CAN: Composable Accessibility Infrastructure via Data-Driven Crowdsourcing

Yun Huang
School of Information Studies
Syracuse University
yhuang@syr.edu

Brian Dobreski
School of Information Studies
Syracuse University
bjdobres@syr.edu

Bijay Bhaskar Deo
Electrical Engineering and Computer Science
Syracuse University
bbdeo@syr.edu

Jiahang Xin
Electrical Engineering and Computer Science
Syracuse University
jxin@syr.edu

Natã Miccael Barbosa
Centro Universitário - Católica de Santa Catarina
Brazil
nata.barbosa@catolicasc.org.br

Yang Wang
School of Information Studies
Syracuse University
ywang@syr.edu

Jeffrey P. Bigham
HCI and LTI Institutes
Carnegie Mellon University
jbigham@cmu.edu

ABSTRACT

Despite persistent effort, many web pages are still not accessible to everyone. Fixing web accessibility problems can be complicated. Developers need to have extensive knowledge not only of possible accessibility problems but also of approaches for fixing them. This paper is about using the large number of accessibility issues on real websites and crowdsourced fixes for them as a unique source of learning materials for web developers to learn how to build accessible components in a cost-efficient manner. In this paper, we present the design, development and study of CAN (Composable Accessibility Infrastructure), a crowdsourcing infrastructure that collects web accessibility issues and their fixes, dynamically composes solutions on-the-fly, and delivers the crowdsourced content as teaching materials. Our unique CAN user interaction and system design enables end users with disabilities to both benefit from and contribute to the system without additional effort in their daily web browsing, and allows web developers to experience real accessibility issues and initiate a learning process with first-hand materials. CAN also provides an opportunity for data-driven discovery of the common implementation practices that cause accessibility issues. We show how CAN addresses a set of accessibility issues on the top 100 popular websites. We also present our user study results where web developers who had varying knowledge of web accessibility all found our system an effective and interesting platform to learning web accessibility.

W4A '15, May 18 - 20, 2015, Florence, Italy.

http://dx.doi.org/10.1145/2745555.2746651.

1. INTRODUCTION

Many people experience accessibility issues as they browse the Web that either prevent them from accessing Web content or reduce their efficiency in doing so. While some users can quickly navigate through existing web pages to find the information that they need, many people have difficulty in doing so because of how web pages are created. For instance, if the web component cannot be focused by the TAB key or a screen reader can not read the information that it contains, some users will be unable to efficiently access the underlying information or functionality. Because there is a large variety of web implementation technologies available, if web developers are not deeply aware of web accessibility issues and methods for fixing them, it can be difficult to recognize and fix the breadth of problems that can occur.

It is surprising how persistent even basic web accessibility problems are. We checked the top 100 popular websites [10] in November 2014. 59 of them have keyboard accessibility issues, and that many of the issues were different from site to site. As a result, certain web content on those sites is not accessible to people who rely on the keyboard; 35 of them had missing alternative text, which means that blind users using screen reader software would not be able to access some information; and, 32 of them had color contrast issues which means that some people with low vision may have difficulty accessing their content. Although an overall decline in accessibility issues in top traffic sites since 1999 has been noted, the rate of accessibility violations on even popular sites remains high [27]. Despite years of effort, the Web is still largely inaccessible.

A variety of initiatives and projects have attempted to correct this problem. Efforts have been made to involve disabled users in reporting accessibility issues, but research studies [38] have found that end users were not always effective at recognizing and reporting problems. When they were, it is often difficult to find someone who can fix the problem [19]. Many developers still do not realize the impor-

tance of accessibility. Even though efforts have been made to highlight the importance of accessibility in computer science education [32, 40], research suggests that more effective training on web accessibility is needed for professionals to go beyond simple awareness [34]. There is also a current lack of tools designed specifically to teach accessibility [12]. Tools designed to identify or even fix accessibility issues are limited in scope.

In this paper, we present a system called CAN (Composable Accessibility Infrastructure) that is designed to work within the difficult and messy reality of web accessibility. It is inspired by successful applications of crowdsourcing in many different application domains [43]. Accessibility work has been increasingly turning towards crowdsourcing models as a means of pulling in experts and helping distribute the responsibility of accessibility [16, 37, 28].

CAN builds from prior approaches that used crowdsourcing as a way to improve web accessibility, but it does so in a way that can drive future data-driven approaches for fixing problems and educating developers. In particular, CAN collects a variety of web accessibility issues on real websites and their software fixes, dynamically composes fix solutions on the fly and delivers the crowdsourced content as teaching materials to improve web accessibility. The reality of web accessibility issues in the real world also suggests the feasibility of the idea. Many of the web accessibility issues reveal common problems in web development practices, where one software solution may fix the same issue across different websites. However, the same problem on two different websites may be caused by different implementation details that need totally different solutions, and fixing all the issues on one website may need a varying combination of software solutions. The large number of accessibility issues on real websites and their fixes can serve as a great source of materials for web developers to learn how to build accessible components.

Major contributions of this paper are as follows. First, CAN proposes a novel user interaction and system design, which removes the burden of reporting issues from the end user with disabilities and allows end users to both benefit from and contribute to the system without incurring any additional effort in their daily web browsing activities. Secondly, CAN crowdsources real issues and their fixes to serve as educational materials, realizing a more low-cost approach than creating new cases only for education purposes in or out of the classroom setting. Thirdly, a lightweight infrastructure design that composes fix solutions on the fly maximally and dynamically leverages individual open source software contributions in addressing real web accessibility issues. Fourthly, with the increasing knowledge of real web accessibility issues, better understanding of their implementation challenges and larger collections of different solutions, CAN provides a great opportunity to address more complex accessibility issues as well as a data-driven means of discovering the common practices that cause them. Finally, CAN's open APIs reduces the technical barrier for web developers to deliver accessible web products by integrating CAN's solution in their program.

2. RELATED WORK

Our work builds from work on *(i)* accessibility standards and tools, *(ii)* prior approaches for crowdsourcing improved accessibility, and *(iii)* accessibility education.

