes, to AP CSP curriculum providers, and to the College Board.

CCS Concepts:

- Human-centered computing → Accessibility systems and tools;
- Social and professional topics → K-12 education;

Keywords:

Blind and Visually Impaired, Accessibility, Computer Science Principles, Professional Development, The College Board, Code.org.

Introduction

After years of curriculum development and piloting, the College Board has created a new AP course for high school aged students to learn computer science and potentially earn college placement or credit. Unlike AP Computer Science A, a course exclusively in Java with a traditional CS1 design, Computer Science Principles (CSP) was designed from the ground up to teach big ideas and computational practices while also being engaging to a wide spectrum of students.

One of the claimed benefits of the CSP course is its appeal to equity, meaning it should be inclusive toward all students. In this regard, data released by the College Board reports that approximately 76,000 students took the AP CSP Exam in 2018 [11]. Of these, they categorize 5,082 (6.7%) as Black/African American, 14,020 (18.5%) as

Latino/Hispanic, and 22,721 (29.9%) as Female. Change can be slow, but current data appears to be in a positive direction.

While CSP is making solid progress toward equity in regard to women and minorities, we know of no systematic work evaluating its impact on students with disabilities. After researching what the College Board makes public and asking them, they do not keep reliable data on participation by students with disabilities. This experience report focuses on the impact of the CSP curriculum on blind or visually impaired students.

To begin thinking about this group, we gathered 11 teachers of diverse backgrounds for a one-week professional development workshop for the purpose of familiarizing them with the CSP framework [10], the Code.org CSP curriculum and pedagogy [12], and the modifications made to a Code.org curriculum that we created in collaboration with accessibility experts and teachers of blind and visually impaired students. Participants in the workshop prepared lessons and taught them as a way of honing their teaching pedagogy and critiquing the original and modified Code.org CSP curriculum. We recruited from teachers who teach and/or support blind and visually impaired students exclusively and general education teachers that have no direct training with blind or visually impaired students, but that nevertheless have such students in their classrooms. We did this to ensure we were listening to perspectives of teachers that have expertise in blindness and those in general education classrooms, as we suspected they might be different.

The rest of this paper is as follows. First, we discuss related work and then the teachers themselves. Second, we discuss Code.org's CSP curriculum, including their embedded online tools, and our modifications to make it more accessible. Third we describe the workshop and some of its highlights. Finally, we outline the overall lessons learned related to accessibility of the AP CSP course.

Related Work

To our knowledge, our workshop is the first to bring in teachers from a variety of backgrounds to specifically focus on education strategies and curriculum development for blind and visually impaired students in CSP. Although working with the teachers directly seems to be relatively rare, many workshops and camps have been held for these students to increase both interest and proficiency for Computer Science and other STEM disciplines.

The data for studies focusing on teachers of blind and visually impaired students is typically obtained through interviews and observation. To understand the strategies that teachers use in mainstream classes, Lartec et al. interviewed 20 teachers with blind students [18], which uncovered several themes, such as verbalizing visual information, providing the student with additional time to understand the information, and allowing peer assistance. When interviewing instructors teaching at a STEM-focused camp for blind students [34], Villanueva et al. found that teachers needed to be aware of three main themes listed in most-to-least importance: availability of accessible resources, understanding the needs of the teacher and student, and general STEM knowledge.

Looking at Computer Science education, several studies introduced tools to increase the efficacy of students navigating code [7] and presented the results of a survey and interviews which found blind students encountered many more problems than their sighted counterparts. For example, students reported struggling with directions that relied on visual information (e.g., click on the blue button on the right) and had many accessibility problems with development environments. As a solution to this latter problem, Stefik et al. developed a programming environment designed for accessibility from the ground up, which is commonly used at residential schools for the blind [30].

As a way to provide exposure to accessible concepts and build interest, many camps have been held which focused on concepts such as data analysis and accessible visualization [17], creating interactive chatbots [9], robotics [19], and general programming [30]. Other camps have been held for general STEM disciplines, such as astronomy [8] and chemistry [37].

Workshop Staff And Teachers

The workshop staff consisted of the four authors of this report. The author Ladner participated in a Code.org professional development workshop in the summer of 2018 several week prior to the workshop. The authors Stefik and Allee are developers of the accessible tools used in the modified curriculum, and Mealin is a very experienced blind computer scientist.

