Accessibility by Demonstration: Enabling End Users to Guide Developers to Web Accessibility Solutions

Jeffrey P. Bigham
ROC HCI Group
University of Rochester
Rochester, NY 14627
jbigham@cs.rochester.edu

Jeremy T. Brudvik
College of Computing
Georgia Tech
Atlanta, GA 30332
jbrudvik@gatech.edu

Bernie Zang
Computer Science
Penn State University
State College, PA 16801
nzz101@psu.edu

ABSTRACT

Few web developers have been explicitly trained to create accessible web pages, and are unlikely to recognize subtle accessibility and usability concerns that disabled people face. Evaluating web pages with assistive technology can reveal problems, but this software takes time to install and its complexity can be overwhelming. To address these problems, we introduce a new approach for accessibility evaluation called *Accessibility by Demonstration (ABD)*. ABD lets assistive technology users retroactively record accessibility problems at the time they experience them as human-readable macros and easily send those recordings and the software necessary to replay them to others. This paper describes an implementation of ABD as an extension to the WebAnywhere screen reader, and presents an evaluation with 15 web developers not experienced with accessibility showing that interacting with these recordings helped them understand and fix some subtle accessibility problems better than existing tools.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; K.4.2 [**Social Issues**]: Assistive technologies for persons with disabilities

General Terms

Design, Human Factors, Experimentation

Keywords

Web Accessibility, Web Usability, Evaluation, Blind Users

1. INTRODUCTION

We introduce *Accessibility by Demonstration (ABD)* as a general lightweight method of bringing the experience of access technology users to developers. In the ABD approach, people use their assistive technology as they normally would. If they encounter a problem, they tell their assistive technology to package a recording of their recent interactions that demonstrate the problem into a URL, which they can then send to the developer of the site using an appropriate medium (email, instant messaging, tweets, etc). This *retroactive recording* is possible because their assistive technology always keeps a buffer of recent actions so that it can package and send such a recording if the users requests it to do so. Importantly, the URL encapsulates not only a recording of the user's actions but also the assistive technology on which to playback the recording so that developers have immediate access to it. Few, if any, developers would view these recordings if they first had to install additional software or an extension to their browser.

Specifically, the steps of ABD are as follows (Figure 1):

- **User browses the web** — ABD is implemented on existing assistive technology web applications so that users do not need to learn new technology.
- **User encounters problem, retroactively records trace** — problems are retroactively recorded using a single shortcut.
- **User shares problem with developer** — a single URL encapsulates both the assistive technology and the recorded problem for easy sharing.
- **Developer experiences user's problem** — opening the URL from the user causes the assistive technology web application to be loaded, and the user's actions that led to the problem to be replayed.
- **Developer fixes problem** — developers use their existing tools to fix the problem and then reload the original ABD URL to verify the problem was fixed.

This paper explores ABD in the context of improving web access for blind computer users. Our implementation, WebAnywhere-ABD is built as an extension to the WebAnywhere [6], a web-based screen reader that, unlike conventional desktop screen readers, can be used on any computer without installation (even locked-down public terminals). Many developers are already using WebAnywhere to test the accessibility of their sites because it is free and does not require installation [1]. When a developer navigates to a WebAnywhere-ABD URL, WebAnywhere is first automatically loaded in the browser (since WebAnywhere is a web application) and then the recording captured by the user is played back. Developers neither need to be as skilled at using access technology nor do they have to be convinced to install new software.

ASSETS'10, October 25–27, 2010, Orlando, Florida, USA.