2.1 Accessibility Standards and Tools

WCAG (Web Content Accessibility Guidelines) is a well-accepted set of guidelines for web accessibility design, published by W3C (Worldwide Web Consortium) [20]. The cornerstone of these guidelines are a set of four principles, which can be shortened as "POUR." Perceivable: all the information and elements on the website are guaranteed to be perceived by all users, regardless of limitations such as vision and hearing. Operable: users can interact with all the controls in the UI components in the way that operations are intended. Understandable: content is clear and straightforward to the public without causing any confusion. Robust: the content can be interpreted by most user agents and assistive technologies. If any of these are not true, users with disabilities will not be able to access the web effectively. The CAN project looks into WCAG guidelines as a basis for generating requirements for the proposed framework. User agents are software to render the web content, including Web browsers, media players, and assistive technologies. The User Agent Accessibility Guidelines (UAAG) guides developers in making such software accessible [30]. This means users can easily access the user agents and the features built in the agents. In the design of CAN, we implement a browser plugin designed to be easily accessible for users with disabilities. WAI-ARIA, the Accessible Rich Internet Applications Suite, defines approaches to make Web content and more accessible [23]. It is widely used in dynamic content and advanced user interface controls developed with Ajax, HTML, JavaScript, and related technologies. It is aimed to make content accessible to assistive technologies (AT), software that can be used by people with disabilities, such as screen readers and screen magnifiers. Modern AT always support ARIA specifications, leading developers to take ARIA into account in web design. The core of the ARIA specification is a set of attribute semantics consisting of role, properties, and states. Technically, WAI-ARIA combines two specifications in the coding: Roles, States and Properties Rules include "menuitem", "application", and "form", states and properties contains "aria-checked", "aria-sized", etc. As this specification does not affect the functionality in code, many website designers ignore this, which is bad for the AT such as screen readers. CAN implements these roles or properties to help detect the accessibility issues according to WCAG.

Tools to detect accessibility issues serve an important role. Research by Alonso et al. [13] determined that manual testability of WCAG is unreliable, suggesting the need for effective evaluation tools. A number of free evaluation sites exist, such as AChecker [8]. It can check and validate many aspects related to web design and rendering, and allows developers to upload their HTML files to check the accessibility before publishing the website. Tools like AEA (Accessibility Evaluation Assistant) have been developed specifically to assist novices in recognizing accessibility problems [31]. Benavídez et al. [14] explored the use of several accessibility tools including HERA, a tool designed to assist in manual accessibility evaluations capable of providing some guidance on resolution. Evaluation tools are helpful for detecting problems, but most are unable to fix them. Recent work on tools like SourceProbe has explored automated processes of evaluation and resolution [41]. Existing tools (e.g. AChecker [8]) could be integrated into CAN as modules.

Recently, projects like the Global Public Inclusive Infrastructure (GPII) [7] and the associated Cloud4all [4] have

suggested a broader approach to accessibility tools, proposing an infrastructure which allows augmentation of various accessibility services and features as needed by the users. We view CAN as complimentary to these efforts and expect that many of the ACs will be utilized in both systems. The main difference is CAN's heavy focus on a crowdsource-based approach to identifying and resolving accessibility problems. CAN is also designed to support learning during website programming so that developers can build related skills.

2.2 Crowdsourcing Accessibility

With the growing understanding that accessibility needs on the Web have become so broad and complex that they cannot be the sole responsibility of the original developer, researchers have begun turning towards collaborative and crowdsourced models of accessibility. Various collaborative projects have demonstrated effectiveness in dealing with accessibility issues. The Social Accessibility project was designed to allow web users to communicate with volunteer developers regarding specific accessibility concerns [37, 35]. Subsequent evaluation of Social Accessibility found it a time effective model of correcting a variety of accessibility problems [38]. Accessibility Commons was created as a collaborative infrastructure within which accessibility metadata can be created and shared freely [28]. Similarly, Prasad et al. [33] explored the use of collaboratively produced external metadata to improve Web accessibility. Many crowdsourced accessibility models rely on volunteer web developers who are already able to recognize and fix such problems. For example, Accessmonkey [16] allows web users and developers to work together to solve accessibility problems. Though some problems, such as missing alternate text, could be resolved in a semi-automated way, other issues required volunteer developers to independently know how to resolve them.

The crowdsourcing model we design in CAN particularly emphasizes reducing the burden of reporting from end users and how one particular contribution potentially benefits a large number of websites and inexperienced web developers.

2.3 Accessibility Education

Literature on computer science education has increasingly highlighted the importance of accessibility topics within the past ten years. Cohen et al. [22] describe efforts to incorporate accessibility topics into an undergraduate CS curriculum, emphasizing the role and importance of hands-on projects. Carter and Fourney [21] outline a specific course created to bring students beyond simple awareness of accessibility, and develop real skills for addressing accessibility issues. Other works have focused on the incorporation accessibility topics and projects throughout multiple courses during the CS program [39]. Much research has focused on the role of accessibility in computer science education within the formal structure of college courses, with comparatively little focus on self-directed accessibility education or web development learning outside of the classroom. Similarly, research into specific techniques of teaching accessibility is still developing. Recent work continues to emphasize the importance of accessibility in computer science education, noting the particular effectiveness of accessibility exercises and projects [32, 40]. Gellenbeck [26] notes the value of accessibility audits and evaluations during the educational process. Freire et al. [25] also discuss the use of accessibility diagnostic tools, as well as the use of screen readers to help students discover accessibility problems. Al-Khalifa and Al-Khalifa [12] note the lack of educational tools to teach accessibility in a more modular way, presenting their work on AEG (Accessibility Example Generator), a tool designed for classroom uses that allows educators to call upon specific problems when teaching students how to resolve accessibility issues.

We designed a novel interaction for teaching web accessibility in CAN, where web developers experience real web accessibility issues and interact with CAN by leveraging the crowdsourced materials (e.g. websites from end users and source code from open source developers).

3. CAN DESIGN AND IMPLEMENTATION

There are three major users in CAN (Composable Accessibility Infrastructure) with their own interests, i.e. (1) end users who need websites to be accessible; (2) web developers who may not know web accessibility and need quick solutions to fix their web page problems, or may have interests in learning how to implement the solutions by themselves; (3) open source contributors who share their code to fix accessibility issues through CAN. In CAN, each open source code is called an Accessible Component (AC), usually addressing one particular accessibility problem and implemented using a particular method.