As mentioned earlier, over the course of a 1-week workshop we interacted with 11 teachers from varying backgrounds. Three were teachers at residential schools for the blind, three were itinerant teachers of the visually impaired (TVIs) working as consultants or for school districts, three were teachers at mainstream public schools that have blind or visually impaired students in their class, one, who was himself blind, taught at the college level, and one was an assistive technology specialist at a school for the blind. All had experience working with blind students either individually, in a classroom setting with only blind and visually impaired students, or in a general education classroom setting with some blind students. All but four of the teachers had experience teaching academic computer science or coding as a career and technical education (CTE) teacher. Two had experience teaching CSP to at least one blind student, trying to adapt Code.org’s curriculum on their own. Table 1 shows the background of each teacher that participated.

We recruited this diverse group, working with blind students in different ways and with different levels of computer science knowledge, intentionally to learn a diversity of perspectives. As a result, we gathered varied feedback from those that were experts in blindness, but knew less about adapting computer science principles, to the opposite, with general education teachers that were intimately familiar with CSP, but did not necessarily know how to modify curriculum to blind or visually impaired students. This diversity helped us gather varied feedback on 1) the Code.org curriculum 2) The College Board’s procedures in regard to blindness, and 3) our modifications for making it accessible.

The Code.org CSP Curriculum

The Code.org CSP curriculum is very much a discovery and inquiry curriculum, whereby in each lesson students are presented with a computing problem to solve. This is done either as an "unplugged" activity, using handouts or other offline materials, or as a "plugged" activity, using the online tools provided by Code.org. The activities are typically completed in pairs or small groups. The discovery activities are guided by the teacher, but students are not told how to solve the problem. The teaching philosophy for student activities is sometimes called Think-Pair-Share. This means students first work individually or in small groups thinking about a solution to a problem, perhaps using trial and error or logical thinking, while sometimes taking notes in a journal. Second, in pairs or small groups, students compare and combine solutions. Finally, students share ideas with the entire class. In this stage there is no correct answer. The goal is for students to share possible solutions.

The discovery and inquiry approach leads to the concepts that the students should be learning, but not necessarily to the technical language that is found in textbooks or on the AP CSP exam. The structure of a lesson follows the pattern "Activity - Concept - Vocabulary." The purpose of the activity is to ground a concept in a concrete way, then move to the technical vocabulary used more generally in the computer science field after. In this way, students gain a deeper understanding of the meaning of the vocabulary rather than being potentially intimidated by it at the beginning.

There are a total of 79 lessons found in 8 units in the Code.org CSP curriculum. The online curriculum covers the 7 big ideas and 6 computational thinking practices found in the College Board AP CSP framework [10]. Briefly, these big ideas are Creativity, Abstraction, Data and Information, Algorithms, Programming, the Internet, and Global Impact. Those and the computational practices embody how the College Board sees the content of a high school computer science course. The 8 units in the Code.org CSP curriculum for 2018-2019 are titled:

- Unit 1 - The Internet
- Unit 2 - Digital Information
- Unit 3 - Intro to Programming
- Unit 4 - Big Data and Privacy
- AP Explore Performance Task Prep
- Unit 5 - Building Apps
- AP Create Performance Task Prep
    - Post AP (done after taking the AP exam in early May)

The curriculum makes extensive use of online tools. They include five versions of an Internet Simulator used in Unit 1, a text compression widget and two pixelation widgets used in Unit 2, the App Lab programming environment used in Units 3 and 5, and the encryption widget used in Unit 4.

Table 1: Background of Teachers

Job Description	School Setting	CS Teaching
CTE Teacher	School for the Blind	Coding
Computer Science Teacher	Public School	AP CSP and AP CS A
Math/CS Teacher	School for the Blind	CS
CTE Teacher	School for the Blind	Coding
Itinerant TVI	School District	None
Itinerant TVI	School District	None
Assistive Technology Specialist	School for the Blind	None
Consultant TVI	Independent	None
CS Teacher	College Level	CS
CS Teacher	Public School	CSP
CS Teacher	Public School	CS

Accessibility Of The Curriculum

In order for an online curriculum to be accessible to blind and visually impaired students, it must be compatible with modern screen readers such as JAWS [25] and NVDA [1] on PCs and VoiceOver [6] on Mac. It must also be compatible with the magnification tools built into PCs and Macs and add-on tools such as ZoomText [27]. For an online curriculum like Code.org's, this is accomplished by adhering to the WCAG 2.1 AA guidelines [36], and by utilizing the ARIA standard [35] for interactive web pages. This is not a straightforward process, requiring a fairly high level of accessible web development expertise. In addition to technical accessibility, there can also be usability issues even when these guidelines are followed. For example, a web page might be WCAG compliant, but still difficult to use through a screen reader.