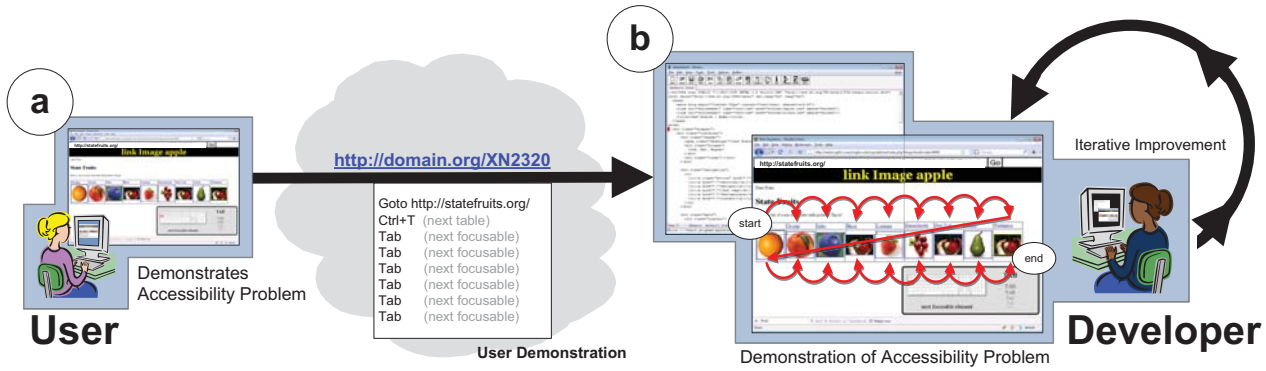

Figure 1: (a) Using the ABD approach, access technology users can capture problematic interactions and send recordings to developers. (b) Developers can replay these interactions to improve understanding of the problem without installing new software. Developers can replay these recordings after changing their content to test if their changes have helped.

The ultimate test of web accessibility is whether someone attempting to access web content can do so effectively. Accessibility has many dimensions, and people with disabilities vary in many ways. Usability is fundamentally tied to skill with appropriate access technology, and so even content that is accessible to someone experienced with a particular screen reader, such as JAWS [15] or Window-Eyes [31], might be inaccessible to someone who is new to the software. Creating accessible web content can therefore be a frustrating, subjective, and difficult task for developers, especially for the vast majority not specifically trained in accessibility.

Guidelines and standards serve as a valuable starting point, but are a high barrier to entry and may not capture many factors influencing the experienced usability of a web page [26]. Creating content possible to access is just one aspect of enabling users to effectively access content. While issues that clearly prevent access, such as images lacking alternative text, often receive the most attention, poorly laid out content or pages on which finding desired content may actually be less usable. Accessibility evaluation tools can highlight many problems, but often fail at conveying the effective accessibility for users. The ABD approach seeks to better connect the consumers of content with those who create it.

1.1 Example Use Cases

ABD can help experienced and inexperienced computer users demonstrate many problems that they face in accessing and using content and send those demonstrations to others. To illustrate how the ABD approach can be used in the context of WebAnywhere, consider the following scenarios:

First, consider Betty, an experienced blind computer user who has experienced an accessibility problem on a web site. The homepage of a local restaurant has arranged elements statically with CSS, resulting in a nonsensical reading order. Betty uses WebAnywhere-ABD to demonstrate her path through the web page, first pressing tab several times to show the response when attempting to jump from link to link, and then showing the unintuitive result of navigating by using the DOWN ARROW to go through the page element by element. Once she has finished, her recording is captured as a URL that she can send along with a comment to the email address listed on the page, for example "The reading order of this page does not match its visual appearance." She is able to demonstrate the problem without visiting the webmaster in person.

Upon receiving Betty's email, John, the geographically distant webmaster, simply clicks on the link that Betty sent to see a replay of her interactions with his site. This replay provides more than just a video to John — it is a live interaction with the site. John can interrupt the replay at any point and see how his site responds to keyboard shortcuts of his choosing. He can also update the site and see if that improves the replay, helping him iteratively improve the accessibility of his site and receive feedback during the process.

Next, consider James, a new screen reader user who hears what sounds like gibberish while browsing an online shopping site. James presses CTRL+R and WebAnywhere-ABD records the process he took to get to where he currently is on the page. The retroactive recording allows James to easily capture problems that he faces without spending any time trying to reproduce them and without having to remember how he got into the state where the problem occurred. James decides to tweet the URL and one of his sighted followers is able to tell him that his screen reader had started reading a complicated URL when he reached an image lacking alternative text. James neither had to understand what problem occurred or the technical reason for it — he only needed to know that he had reached a point that was inaccessible to him, and the system gave him a URL that captured the problem in an easy-to-share way.