3.1 User Interactions and Workflow

Based on our prior literature review and the needs of different users, we define the following design goals of CAN user interactions. (a) End users should have no additional effort with using CAN, because their highest demand is to access the web components, and they have limitations on identifying accessibility issues (for example, if they visit a page, they do not know if there is a drop down menu if the drop down menu is not reachable at all), and it will cost them much more time to report an issue than regular web users. (b) CAN AC contributors should be able to contribute any independent functional modules that fix certain websites' accessibility issues, and able to update their code without worrying about how the code is executed on others' webpages. (c) CAN interaction with web developers should be designed to follow the general practice of web development. Generally speaking, if web developers do not have accessibility knowledge, but they are required to make their products accessible, they usually develop the pages first, then use some open tool (e.g. WAVE [11]) to check the problems on their page, and then fix the individual problems. The way they approach the problems may be more in a passive manner, i.e. addressing the problem when it happens. Thus CAN needs to provide user interactions with web developers when they try to test their webpages.

Figure 1 presents a vision of CAN framework. The four boxes in the greyed layer illustrate four interfaces for the three types of users. End users take a one-time installation effort to install CAN's *End User Browser Plugin*. Since then, whenever the end users open a new webpage link, CAN services will be executed without end users' awareness. Running the browser plugin, end users both benefit from the system directly and contribute to the system implicitly, because the plugin checks for accessibility compliance and logs accessibility problems of the websites to the *Issue Repository*.

The second group of users who directly contribute to our systems are open source software developers. They can share

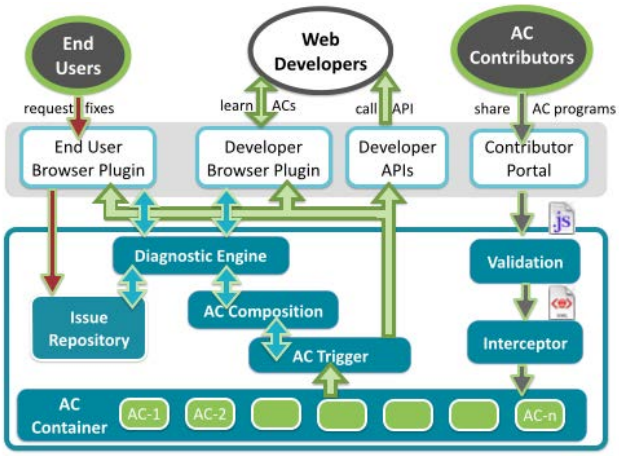

Figure 1: The CAN Framework. End users contribute different websites' accessibility issues implicitly, and the AC contributors contribute ACs to the system explicitly.

the code via the *Contributor Portal*. The CAN *Validation* component checks the uploaded script using pre-defined validation rules. Once the ACs are approved, the *Interceptor* configures them and stores them in the *AC Container*.

Web developers can benefit from using CAN services via two user interactions, i.e. reactive mode and proactive mode. In the reactive mode, they may have developed a web page and want to check and fix its accessibility issues, or simply want to find out how to fix certain issues in some others' websites. They can initiate a dialog with our system by turning on the *CAN Developer Browser Plugin*. The CAN web developer plugin first diagnoses their webpages, then can call the CAN fix function to have detected problems fixed. The dialog is executed through the browsers' developer console. CAN also provides developers the link to the source code, so that they can read on and understand how the problems are fixed. They can choose to leave the issues as they are, as the end users can have the CAN end user plugin fix the issues on-demand. They can also choose to revise their code and fix the problems by themselves or call CAN developers' API [36] directly. In the proactive mode, when a web developer implements his web apps, they can directly call individual CAN AC's API, such that they do not need to implement the solutions themselves. This way, they can avoid potential problems proactively. Providing these two interaction modes to web developers, CAN allows web developers to have to choice of focusing on their website design and development effort without worrying about developing their own solutions.

Because each website can have different accessibility issues, existing ACs in CAN may not be able to solve all the issues. When either of the *Browser Plug-in*s are triggered, a *Diagnostic Engine* will identify which issues can be fixed by CAN. Then an *AC Composition* component will take the report of the *Diagnostic Engine* and compose the ACs together by requesting a *AC Trigger* module to load all the ACs from the *AC Container*. Eventually the composed ACs will be delivered and executed via the *Browser Plug-in*s.

In the next section, we will present detailed descriptions of the system components.

3.2 Key System Components

There are many technical challenges that need to be addressed to realize the above workflow. In this section, we discuss the key system components related to the crowdsourcing features of CAN. For example, the *Diagnostic Engine* involves collecting and analyzing crowdsourced accessibility issues from end users; and the AC *Validation* and *Interceptor* check the crowdsourced code from the open source contributors and try to address conflicts or updates of these contributions. The CAN server platform and components are built on Node.js, and current CAN accepts AC code written in JavaScript.

3.2.1 Diagnostic Engine

There are many web-based and open source tools (e.g. Wave, Achecker [8]) that help identify accessibility issues on a webpage. They may also suggest simple fixes to address the issues, e.g. adding an alternative text for an image. The CAN *Diagnostic Engine* leverages existing open source accessibility checking tools, but only relying on the existing tools will not provide sufficient features, because many times, complex issues are not able to be identified, especially those involving dynamic web content, e.g. a drop down menu. When two different websites show the same symptom, e.g. the drop down menu is not reachable by the keyboard, the underlying implementations may be different, therefore the ACs that can be used to fix those issues will not be the same. Additionally, different websites may need different sets of ACs to address their problems, and it is not necessary to load all the ACs from the *AC container* to users' webpage. The *Diagnostic Engine* enables the implementation of a light-weighted framework, i.e. only loading the ACs that are needed.

Being able to clearly identify the issues using the *Diagnostic Engine* not only fulfills the above goals in regards to system performance, but also helps deliver the educational goals for the web developers. The *Diagnostic Engine* enables a dialog with web developers, informing them which AC fixes which accessibility issues on the tested webpage.

When the system is deployed in real world with large number of end users using the system, the *Issue Repository* can help identify common issues that are not fixed by the *Diagnostic Engine*, providing a great opportunity to address more complex issues. In the result section, we will illustrate different demo scenarios with real popular websites. We will show different implementation details that could cause the same symptom, and how a common web accessibility issue is not easily detected.