Generally speaking, Code.org's curriculum is not accessible to students who are blind and use screen readers. With their permission, we first analyzed all of the lessons and content and tracked what was and was not accessible. We provide here an overview of our findings.

To begin with, the online website for the course is often not compliant with WCAG 2.1, which means that people with disabilities may not be able to use it. Examples of this might be as simple as missing alternate descriptions for an image, to more complex minutia related to WCAG that are not crucial to describe in this report. Second, for the lessons themselves, many of the unplugged activities use visual metaphors and artifacts. To be clear, blind and visually impaired people can certainly interpret visual metaphors if the content is accessible, but that does not mean it is easy.

Because this requires some explanation, consider that some of these visual concepts in the curriculum could be made to have physical metaphors as an alternative. For example, Unit 1 Lesson 5 of Code.org's curriculum provides students with a "Flippy Do" template [13], which is used to teach binary numbers. Students are asked to cut and fold a piece of paper along printed lines and fill in blank spaces. Modifying this lesson could involve a variety of possible changes. One could remove the Flippy Do, modify a digital version for use with a screen reader, or create a Flippy Do template using Braille. In our modification, we decided to be pragmatic. We wrote changes to the text to make the Flippy Do less necessary, but also provided a Braille Flippy Do as an alternative.

So far as we could tell, all of the visual, and incredibly creative, online interactive tools in the Code.org curriculum are not screen reader accessible. In particular, the App Lab programming tool is not accessible even though there was a way to move from a blocks-based code to and from text-based JavaScript code, that could, in principle, be accessible. However, accessing that text through App Lab prevents that text from being sent to the screen reader. To put the problem plainly, a blind user using App Lab JavaScript mode through a screen reader would hear no sound, regardless of the fact that it is just text.

By blocks-based environment, we mean a programming setting where visualization and direct manipulation are used to write programs, as opposed to writing text. While the reader might assume that a highly visual blocks-based programming environment might be inaccessible to the blind or visually impaired, and this assumption is correct, there has been some progress on making blocks-based environments accessible. For example, Milne and Ladner [20] created an accessible blocks-based environment for younger children. This is possible to do, so long as appropriate accessibility standards are followed on a per platform basis.

While we point out that current blocks-based environments are not accessible, we do think it is important to mention evidence from Weintrop [38, 39], which shows that they do have some benefits to learners. Effect sizes in such studies are relatively small, with some showing no effect and others showing a small one. Our interpretation of that work is that, depending on how blocks-based programming environments are used, they overall have a limited, but real, positive impact on learning at the high-school age. However, they also exclude both our population entirely and some others with disabilities not discussed in this report (e.g., those with certain physical disabilities).

Making The Curriculum Accessible

The goal of our modified version of the Code.org CSP curriculum was to adopt the "discovery and inquiry" pedagogy and overall educational philosophy of the curriculum but convert all the online lessons to accessible formats using the WCAG 2.1 accessibility guidelines and ARIA as our initial guide. This meant finding alternative unplugged activities for the strongly visually oriented ones and developing alternative interactive tools to provide pragmatic solutions for our population.

The modified curriculum was done prior to the workshop. For nearly a year, we employed a team of three teachers of blind and visually impaired students to modify the curriculum across the board, focusing especially on the lessons and their unplugged activities to make accessible alternatives. At the same time, our development team collaborated with outside experts and developed equivalent accessible interactive tools as replacements for the inaccessible ones in the Code.org CSP online curriculum. We wrote our own accessible versions of many of them as replacements for the inaccessible tools. Our replacements work by creating similar tools in the Quorum web-based programming environment that is already WCAG 2.1 compliant. For each of these tools, the Quorum source code is available to students or teachers who can make changes to customize it or make it better in some way. The modified curriculum uses the Quorum programming language environment instead of App Lab for the programming development lessons especially in Units 3 and 5.