1.2 Summary of Contributions

This work makes the following three contributions:

- We introduce the concept of Accessibility by Demonstration, and present an implementation for screen reader users that can be used on any computer without installing new software.
- We show how retroactive recording of trails can be used as part of the ABD approach to capture and share recordings of accessibility problems after they have already happened.
- We present an evaluation that shows that web developers are more successful at improving some accessibility problems when using ABD than without it, illustrating the promise of the approach.

2. RELATED WORK

The ABD approach seeks to make it easier for developers to leverage the expertise of disabled users to improve the accessibility of web sites. Related work falls into three categories: (i) accessibility assessment and evaluation, (ii) systems for improving web accessibility, and (iii) programming by demonstration and interactive help.

2.1 Accessibility Assessment

Accessibility validators help developers evaluate and improve their web pages, including Bobby [32], FAE [11], and WAVE [28]. Developers access their web sites with these tools, and receive a list of accessibility errors and warnings. Although a valuable first step, evaluators have two primary shortcomings. First, validators cannot detect accessibility issues that cannot be detected automatically. For instance, evaluation tools can detect if an image lacks alternative text, but cannot judge if that alternative text is appropriate and informative [9]. Subtle usability problems, such as problems with reading order or lack of heading tags or other markup, are even more difficult to detect automatically. These usability problems seem likely to get worse as web pages become more complex and begin to behave more like applications than static documents [20].

As a fallback, evaluation tools present warnings about content, and often present so many warnings that developers can be overwhelmed or be discouraged from fixing anything. While those who are motivated will investigate all warnings, typical developers may be unwilling to investigate each potential issue. As an example, WAVE displays a warning anytime a `tabindex` attribute is used within a web page. This attribute can be used correctly to enforce a meaningful tab order through a web page, but when used incorrectly may result in a confusing ordering. Because WAVE and other tools cannot automatically judge its correct usage, they present warnings for all uses of of `tabindex` regardless of whether there is actually a problem. Figure 2 shows the complexity that can result from all of these warnings. WebAnywhere-ABD enables users of a web site to demonstrate and share the most troubling accessibility issues for them, helping to focus the efforts of web developers and providing an opportunity to improve the understanding that developers have of why the issue is a problem.

Another approach to evaluation is to provide views of content that simulate what a disabled user might experience. For instance, ADesigner can visually simulate the usability of a web site as either a blind, low-vision, or color-blind person might experience it, display content in reading order and give an indication of the time required to reach particular elements on the page [23]. Although such tools help developers understand how certain groups might experience the web, they miss the personal nature of accessibility and require developers to understand the problems highlighted by these tools. WebAnywhere-ABD demonstrates a particular problem experienced by a blind web user, allowing developers to isolate the problem, iteratively improve it, and replay the recording until it is fixed.

Screening is the use of access technology to help identify accessibility issues. Henry describes the advantages and limitations of using screening techniques to evaluate accessibility [13]. The advantage is that it may help someone appreciate the experience of someone with different capabilities, but a disadvantage, especially when using a complicated software program like a screen reader, is that an inexperienced user may incorrectly associate their own inability to accomplish a task using the assistive technology software with the inaccessibility of their site. Although valuable, screening also suffers from the fact that screen readers are expensive and complex software programs that developers might be unlikely to install. In contrast to existing methods for screening, WebAnywhere-ABD provides the experience of using a screen reader without requiring developers to install new software and guides developers through a specific problem encountered during a blind web user's experience, instead of requiring developers to figure out how to use a screen reader alone.

Mankoff *et al.* compared the results of multiple sighted developers using screening techniques with evaluation by remote blind users [17]. Developers using screen readers found many more problems than did the blind evaluators, although the problems found by blind users were quite accurate. Commonly, when blind users perform an evaluation they focus on the most problematic accessibility problems and sometimes are not able to fully evaluate a web site because the accessibility problems discovered prevent full access to the site [18]. Takagi *et al.* describes the problems of users not knowing what is not accessible to them as a challenge for IBM's Social Accessibility project [25]. ABD seeks to provide the best of both evaluation techniques — disabled users can indicate problems that are problematic for them and send a clear demonstration to developers.