3.2.2 AC Validation and Interceptor

In order to enlarge the community of the open source AC contributors, CAN does not require the open source contributors to have any knowledge of cross browser extension SDK. The contributors can focus on developing the ACs assuming they are interacting with the web page directly. As presented in the above sections, CAN system takes care of wiring the ACs with the browser plugin. However, while contributing the ACs, open source contributors need to follow certain rules that CAN defines, so that their contributed code can be integrated and executed by CAN. For example, each AC should have a single entry function that can be triggered to execute its function. Such validation rules are checked in the *Validation* component after an AC JavaScript

is uploaded by the contributor.

In addition, as discussed in the prior literature, different accessibility projects can be disconnected from one another. Accessibility Commons [28] was designed to collect metadata of these project efforts, so that web accessibility can be improved by merging existing disconnected contributions. Similarly in CAN, in order to resolve potential conflicts between crowdsourced ACs, AC contributors should provide basic information of their ACs in a configuration file that can be read by the *Interceptor*.

```
{
 "name":"Dropdown Menu",
 "description":"render the dropdown menu implemented by CSS selector",
 "version":"0.1",
 "web-browser":"Google Chrome",
 "browser-version":"38.0.2125.111",
 "main-function":"init",
 "included-websites":["http://www.bloomberg.com/"],
 "excluded-websites":["http://www.walmart.com/"],
 "last-tested-timestamp":"11/08/2014"
}
```

Figure 2: A sample configuration file

A sample configuration file is presented in Figure 2, and such a configuration file should be uploaded together with the AC JavaScript code. For example, in the configuration file, the value of the "main-function" is configured as "init", namely, the init function is entry point of the AC.

Oftentimes, one AC may work for one particular implementation scenario of an accessibility issue, but does not work for other implementation scenarios or does not work in all browsers. The AC contributors should already know which websites the AC works with and may even know which websites the AC does not work with. AC contributors can specify these cases in the configuration file. For example, the "excluded-websites" attribute tell the CAN system to exclude the AC from executing for the list of websites such as walmart.com. The "included-websites" direct the system to execute the AC for that website such as "bloomberg.com". The *Interceptor* takes the configuration file and applies all the rules to check whether the component is eligible to be executed for the website accessibility issues. The current rules are basic. For example, the code has to compile successfully, and be free of global variable naming issues that would hinder it from bei ng integrated with other ACs. These validation and configuration processes allow the *AC Composition* to identify the right ACs.

4. SYSTEM EVALUATION

As part of the pilot system, we released two chrome browser plugins on the Chrome Web Store, i.e. one for end users to fix web pages [3] and the other one for web developers to learn CAN ACs [2]. The end user plugin is enabled after installation, and the developer plugin can be triggered with one TAB key access or a shortcut key, just one-click-away from reaching CAN. The shortcut key can be defined by users in their browser settings after they install the plugin. The interfaces for open source developers to contributor their code and configuration files are presented in Figure 3 and Figure 4.

CAN is currently created to work best with the Google Chrome web browser, used in conjunction with screen reader software, such as JAWS when used in Windows and VoiceOver when used on a Mac. The CAN end user plugin was designed and tested to ensure that it works with screen

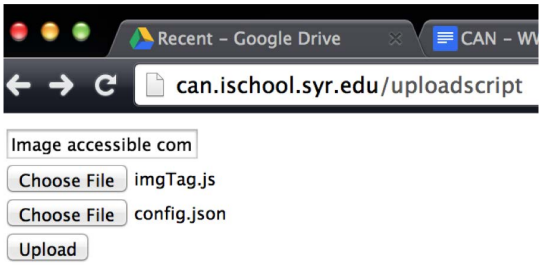

Figure 3: Web Portal for AC Contributor.

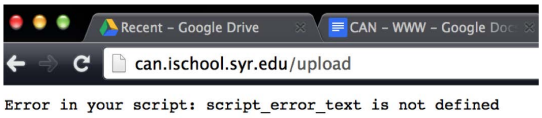

Figure 4: Sample Validation Output.

readers and when using only the keyboard. CAN will eventually support Firefox and Internet Explorer as well, and we are in the process of developing these plugins.

In this section, we focus on presenting the results that serve the following objectives: *(i)* justify that the CAN crowdsourcing framework enables a flexible infrastructure that can address accessibility issues at multiple levels, *(ii)* demonstrate the potential for CAN to provide feasible solutions in the real world, *(iii)* motivate the novel methodology enabled by CAN for teaching web accessibility.

More specifically, we showcase how CAN can compose and deliver three levels of ACs. First, we present how we bootstrap the crowdsourcing system with two ACs that address very common web accessibility issues that can be detected by existing tools [8]. Then we show how CAN fixes inaccessible web components, e.g. dropdown menus, which are not easily detected by existing tools. Further we present an AC that improves end users' webpage login experience that seems not to be an accessibility issue, but is a real challenge as revealed by intensive user studies. We show the implementation challenges and the promising results, where each AC can fix the common issue of a large portion of the first 100 popular websites [10] that we have discussed in the introduction. Finally, we present the developer user study, showing the effectiveness of using this crowdsourced real case platform to deliver web accessibility knowledge and discuss participants' positive and encouraging feedback.

4.1 Demo Scenarios of Popular Websites

The assumption of CAN is that one AC crowdsourced from open source developers can potentially fix the issues on many other sites. We now use real popular websites to showcase CAN composes ACs that can address accessibility issues at different levels on-the-fly.

4.1.1 Basic ACs for Easily Identified Issues

As we discussed in the introduction, there are widely applied common issues among popular websites. To bootstrap CAN, we developed two ACs that fix two common issues, missing alternative image texts and poor color contrast, which can be easily identified by existing tools [8].

AC-1 (Alternative Image Text): According to WCAG guidelines, all website images should have a meaningful and simple "alt" - alternative text, so that screen reader software can convey information in the image to blind users,

but many websites fail to do so. We implemented an AC that builds on the methods introduced by WebInSight[17, 18] to automatically provide alternative text. First, the AC goes through all the "img" tags to find the ones that miss the "alt" values. Then the AC tries to find an appropriate expression to define the "alt" value by checking four conditions, which help identify a relatively meaningful expression to define the "alt" value. First, the AC tries to trim the source of image into a simple expression; the second tries to find the class name of this img element itself; the third condition is to retrieve the "href" information from its parent element of "img" and trim it; and the fourth one is to find the "id"and "class"above the img tag. If any of the conditions is satisfied, then the "alt" value can be identified and the AC will not check the remaining conditions; otherwise, the AC will finish checking all the conditions. If none of them is qualified, the AC will take the source of the image as a temporary value to define the "alt" value. This intuitive algorithm may not produce the optimal output, but it is a starting point that suffices in some cases. This AC could be optimized by applying a text mining algorithm to look up the context around the image and suggest appropriate "alt" values [18].