As one final point on Code.org's curriculum and the complexities surrounding it, we want to be very careful in stating that these past two sections of our paper were not intended to be critical of any CSP curriculum development group. In particular, Code.org actively requested our team to evaluate their curriculum for accessibility and has been overwhelmingly supportive of our explicit search for what is and is not currently accessible. We imagine many other curriculum providers are in a similar situation. We would point out that because of this collaboration, and that of our teachers and students, all of the materials and tools described in this section are available for use at the time of this writing.

Quorum Programming Language

The Quorum programming language is a "born accessible" programming ecosystem, meaning it was accessible from its formation and has accessibility as a first-class design goal. By our own internal tracking, Quorum is used in approximately half of residential schools for the blind and visually impaired in the United States. The language is not solely for individuals with visual impairments, but it is inclusive of that group. The Quorum language has a wide number of applications, including LEGO robotics programming, digital signal processing, 2D/3D graphical, physics simulations, 3D positional audio or other programs—all of which we have made accessible.

The Quorum programming language is also classed as an "evidence- oriented" programming language [29], which means its core syntax and semantics are derived from both technical specifications and human-factors data. For those unfamiliar with evidence-oriented programming, a recent Dagstuhl was conducted on the topic [28]. Language constructs like "for", for looping, are intentionally not included, and were replaced in favor of notation choices with more obvious semantic identity, like "repeat." These choices are not arbitrary and were derived from measurements of human productivity, not just the opinions of the designers. In addition to making the language easier for programmers to understand, reducing syntactic clutter in the language (such as removing unnecessary parenthesis and end-of-line symbols) makes code easier to understand when using accessibility technologies such as screen readers.

While the core of our curriculum is online and we support accessibility in this sense through standard mechanisms, some teachers prefer offline tools because screen reading technologies on desktop are mature. For this reason, users can either program online using our editors or they can use Sodbeans, an offline plugin for the NetBeans development environment that adds Quorum and accessibility support. Our team is also creating a new environment, called Quorum Studio, that bypasses the standard Java Accessibility bridge for a custom accessibility layer. We call our approach "accessible graphics mapping."

Briefly, the way the accessible graphics mapping works, which is available today and which Quorum Studio is being built upon, is to have an accessibility intermediate layer that combines native call-downs to accessibility systems on a per-platform basis. This might sound trivial, but there is no common intermediary for accessibility without it. Every platform, to our knowledge, is incompatible with accessibility systems on every other. From there, a graphics pipeline, like OpenGL or DirectX, "maps" to the intermediate layer.

On Microsoft platforms, for example, it is easy to create an application that has "buttons" on a "form" that are accessible. However, it is significantly more difficult to make graphical applications, like those in games, accessible, even if they also have items that look and act like buttons. On Windows, Quorum tackles this problem by connecting to what Microsoft calls UI automation.

Thus for accessible graphics mapping, Quorum users first create standard events (e.g., button clicks, typed keys), which are automatically sent to the intermediate layer and translated into native accessibility events. After translation, the intermediate layer tells the operating system that graphical components are "acting like" various accessibility systems, allowing bi-directional control. This creates an operating system specific accessibility event stream.

At the native level, accessibility technologies like screen readers pick up on this event stream, which makes a graphical component appear to be a native accessible application. Thus, whether the user created a button or a complex 2D or 3D shape, the user obtains information in a similar way. The full technical details of this system are too complex to describe in this short paper. Suffice to say that while versions of the technology are available today in Quorum 6, they are complex and will evolve over time.

Highlights Of The Workshop

The workshop was modeled after the Code.org TeacherCon conference that was held in June 2018 in Atlanta, GA. A major component was the use of their Teacher-Learner-Observer (TLO) model whereby the teachers were placed into pairs who prepared and co-taught a lesson from the Code.org CSP curriculum. With 11 teachers, one teacher was paired with our blind staff member to form six pairs. Each pair was assigned one Code.org CSP lesson to prepare and teach where the resource for the lesson was a combination of a Code.org lesson plan and a modified accessible lesson plan. Each pair was asked to prepare a 40-minute accessible lesson using whatever resources they wanted from the two lesson plans, and in addition, teachers were asked to be creative in making the lesson as accessible as possible. Because so many of our teachers were experienced with blind and visually impaired students, many new ideas to improve the modified accessible lesson plans emerged. While each pair taught a lesson, the remaining 9 participants played the role of students. Among the 9 participants there were always one or two who were blind. This forced teachers to take into account that they have blind students in their class. The remaining staff members stayed in the room as observers, helping participants remain in their roles. The observers' took notes and provided additional feedback when the lesson was completed.