2.2 Involving Users

Target users should be involved in accessibility evaluation whenever possible. Although Mankoff *et al.* found that developers discovered the most accessibility problems (had high recall), the blind participants in their study found more subtle problems and were very precise [17]. Communication between developers and users has primarily been restricted to the descriptions that can be sent over email.

Several projects have explored how blind web users might independently improve the accessibility of the content that they access. For instance, by writing Accessmonkey scripts using provided end user tools, blind web users could improve the accessibility of web sites [4]. These scripts could be shared with developers and the changes made by the scripts saved to HTML with the goal of having the developer incorporate the changes into the original page.

AxsJAX [12] is a scripting framework that makes web pages into accessible web applications. Key functionality of each web page is exposed using shortcut keys and the semantics of the page are leveraged to make pages easier to use. Although programming is required to create an AxsJAX script, any programmer can write a script that other people can use. AxsJAX requires the user to already have a screen reader installed. ABD seeks to involve non-programmers in the process of improving web pages and provides a clear and easy path to demonstrating problems to developers.

The Social Accessibility project seeks to match blind web users experiencing accessibility problems with volunteers who can help them improve these problems [24]. In some sense, the blind users of Social Accessibility are demonstrating accessibility problems and sharing those problems with the volunteers. Currently, however, demonstrations are restricted to selections — users can select an image that lacks alternative text. They can also describe more complex problems

using free text — for example, "This page lacks heading tags." or "None of the form elements have labels." Using the WebAnywhere-ABD, users can record themselves performing richer tasks and send them to others who can view them without installing new software.

Users can also directly improve access and usability for themselves by demonstration, which we discuss next.

2.3 Web Automation

Programming by Demonstration (PBD) and web automation systems capture procedural knowledge and express it to users. COACH [22] and Eager [10] are early systems that work with standard desktop applications instead of the web. COACH observes computer users in order to provide targeted help, and Eager learned and executed repetitive tasks by observing users. Selenium [21] records and plays back scripts through the web and are used to automate testing for web sites. WebAnywhere-ABD adds the ability to easily share not only recordings but also the fully interactive interface used by the person who recorded the script.

Expressing procedural knowledge in order to assist a user who is currently working to complete a task is an important issue for interactive help systems. Stencil-based tutorials demonstrates a variety of useful mechanisms, such as blurring all items except for those which are relevant to the current task and adding useful contextual information within the application with sticky notes [16]. To help convey what a running demonstration is doing, ABD not only shows the result of the user actions that were recorded but also a visual display of the keyboard shortcut used and a description of the semantic action to which it corresponds (Figure 4).

TrailBlazer lets blind web users record and replay web tasks [5]. WebAnywhere-ABD makes trail recording and playback similar to those exposed by TrailBlazer easily available without installing new software, and supports a new use for it — as a communication path to developers. Although TrailBlazer scripts are shareable, a recipient of a TrailBlazer script needs to install TrailBlazer (and a screen reader) for playback. Although tools like TrailBlazer and Selenium are conceptually similar to ABD, the focus on sharing (including the sharing of the assistive technology) differentiates ABD as a tool to help users communicate the problems they encounter on the web.

3. ACCESSIBILITY BY DEMONSTRATION

The ABD system presented here is implemented as a component of the open source WebAnywhere web application [29]. It adds the ability to record and play back sequences of actions, enabling users to demonstrate the problems that they experience and then share those descriptions with others. WebAnywhere is attractive for use with ABD for the following two reasons: (i) it is an open platform that provides an API for accessing the actions that users perform using it, and (ii) as a web application that requires no software to run, it makes sharing demonstrations recorded using it straightforward.