AC-2 (Color Contrast): In order to make websites visually attractive, web developers may choose color settings that do not comply with guidelines. However, about 8% of men and 0.5% of women have color blindness [5]. It may be difficult for them to distinguish between different objects on websites if the colors are not set appropriately. To resolve this issue, we have created the second AC that adjusts the color contrast.

The *Diagnostic Engine* scans through the website, and checks the background and font color pairs which do not comply with the WCAG 2.0 guidelines. It calculates color brightness and difference of the text objects. Then it compares these values with the standard values [1]. If calculated values are less than standard values (e.g., the color difference is less than 500 and the color brightness is less than 125), it means that the colors are not distinguishable and color blind people cannot identify those objects. As an initial solution, the AC-2 changes the color pairs to be a black and white combination (which may involve updating the CSS file of the website). After executing AC-2, the website can become accessible to color blind people.

4.1.2 ACs for Fixing Complex Issues

Existing checking tools can not detect all accessibility issues, especially involving dynamic web components. For instance, existing tools may not be able to identify if a dropdown menu is keyboard accessible. This needs open-source contributors to manually test the websites and contribute their ACs that can fix the issues.

AC-3s (Dropdown Menu): The most common implementation of dropdown menu can be illustrated using the case of scientificamerican.com. Even though this site is not one of the top 100 popular websites, we started the first AC development with this site and later we found this AC widely adopted by many other sites. For example, the web view and its source code are presented in Figure 5. The menu main items (e.g., News & Features, Topics, etc.) are $< li >$ elements and submenu dropdowns are $< ul >$ elements (e.g. starting from the *Latest Stories* to the *Interactive Feature*). The main menu items have event listeners such as mouseover and mouseout events to hide and display submenu whereas the submenu items have no events attached. Hence, the AC-3-dropdown-Mouseevent sets the tabindex and attaches a jQuery focusin event and a jQuery focusout event to main menu items, such that the menu and submenu are accessible by the TAB key and the mouseover and mouseout events triggered by the mouse can also be done by keyboard.

Figure 5: Dropdown menu in scientificamerican.com

Unlike scientificamerican.com, another more complex implementation for dropdown menu is to use CSS hover selector. Taking bloomberg.com as an example, we developed an AC-3-dropdown-CSS. The typical hover implementation in this scenario is "li:hover ul" , in which the "li" element is the item in the navigation bar, and the "ul" element is the dropdown menu. The "hover" should be fired when the TAB focus is on the "li" element, but the current website fails to do so. The AC that fixes this dropdown issue needs to enable "tab" to fire the "hover". However, there are usually two major issues to address. First, the AC needs to search through the "style" definition and find out the CSS hover using the DOM function "document.styleSheets" and "document.styleSheets[i].cssRules". If the web designers put the CSS hover to a <style> definition and we can directly extract them. In general web designers prefer to encapsulate CSS part into a separate file located in a different domain from the current website. Thus accessing this file will lead to "Cross-origin" problem. In the future production system, CAN needs to realize similar functions like what the Chrome extension "Allow-Control-Allow-Origin" does to bypass this problem. Secondly, as the CSS file usually is a plain text, the AC cannot apply DOM function on it, so the AC should package the whole file content into a <style> part. Then the AC can access CSS hover selector. After the AC locates the "li:hover ul", it converts it into a class-style expression "ul.hover ul",then the AC can attach this class into jQuery focusin() and focusout() function to enable the TAB key to behave like a mouse hover.

Overall, the same symptom of inaccessible dropdown menus can be a result of different development practices and the solutions to fix those issues vary significantly in terms of the level of complexity. Mouseevent-driven can be a more common scenario and easier to fix. CAN platform provides a great opportunity to fix more complex issues.

4.1.3 ACs for Enhancing Web Experience

The three ACs discussed above fix accessibility errors, which prevent people from accessing the web components. In addition to the ACs that fix accessibility errors, CAN also

tries to crowdsource ACs that can improve end users' web browsing experience.

For example, user study sessions with blind and low-vision users conducted by our prior research project [24] have brought to attention that those users have a hard time trying to reach the login form when relying on screen reader software to interpret the page. This is because even though the login form seems positioned at the very beginning of the webpage to regular users, it is usually not the first web component the screen reader software would read. Moreover, when the user interface of the login page had changed (e.g. user got redirected to a different page due to an error), users have to learn and understand the changes through screen reader software, which oftentimes can be confusing because they have already memorized how that page was usually read. Additionally, users complained that the terms used for logging in are not standardized (e.g. some websites use "user name and password", other websites use "email and password"). Theses all make it even harder for them to find and interpret the login form.

These motivated us to develop a Login First AC, which tries to detect a login form on a page. The current algorithm finds the form whose children inputs are one password field and one other visible field of text or email type. Once a login form is detected, the AC then clones the original form fields into a new, clean and accessible form that can be easily interpreted by screen readers, and most importantly: will be the first element to appear in the DOM tree, leading screen reader software to read the cloned form first. This enables end users to reach the login form and interpret the web components effortlessly. Once the user enters their credentials into the new, cloned form and confirms that those credentials are good (i.e. presses the Enter key), the values entered are then copied to the original form on the page and the action to submit the form can be triggered either manually by the user, or automatically (this could be configurable by the user). This process happens entirely on the client-side (i.e. the browser), not affecting the login workflow already in place in the website being visited.

4.1.4 *Effectiveness of CAN ACs*

We revisited the same top 100 popular websites [10] that we have been discussing, checked if there were issues or needed improvements that our four ACs could be applied to, and tested the CAN end user plugin using both JAWS in Windows and VoiceOver on Mac.

At the time we tested on Nov. 10, 2014, among the 100 popular websites, 7 of them would not need any of the four ACs. Namely, these websites do not have missing alternative text for images, color contrast issues, dropdown menu issues or a login form. However, 16 of them would need one AC; 20 of them would need two ACs; 38 of them would need three ACs; and 19 of them would need four ACs.