At the end of each lesson, one of the observers and the pair of teachers left the room to reflect on their teaching in terms of what choices they made and why they made them, with a particular emphasis on making the lesson accessible. At the same time, the 9 "students" in the lesson reflected on what they learned and what might have been improved to make the learning experience better. After about 5 to 10 minutes, everyone joined back together to share their reflections with each other. During this final reflection period many new ideas came out about how to make the lesson more accessible and engaging for the students.

In addition to the TLO activities, there were other activities we focused on generating ideas for improving the modified curriculum, to strengthen the teachers' capacity to teach blind and visually impaired students, and to prepare the students for the AP CSP Exam. These activities included:

- Overview of the history of CSP and its framework.
- Overview of the Code.org CSP curriculum and its accessible modification.
- Overview of the Explore Performance Task and discussion of interesting topics that might interest blind and visually impaired students.
- Overview of and practice with the Quorum programming environment.
- Overview of and practice with two Quorum-based online educational tools.
- Demonstration of SAS Graphics Accelerator [24] for accessible big data analysis.
- Overview of the Create Performance Task and discussion of interesting non-visual programming tasks that blind and visually impaired students might develop.
- Discussion led by our blind staff member on highlights of his career in computer science including obstacles he has encountered and how they were overcome.
- Discussion and critique of the AP CSP Exam led by teachers who have taught CSP.
    - Discussion of how to build a computer science community among teachers of blind and visually impaired students.
    - Discussion of proper ways to interact with students who are blind or visually impaired, led by TVIs.

Lessons Learned

Although the primary purpose of the workshop was professional development for teachers of blind and visually impaired students, there was much to learn from these teachers that apply to any teachers who have blind or visually impaired students in their classes, to AP CSP curriculum providers, and to the College Board. In this section, we review some of these lessons learned.

Lessons for Teachers

It is important to recognize that students who are blind or visually impaired generally have the capability of completing the AP CSP course. Including a student who is blind or visually impaired who is prepared intellectually and has facility with technologies like screen readers or magnification tools need not be an excessive burden provided the curriculum itself is accessible. Most of these students in mainstream settings have Teachers of the Visually Impaired (TVIs) who work with students to improve their access to academic subjects. AP CSP teachers should work closely with the TVI to help make their class as accessible as possible and to learn some of the best ways to interact with these students. The TVI may or may not know about accessible educational tools for computer science that are available to their students. Here is a short list of accessible educational tools that were discussed during the workshop:

- The Quorum Language for general programming [33]
- SAS Graphics Accelerator for data analysis [24]
- Instructional Material Centers as a resource to find accessible materials for any course [16]
- Flying Blind (Top Tech Tidbits in Assistive Technology) [14]
- AccessCSforAll to find a hotline for individual support for teachers [3]

Some advice that came from the workshop teachers is to be flexible. If an activity needs to be modified or replaced, just do it. Share your modification or replacement with others to save them time. If a tool can't be used by one of your students, report this to the curriculum provider so that they can improve it.

Lessons for AP CSP Providers

A major goal of AP CSP is broadening participation, which means the inclusion of all students regardless of gender, race, ethnicity, socioeconomic, or disability status. This goal cannot be achieved without attention to the accessibility of the curricula.

A first step would be to do an evaluation with regard to accessibility. This evaluation should be done by a qualified accessibility professional to be as complete as possible. After the evaluation, revisions of a curriculum should be prioritized using criteria based on the impact on students and teachers. A revision could be as simple as providing an alternative accessible unplugged activity. For the Explore Performance Task, include examples that relate to accessibility. These examples will invigorate all students, disabled or not, to think outside of the box. Examples mentioned by work- shop participants included content like the Graphiti refreshable tactile graphics display [5], Orcam MyReader [23], Smart cane [4], Tap wearable keyboard [32], or Sunu Band [31].

For the Create Performance Task, it might be necessary to provide an alternative and accessible programming environment. Programming environments that only produce visual animations can be limiting for all students. Programming environments that control robots and/or audio are reasonable alternatives, but not all platforms are equally accessible. The broader point is that curriculum should not be exclusively in one sensory modality. Even if one is perceived to be engaging, like visuals genuinely can be, content providers need to be creative and engage in varying sensory modalities if they wish to be general purpose and for all students.