The idea of recording and sharing user actions as parameterized URLs was inspired by the USA Track and Field's "Map it." On this web site, people can draw a running route on a map, save it to a centralized repository, and then share it with others as a parameterized URL [27]. Capturing all necessary information to replay a trace as a URL makes these recordings flexible and easy to share, and enables anyone with access to a web browser able to play back ABD recordings. Just as one's entire running route can be sent in an email or instant message, included on social networking sites like Twitter or Facebook, or even printed as part of paper correspondence; so can the trail formed by one's browsing interactions.

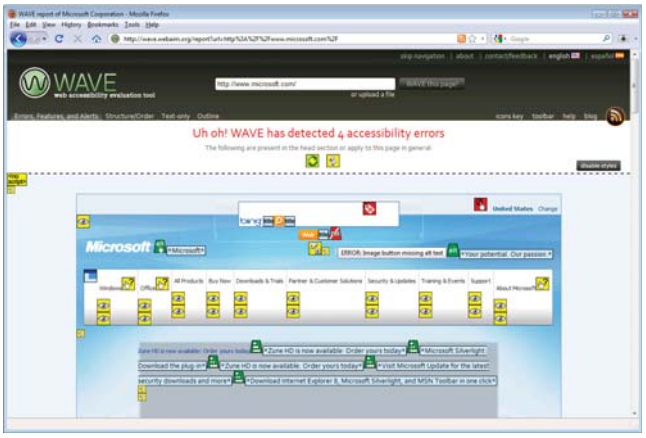

Figure 2: The WAVE accessibility evaluation tool's analysis of a web page showing numerous warnings, many of which are not actually problems.

3.1 WebAnywhere-ABD System

WebAnywhere-ABD is implemented as an extension to the open source web-based screen reader WebAnywhere [6]. WebAnywhere is written using JavaScript, which makes it compatible with any web browser on any platform[1]. WebAnywhere exposes an API that makes observing user interactions and playing them back relatively straightforward. The ABD extension captures each keypress that users make during recording and later simulates that keypress for the developer using the appropriate WebAnywhere API call.

3.1.1 Retroactive Recording

We designed WebAnywhere so that users do not need to start out intending to demonstrate an accessibility problem, but instead can capture the interactions that led to an accessibility problem retroactively. Retroactive recording builds on the idea of smart bookmarks [14], which were designed to let users bookmark dynamic web pages that may not have a static URL associated with them.

Retroactive recording enables ABD users to record an accessibility problem when they experience it by pressing a single shortcut key. Users simply browse the web as they usually would and press the "retroactive record" shortcut key when they encounter an accessibility problem. This may help novice users who may be unable to reproduce the steps necessary to get into a problematic state from the start, or experienced users who want to quickly record a problem and not bother with starting with a fresh demonstration. The goal is to make creating and sharing recordings of accessibility problems easy and efficient — retroactive recording requires users to press only one keyboard shortcut to record an entire problematic interaction.

To implement retroactive recording, WebAnywhere-ABD

[1]Try WebAnywhere at webanywhere.cs.washington.edu

#	Command	
1.	goto	http://www.nytimes.com/
2.	ctrl t	next table
3.	tab	next focusable element
4.	tab	next focusable element
5.	tab	next focusable element
6.	tab	next focusable element
7.	tab	next focusable element

Figure 3: An example ABD recording. This recording first visits the New York Times homepage, then skips to the first table, and finally tabs through the focusable elements five times. A short description accompanies each action to aid developers who may not be experienced with assistive technology or the specific shortcut keys used in WebAnywhere.

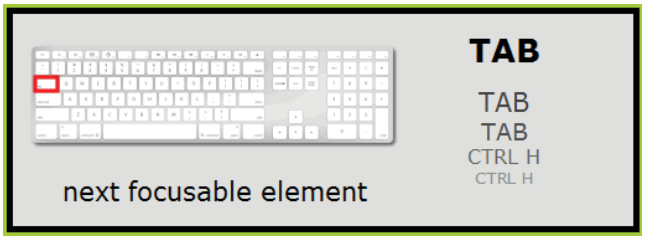

Figure 4: A graphical illustration of WebAnywhere-ABD's actions is provided by the ABD component to help inexperienced users understand what is happening, addressing one of the primary problems with screening: access technology can be too confusing for a novice to use as part of evaluation.