More specifically, among the 100 sites, 35 of them had the issues of missing alternative texts for images. The CAN AC-1 fixed all of them. Note that 10 websites did not support the jQuery libraries AC-1 uses. Without CAN end user plugin loading the libraries, the AC would not be able to fix those 10 websites. When testing the color contrast AC, 68 of total 100 websites did not have any color contrast issues, and 32 of them had color contrast issues based on the algorithm applied in the AC. The AC-2 fixed 30 of them and did not work with two websites (e.g., typepad.com and zc2.com).

Website	CAN plugin fixes	Optimization
amazon.com	Yes with optimization	$li- > a$
imdb.com/	Yes	none
about.com	Yes	none
nih.gov	Yes	none
huffingtonpost.com	Yes	none
ebay.com	Yes with optimization	$li- > td$
washingtonpost.com	Yes	none
forbes.com	Yes with optimization	$li, li- > a$
usatoday.com	Yes	none

Table 1: Websites are listed in order of popularity. 6 websites can be fixed using the existing CAN ACs directly, and 3 of the above can be fixed with a little optimization to the existing AC-3-dropdown-Mouseevent.

Among these 100 websites, there were 22 websites with dropdown menus, but only two websites' dropdown menus were keyboard accessible (i.e. technorati.com and networkadvertising.org). 20 of them had accessibility issues. Table 1 shows the websites that can be fixed by AC-3-dropdown-Mouseevent. In essence, these websites have common issues that can be fixed by attaching events to the web elements. Another 11 websites' dropdown menu problems cannot be fixed with either AC-3-dropdown-Mouseevent or AC-3-dropdown-CSS (that fixes bloomberg.com), and need new AC solutions.

When Login First AC was tested on the 100 websites, 11 out of the 100 websites could not be tested because of multiple reasons (e.g. the website did not have a login form, the website did not respond at that time). The remaining 89 websites were then tested and the component successfully prioritized the login form on 73 websites. The component failed to prioritize the login form on 16 websites. Figure 6 shows the login form is cloned at the very beginning of the webpage. Note that it is not necessary to change the view for blind users, because screen reader software would interpret the DOM structure and does not require the element to be visible. However, such a change was an intent to draw web developers' attention to the issue, and to raise awareness on how they could optimize their design to implement more accessible login pages.

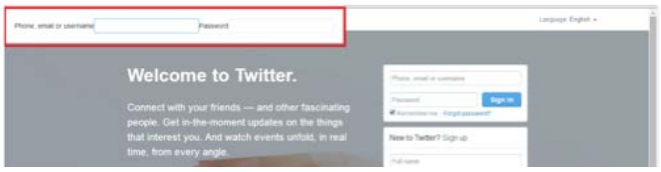

Figure 6: Accessibility enhancement to user login

4.2 Web Developer User Study

To evaluate how web developers would perceive and use CAN, we conducted a set of preliminary user studies focusing on how they use the web developer plugin and if they like this learning process. We intended to recruit web developers of varying experience from our university's community who had previous experience with JavaScript. At the beginning of each study, participants came to the lab, read and agreed to the consent form, and were given a brief

pre-survey to fill out. The pre-survey asked basic questions about web development experience and experience with accessibility issues. Following the survey, participants were shown the website scientificamercian.com and asked to navigate the site using the mouse or trackpad. Then participants were asked to navigate the menus using only the tab key to simulate a user with screen reader software and asked if they noticed an accessibility problem (i.e. inability to access the dropdown menus). Participants were then asked if they would be able to correct the problem, and if not, how they would go about learning how to do so. Then, they were instructed to initiate the CAN web developer plugin and follow the directions by finishing a dialog with the plugin. After implementing CAN's solutions, participants were asked to view the source code and explain their understanding of the solution. Afterwards, participants were interviewed with follow-up questions concerning their thoughts on CAN and how they typically learn web development skills. At the conclusion of the study, participants received five dollars for their participation. The study was approved by our university IRB office beforehand.

Participants consisted of nine web developers ranging from college juniors to professionals who had completed graduate degrees. Experience in developing web applications ranged from two months to five years. When asked to describe their knowledge of web accessibility, seven responded they had little knowledge or beginner's level knowledge. Only the most experienced participant had any prior experience solving an accessibility problem.

When first tasked with fixing the dropdown menu accessibility problem, none of the participants were immediately able to fix it. Five of the participants' suggestions involved redesigning the website rather than quickly fixing the dropdown menu. When asked how they might then learn how to correct the problem, seven of the participants reported they would do a web search to find solutions online. Participants' searches included "*how to access drop down menu by keyboard,javascript drop down menu from keyboard,*"and "*make drop down tabs screen reader accessible.*"Participants searching for focus or mouseover events discovered relevant material online. The student developers who had little experiences were unable to report the amount of time they required to continue reviewing solutions and begin testing them out. The experienced developer P8 could figure out within 1 hour, and P1 with 3 years web development experience said he could figure out the solution within a day.

Figure 7 shows the pop up message that helps developers understand how to start the dialog with CAN. Figure 8 shows a sample interaction scenario between the web developer and CAN, where web developers follow the prompt instruction to understand the identified web accessibility issues on the page.

After using the CAN web developer plugin, all participants were satisfied with the solution provided to fix the dropdown menus. In general, participants with less experience and less exposure to the concept of accessibility reported less understanding of CAN's solution. The participants who felt they could now recognize and correct the dropdown menu problem on their own after reading CAN's solution had more web development experience and some previous exposure to accessibility concepts.

After interacting with CAN, participants reported they were more aware of and concerned with accessibility. In addition, all participants reported that they would use a system like CAN if they needed to conduct accessibility work, though P8 reported he would only want to use CAN to identify issues and would prefer to fix them himself. Suggestions to improve CAN centered around messages and feedback information. One participant requested to "*make the output more detailed*," providing further explanation of what was changed and how exactly it was done. Another participant found the feedback helpful but suggested CAN "*offer notification but without popups, maybe just a thin banner at the top*" to show something had happened, perhaps with a button to undo CAN's changes. Future work on CAN will attempt to incorporate some of these suggestions.

Figure 7: A pop up window suggesting the developers to open the console window.