This is an issue of equity and the law, since students with disabilities have the legal right to participate in K-12 education through section 504 [21]. Some states have been more explicit. For example, in Nevada, SB-200 [26] requires all computer science courses at the K-12 level to be accessible. To our knowledge, this is the only state to call out this requirement for computer science explicitly as of today, although we would be unsurprised if other states followed suit given the overall goal of equity that is often discussed as part of computer science education.

Lessons for The College Board

The teachers in the workshop had three categories of advice for the College Board, which were related to practice examples, accommodations, and the AP CSP Exam. When coming to their conclusions, teachers had full access, and many knew, the College Board's existing accessibility and accommodations guidelines [2]. Generally, there is a need for more clarity and acknowledgement that students who are blind or visually impaired are included in their advice to students in their documentation. In the sample tests and questions, as it stands now, teachers found that they did not convincingly take disability status into account. Many examples were purely visual, with no alternatives, and it was unclear to them whether this might deflate a blind student's exam score without just cause.

Teachers had a number of concerns with the practice examples provided by the College Board. These centered largely around the fact that much of the content provided was purely visual. In theory, teachers recognized the College Board "might" accept non-visual alternatives, but the guidelines were vague or led to other concerns. Examples might include, would video with scrolling text or a spoken podcast be acceptable formats for the Explore Task? We would encourage the College Board to re-think such issues.

Teachers had many questions about accommodations, but they boiled down to what kind of help a blind or visually impaired student can receive while doing either of the Performance Tasks. For example, if a pertinent phrase is listed in a student's Individualized Education Program (IEP), like that the teacher is able to describe visuals to them, can they have similar accommodations in the explore/create task (e.g., a teacher verifying a camera is not pointed at the ceiling, putting a visual box or circle around lines of code specified by the student)? There is no acknowledgements for exceptions in the "musts, may, and may not" list for teachers involvement in performance tasks. Considering that about 13% of K-12 students in the United States have an IEP [15], more clarity in addressing the needs of these students in taking AP CSP and other AP courses should be addressed.

There was considerable concern by the teachers about the use of Braille math on the exam. There are currently two competing standards: Nemeth and Unified English Braille (UEB). Depending on which state the student is from, they learn one or the other, yet the College Board uses only Nemeth for CSP. Teachers urged us to make it clear that this matters a great deal. A student who knows UEB, but not Nemeth, would have a truly extraordinary barrier placed in front of them and their exam scores may not reflect their abilities. These dialects of Braille are quite different and knowing one hardly means that a student is familiar with the other. Overall, as the Braille Authority of North America points out, the situation regarding Nemeth and UEB is complicated [22].

There is also concern about whether any practice exams are available in appropriate Braille formats and how equity is achieved as certain questions are omitted from the exam for blind test takers. For example, if statistical procedures at the College Board do not take into account the "ease" of a question when dropping questions for reasons of disability, this could provide students a non-equitable advantage or disadvantage. In either case, the statistical evidence for the decisions made needs to be publicly available for scrutiny.

Roadmap

As one final consideration in this work, we want to at least mention what we think needs to change for computer science to be accessible in K-12. First, the AP course aside, a great deal of the tools used by students in K-12 are not accessible to blind or visually impaired students. This includes all of the ones used commonly, like Scratch, SNAP!, Alice, and many others. This needs to change. All of Quorum's accessible output systems are open source and the ideas, or even code, can be adapted freely.

Part of the issue with accessibility is that it is a second class citizen. We regularly hear about equity for women and underrepresented minorities and we agree this is critical. However, if we are to embrace equity and computer science for all, we have to really mean it. The College Board, for example, should not endorse any curriculum, for any discipline, that is not accessible. Accessibility can be the default if we want it to be.

Conclusions

We held a one-week professional development workshop in the summer of 2018, the goal of which was both to train teachers of blind and visually impaired students and evaluate accessibility in CSP. This included a modification of Code.org's online curriculum for this population, in addition to analysis of it by 11 teachers. Overall, we found that while many lessons can be modified, the CSP community has significant challenges ahead. Notably, curriculum today has systemic barriers including the lack of engaging non-visual alternatives. Many tools are not compatible with web standards such as WCAG 2.1 AA. We think making progress on this will take dedicated effort, but with evidence in hand on the barriers, we think teachers, content providers, and others now have a roadmap for what challenges to consider first.