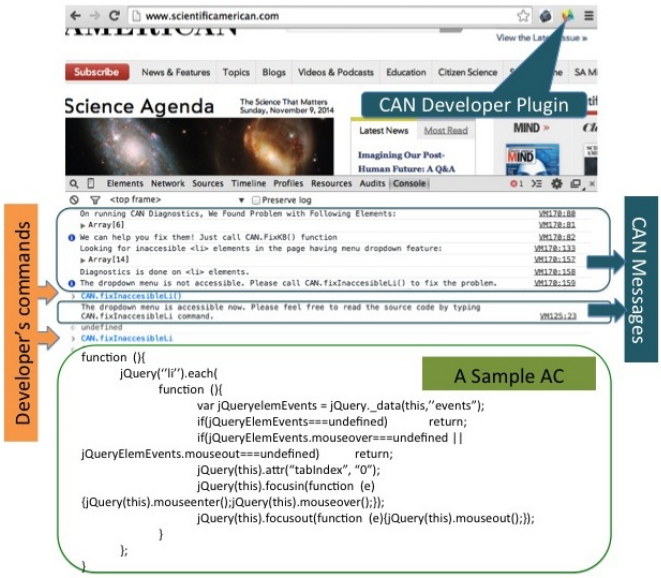

Figure 8: CAN Prompted Messages, developer's inputs, and one sample AC (with source code) are displayed in the developer console window.

5. DISCUSSION

This paper reports our initial development and study of CAN. There have been a lot of challenges working with real websites that can be updated frequently. First, different browsers (IE, Chrome and Firefox) may show different accessibility issues on the same websites, and different screen readers (*e.g.*, JAWS on Windows and VoiceOver on Mac)

may be able to reach and read different contexts. After web pages are updated, the previous fix might be broken in the new web page, even though the same problem may still remain. For example, similar to scientificamerican.com, walmart.com used to have the same implementation and dropdown menu issue. During our development and test period, an update to their website changed the implementation approach. The previous AC developed for the previous walmart.com did not work anymore. In the new walmart.com design, the main menu <li> has no event attached to it. The mouseover and mouseout event are not attached to the same element and moreover, each hidden submenu items have mouseover and mouseouts. So it makes it difficult to bind focusin and focusout events related elements to achieve the purpose. These all stress on the importance of building a comprehensive *Issue Repository* and an intelligent *Diagnostic Engine* that can fulfill the flexibility of the proposed CAN infrastructure. In this pilot system, the logic we implemented is quite intuitive. In future work, we plan to write optimization algorithms to detect if existing ACs can be improved to address more scenarios of the same accessibility issues to enrich the AC inventory.

Our experiences with CAN shows its promise as an educational tool for web developers in learning about accessibility issues and how to fix them. Interviews during our initial user study showed that many developers learn specific skills through self-directed methods. CAN's web developer plugin could be used in the classroom, or in self-directed learning. Importantly, CAN allows developers to both discover real accessibility issues and witness the changes after fixes are applied and explained. The experience serves to raise awareness and deliver specific educational objectives in an effective way. The initial user study utilized a small sample of web developers and featured one of the less complicated accessibility problems. Additional user studies focusing on other accessibility issues with a larger and more varied set of developers should be explored, while at the same time recognizing that many developers may have little accessibility awareness and training. The educational process should be optimized to accommodate for such developers, allowing an increased or customizable amount of explanatory feedback. Simply showing the code is not enough for all web developers to understand the solution. Further work should look at which available learning resources may be most useful to incorporate into CAN's process to meet the different learning styles and needs of web developers. In addition, further teaching and learning methodologies should be evaluated in an effort to improve developer interaction with CAN and increase learning outcomes.

There is a lot of existing literature on how to motivate online communities to contribute [29], how to engage people with disabilities to contribute in a crowdsourcing context [15, 38], and how to encourage open source developers to share code [42]. Designing a socio-technical system that encourages contribution will be vital for the sustained success of this project. One way to solicit ACs from opensource developers could be building a messaging system that enables effective and timely communication with a larger community of opensource developers. For example, if the *Diagnostic Engine* finds existing ACs do not fully address the same problems in certain websites, the CAN messager can send requests to discussion forums (e.g.stackoverflow.com) and seek solutions. CAN's data-driven collection of real-world examples and resulting educational opportunities may be unique features that can be leveraged for motivation.

6. CONCLUSION

CAN collects a variety of web accessibility issues on real websites and their software fixes, dynamically composes fix solutions on-the-fly and present this crowdsourced content as educational materials. In doing so, the proposed system provides a unique combination of crowdsourced services and infrastructure to address many of the challenges currently facing web accessibility. Recognizing the continuing prevalence of accessibility issues on even highly trafficked websites, CAN takes inspiration from successful crowdsourcing models in other domains and allows a more collaborative approach to the responsibility for an inclusive web. CAN also facilitates easy and meaningful contributions from end users, allowing them to directly contribute to the correction of accessibility issues without incurring any additional effort. Accessibility issues can be diagnosed, and crowdsourced fixes to common problems can be applied automatically, immediately benefiting the end users. More complex or unique problems can also be accommodated through CAN's design, allowing domain experts to receive and correct accessibility problems through open source contributions.

Though interest in accessibility as a topic web development education is increasing, lack of awareness and accessibility training on the part of web developers directly impacts website accessibility. Pre-existing accessibility issues can serve as useful educational materials for web developers though. CAN shows promise as a cost-efficient educational tool as well, using crowdsourced accessibility fixes to raise web developer awareness and understanding. An initial user study showed that web developers find CAN to be an effective and interesting way to learn about web accessibility.

CAN's crowdsourced approach leverages contributions from both end users and open source developers to provide real accessibility benefits. The current ACs were presented as examples across three levels of accessibility issues for other AC contributors. Ideally, CAN can create a positive workflow and improve web accessibility with both immediate and a long term impacts.

7. ACKNOWLEDGEMENTS

The work is funded under a grant from the United States Department of Education through the National Institute on Disability and Rehabilitation Research (H133A130057).

8. REFERENCES

[1] Algorithms Contrast Ratio or Contrast / Brightness Difference. http://gmazzocato.altervista.org/colorwheel/.

[2] CAN Developer Accessibility Tool. https://chrome.google.com/webstore/detail/developer-accessibility-t/fcngonjbamdeknokaghjgdgkbgcdahdl.