Acknowledgments

The work is supported by the National Science Foundation under Grant numbers 1738259 and 1738252. We would like to especially thank Baker Franke at code.org for his kind and helpful comments throughout this project.

References

1. NV Access: Empowering Lives through Non-Visual Access to Technology. https://www.nvaccess.org/. Retrieved August 28, 2018
2. AccessCollegeBoard 2018. *AP SSD Guidelines*. https://apcentral.collegeboard.org/pdf/ap-ssd-guidelines-2017-18.pdf . Retrieved October 30, 2018.
3. AccssCSforAll Website. https://www.washington.edu/accesscomputing/accesscsforall. Retrieved August 31, 2018.
4. Access CS ForAll. [n. d.]. AssisTech. Retrieved August 31, 2018 from http://assistech.iitd.ernet.in/smartcane.php
5. Inc. American Printing House for the Blind. [n. d.]. Graphiti refreshable tactile graphics display. Retrieved August 31, 2018 from https://www.aph.org/graphiti/
6. Apple. [n. d.]. Accessibility. Retrieved August 28, 2018 from https://www.apple.com/accessibility/mac/vision/
7. Baker C.M. 2017. Increasing Access to Computer Science for Blind Students. *SIGACCESS Access. Comput.* 117 (Feb. 2017), 19–22. https://doi.org/10. 1145/3051519.3051523
8. Beck-Winchatz B. and Riccobono M.A. Advancing participation of blind students in Science, Technology, Engineering, and Math. *Advances in Space Research* 42, 11 (2008), 1855–1858. 2008.
9. Bigham J.P, Aller M.B, Brudvik J.T, Leung J.O, Yazzolino L.A, and Ladner R.E. Inspiring Blind High School Students to Pursue Computer Science with Instant Messaging Chatbots. *SIGCSE Bull.* 40, 1 (March 2008), 449–453. doi:10.1145/1352322.1352287 2008.
10. The College Board; AP Computer Science Principles. Retrieved August 28, 2018 from https://apcentral.collegeboard.org/courses/ ap-computer-science-principles
11. The College Board; Number of Girls and Underrepresented Students Taking AP Computer Courses Spikes Again. Retrieved August 31, 2018 from https://www.collegeboard.org/ membership/all-access/counseling-admissions-financial-aid-academic/ number-girls-and-underrepresented
12. Code.org; Computer Science Principles. Retrieved August 28, 2018 from https://code.org/educate/csp
13. Code.org; Lesson 5: Binary Numbers. Retrieved August 28, 2018 from https://curriculum.code.org/csp-18/unit1/5/
14. LLC Flying Blind; Flying Blind Empowerment Through Technology. Retrieved August 31, 2018 from http://www.flying-blind.com/
15. Fast Facts, Students with Disabilities. National Center for Education Statistics. Retrieved August 28, 2018 from https://nces.ed.gov/fastfacts/display. asp?id=64