[3] CAN End User Tool - Enhance Web Page Accessibility. https://chrome.google.com/webstore/detail/enhance-web-page-accessib/mceoeijcofnglhhnanfinbemamafooma.

[4] Cloud4all. http://cloud4all.info.

[5] Colorblind population. http://www.color-blindness.com/2006/04/28/colorblind-population.

[6] European Blind Union - facts, figures and definitions concerning blindness and sight loss. http://www.euroblind.org/resources/information/.

[7] Global public inclusive infrastructure (GPII). http://gpii.net/.

[8] IDI web accessibility checker : Web accessibility checker. http://achecker.ca/checker/index.php.

[9] JAWS Screen Reader. http://www.freedomscientific.com/Products/Blindness/JAWS.

[10] Top Sites: The 500 most important websites on the internet - ranked by moz.com. http://moz.com/top500.

[11] Wave (web accessibility evaluation tool). http://wave.webaim.org/.

[12] A. S. Al-Khalifa and H. S. Al-Khalifa. An educational tool for generating inaccessible page examples based on WCAG 2.0 failures. W4A 2011.

[13] F. Alonso, J. L. Fuertes, A. L. González, and L. Martínez. On the testability of wcag 2.0 for beginners. W4A 2010.

[14] C. Benavidez, J. L. Fuertes, E. G. y Restrepo, and L. Martinez. Teaching web accessibility with "contramano"and hera. ICCHP 2006.

[15] J. P. Bigham, J. T. Brudvik, and B. Zhang. Accessibility by demonstration: Enabling end users to guide developers to web accessibility solutions. ASSETS 2010.

[16] J. P. Bigham and R. E. Ladner. Accessmonkey: a collaborative scripting framework for web users and developers. W4A 2007.

[17] J. P. Bigham, R. S. Kaminsky, R. E. Ladner, O. M. Danielsson, and G. L. Hempton. WebInSight: Making Web Images Accessible. ASSETS 2006.

[18] J. P. Bigham. Increasing Web Accessibility by Automatically Judging Alternative Text Quality. IUI 2007.

[19] E. Brady and J. P. Bigham. How Companies Engage Customers Around Accessibility on Social Media. ASSETS 2014.

[20] B. Caldwell, L. G. Reid, M. Cooper, and G. Vanderheiden. Web content accessibility guidelines (WCAG) 2.0. W3C recommendation, W3C, Dec. 2008. http://www.w3.org/TR/2008/REC-WCAG20-20081211/.

[21] J. A. Carter and D. W. Fourney. Techniques to assist in developing accessibility engineers. ASSETS 2007.

[22] R. F. Cohen, A. V. Fairley, D. Gerry, and G. R. Lima. Accessibility in introductory computer science. SIGCSE 2005.

[23] M. Cooper and J. Craig. Accessible rich internet applications (WAI-aria) 1.0. W3C recommendation, W3C, Mar. 2014. http://www.w3.org/TR/2014/REC-wai-aria-20140320/.

[24] B. Dosono, J. Hayes, H. Xia, and Y. Wang. "i'm stuck!": A contextual inquiry of people with disabilities in authentication. *Submitted to CHI 2014.*

[25] A. P. Freire, R. P. de Mattos Fortes, D. M. Barroso Paiva, and M. A. Santos Turine. Using screen readers to reinforce web accessibility education. ITiCSE 2007.

[26] E. Gellenbeck. Integrating accessibility into the computer science curriculum. *J. Comput. Sci. Coll.*, 21(1):267–273, Oct. 2005.

[27] V. L. Hanson and J. T. Richards. Progress on website accessibility? *ACM TWEB*, 7(1):2:1–2:30, Mar. 2013.

[28] S. Kawanaka, Y. Borodin, J. P. Bigham, D. Lunn, H. Takagi, and C. Asakawa. Accessibility commons: A metadata infrastructure for web accessibility. ASSETS 2008.

[29] R. E. Kraut, P. Resnick, S. Kiesler, Y. Ren, Y. Chen, M. Burke, N. Kittur, J. Riedl, and J. Konstan. *Building Successful Online Communities: Evidence-Based Social Design.* The MIT Press, 2012.

[30] K. Patch, K. Ford, J. F. Spellman, and J. Allan. User agent accessibility guidelines (UAAG) 2.0. Nov. 2013. http://www.w3.org/TR/2013/WD-UAAG20-20131107/.

[31] E. Pearson, C. Bailey, and S. Green. A tool to support the web accessibility evaluation process for novices. ITiCSE 2011.

[32] G. M. Poor, L. M. Leventhal, J. Barnes, D. R. Hutchings, P. Albee, and L. Campbell. No user left behind: Including accessibility in student projects and the impact on CS students' attitudes. *Trans. Comput. Educ.*, 12(2):5:1–5:22, Apr. 2012.

[33] G. V. S. Prasad, T. B. Dinesh, and V. Choppella. W4A 2014.

[34] C. Putnam, K. Wozniak, M. J. Zefeldt, J. Cheng, M. Caputo, and C. Duffield. How do professionals who create computing technologies consider accessibility? ASSETS 2012.

[35] D. Sato, M. Kobayashi, H. Takagi, and C. Asakawa. Social accessibility: the challenge of improving web accessibility through collaboration. W4A 2010.

[36] R. Somani, B. B. Deo, J. Xin, and Y. Huang. Building keyboard accessible drag and drop. ASSETS 2014.

[37] H. Takagi, S. Kawanaka, M. Kobayashi, T. Itoh, and C. Asakawa. Social accessibility: achieving accessibility through collaborative metadata authoring. ASSETS 2008.

[38] H. Takagi, S. Kawanaka, M. Kobayashi, D. Sato, and C. Asakawa. Collaborative web accessibility improvement: challenges and possibilities. ASSETS 2009.

[39] A. Waller, V. L. Hanson, and D. Sloan. Including accessibility within and beyond undergraduate computing courses. ASSETS 2009.

[40] Y. D. Wang. A holistic and pragmatic approach to teaching web accessibility in an undergraduate web design course. SIGITE 2012.

[41] S. Yan. SourceProbe: web accessibility remediation framework. W4A 2010.

[42] Y. Ye and K. Kishida. Toward an understanding of the motivation open source software developers. ICSE 2003.

[43] M.-C. Yuen, I. King, and K.-S. Leung. A survey of crowdsourcing systems. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Conference on Social Computing (SocialCom), 2011.*