16. Instructional Resource Centers for the Blind and Visually Impaired. American Federation for the Blind. Retrieved August 31, 2018 from http://www.afb.org/info/afb-national-education-program/ national-instructional-materials-accessibility-standard-nimas/ instructional-resource-centers/235
17. Kane S. and Bigham J. Tracking @Stemxcomet: Teaching Programming to Blind Students via 3D Printing, Crisis Management, and Twitter. *In Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. ACM, New York, NY, USA, 247–252. doi:10.1145/2538862.2538975. 2014
18. Lartec J.K. and Espique F.P. Communication Strategies of Teachers Educating Students Who Are Legally Blind in the General Education Setting. *Insight* 5, 2 (2012), 70. 2012.
19. Ludi S. and Reichlmayr T. The Use of Robotics to Promote Computing to Pre-College Students with Visual Impairments. *Trans. Comput. Educ.* 11, 3, Article 20 (Oct. 2011), 20 pages. https://doi.org/10.1145/2037276. 2037284. 2011
20. Milne L. and Ladner R. Blocks4All: Overcoming Accessibility Barriers to Blocks Programming for Children with Visual Impairments. *In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (CHI '18). ACM, New York, NY, USA, Article 69, 10 pages. doi:10.1145/3173574.3173643
21. Protecting Students With Disabilities, Frequently Asked Questions About Section 504 and the Education of Children with Disabilities. *U.S. Department of Education*. Retrieved August 28, 2018 from https://www2.ed.gov/about/ offices/list/ocr/504faq.html
22. Braille Authority of North America. Unified English Braille (UEB). Retrieved August 28, 2018 from http://brailleauthority.org/ueb.html#plans
23. OrCam MyReader 2 Website. Orcam. Retrieved August 31, 2018 from https://www.orcam.com/en/myreader2/
24. SAS Graphics Accelerator for data analysis. SAS. Retrieved August 31, 2018 from http://support.sas.com/software/products/graphics-accelerator/index. html
25. Freedom Scientific's JAWS home page. Freedom Scientific. Retrieved August 28, 2018 from https://www.freedomscientific.com/Products/Blindness/ JAWS
26. "Senators Woodhouse; Denis; Ford; Spearman; Cancela; Carlton; Frierson; Atkin- son; Cannizzaro; Gansert; Manendo; Parks; Ratti; Segerblom; and Fumo". *Senate Bill No. 200.* Retrieved August 28, 2018 from https://www.leg.state.nv.us/ App/NELIS/REL/79th2017/Bill/5073/Text
27. Ai Squared. Zoom Text. Retrieved August 28, 2018 from https://www.zoomtext.com/
28. Stefik A., Sharif B., Myers B., and Hanenberg S. Evidence About Programmers for Programming Language Design (Dagstuhl Seminar 18061). *Dagstuhl Reports* 8, 2 (2018), 1–25. doi:10.4230/DagRep.8.2.1
29. Andreas Stefik and Susanna Siebert. 2013. An Empirical Investigation into Programming Language Syntax. Trans. Comput. Educ. 13, 4, Article 19 (Nov. 2013), 40 pages. doi:10.1145/2534973
30. Stefik A., Hundhausen C., and Smith D. On the Design of an Educational Infrastructure for the Blind and Visually Impaired in Computer Science. *In Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, New York, NY, USA, 571–576. doi:10.1145/1953163.1953323 2011.
31. The Sunu Band. Sunu. Retrieved August 31, 2018 from https://www.sunu.io/index.html
32. Tap Wearable Keyboard, Mouse and Controller. Tap. Retrieved August 31, 2018 from www.tapwithus.com

33. The Quorum Language. Quorum Development Team. Retrieved August 31, 2018 from https://quorumlanguage.com/
34. Villanueva I. and Di Stefano M. *Narrative Inquiry on the Teaching of STEM to Blind High School Students.* Education Sciences 7, 4 (2017), 89. 2017.
35. W3C. Accessible Rich Internet Applications (WAI-ARIA) 1.1. Retrieved August 28, 2018 from https://www.w3.org/TR/wai-aria-1.1/
36. W3C. Web Content Accessibility Guidelines (WCAG) 2.1. Retrieved August 28, 2018 from https://www.w3.org/TR/WCAG21/
37. Wedler H.B., Boyes L., Davis R.L., Flynn D., Franz A., Hamann C.S., Harrison J.G., Lodewyk M.W., Milinkevich K.A., Shaw J.T., Tantillo D.J., and Wang S.C. *Nobody Can See Atoms: Science Camps Highlighting Approaches for Making Chemistry Accessible to Blind and Visually Impaired Students*. Journal of Chemical Education 91, 2 (2014), 188–194. 2014.
38. Weintrop D. *Comparing Text-based, Blocks-based, and Hybrid Blocks/Text Programming Tools.* In Proceedings of the Eleventh Annual Inter- national Conference on International Computing Education Research (ICER '15). ACM, New York, NY, USA, 283–284. doi:10.1145/2787622.2787752
39. Weintrop D. and Holbert N. *From Blocks to Text and Back: Programming Patterns in a Dual-Modality Environment.* In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17). ACM, New York, NY, USA, 633–638. doi:10.1145/3017680.3017707. 2017.

Author Details

Andreas Stefik
Department of Computer Science
University of Nevada, Las Vegas
Las Vegas, NV USA
stefika@gmail.com

Richard E. Ladner
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, WA USA
ladner@cs.washington.edu

William Allee
Department of Computer Science
University of Nevada, Las Vegas
Las Vegas, NV USA
wallee777@hotmail.com

Sean Mealin
Department of Computer Science
North Carolina State University
Raleigh, NC USA
spmealin@ncsu.edu