always keeps a list of the actions taken after new page loads. When a user chooses to make a retroactive recording, the ABD extension traverses backward as far as necessary to reach a known state on the page and saves all steps after that point as part of the recording. A known state is one that can be reached from a new browsing session (see [14] for details). ABD records both the keyboard shortcuts used and a direct address to elements (as an XPATH). Recording shortcut keys is critical for recording common accessibility problems. For example, the easiest way to demonstrate a lack of headings on a web page is to try to use CTRL+H to navigate between heading tags, which will immediately show that no headings exist — potentially a big problem!

Once a recording is made, WebAnywhere-ABD sends it to our server with an XmlHttpRequest and it is referenced using a parameterized URL. The URL first loads WebAnywhere and then plays back the recorded script. The effect is that developers can use screening techniques without installing new software or understanding how to use the complex screen reader interface, as will be explained next.

3.2 User Experience for Developers

Developers first use WebAnywhere-ABD by clicking a link shared by a user. This immediately opens the user's trace (see Figure 3) recorded while using the developer's web site and replays it in WebAnywhere. The content that is being read aurally — what a screen reader user hears — is accompanied by visual highlighting to aid the developer in following along. WebAnywhere-ABD also provides a view of the current and previous keyboard commands made by the user (see Figure 4). By displaying the keyboard shortcuts, the developer can not only follow the user's actions more easily but can also help the developer learn what is necessary to navigate through the page on their own without the recording — so that potential problems might be examined later.

As a specific example, consider this URL:

> http://webinsight.cs.washington.edu/exp/abd/wa/index.php?script=AXKJLS

Following this URL opens WebAnywhere and, based on the URL's parameter (script=A7XAK9), visits a sub-page at the University of Rochester and plays a recording of how headings are often navigated by screen reader users. The recording navigates through the page using the CTRL+H shortcut, which skips from heading to heading announcing each as they are visited. If no headings were available on this page, the recorded keyboard shortcuts would instead cause WebAnywhere to announce "no heading." The developer could then make iterative improvements to the web page and test them by reloading the URL until the problems are fixed. Because the ABD approach loads each page with a real browser in real assistive technology, it always loads the current version of a web page and reflects its current accessibility. Experienced developers can also go "off script" and navigate the page directly in WebAnywhere without being guided by the user's recorded actions.

4. EVALUATION

Our evaluation considered whether the addition of task descriptions and the ability to play them back would help web designers improve on the accessibility of their websites. We did not conduct a formal study with blind web users, but instead focused on the effect that WebAnywhere-ABD may have on developers because this is a prerequisite for the success of the ABD approach.

For this study, we focused on 4 common types of accessibility problems motivated by [7]:

- **Alternative Text** — Alternative text provides an accessible alternative for images on web pages. Designers include "alt" attributes as a part of the HTML image tags so that screen readers can read the text in place of the picture. While alternative text is very useful for these reasons, they are frequently forgotten, thus concealing content from blind users.

- **Heading Usage** — Heading tags (h1, h2, etc.) are generally used to provide visitors with a sense of structure and hierarchy. Specifically they signify the beginning of a new section and allow users to quickly find new sections. While for sighted users, these tags provide visual cues as the fonts size and weight vary, blind users can use them to quickly navigate through a page. However, particularly when websites are created using WYSIWYG editors, it may not be clear to the designer if these tags are being used. In some cases a designer will use tags to change the visual style without using a heading tag.

- **Reading Order** — Screen readers most commonly navigate through a page linearly according to the DOM

Figure 5: Pages exhibiting accessibility problems: (a) using tables for layout, which causes incorrect reading order, (b) improper use of heading tags, and (c) a combination of four accessibility problems.

order of the page. With the addition of advanced CSS techniques, how a page is rendered in a web browser may not reflect its DOM order. For instance, some WYSIWYG editors use absolute positioning to provide more flexibility. As a result, if a designer does not pay attention to the order they are inserting content, the rendered view can be vastly different than the reading order in the HTML source.

- **Tab Order** — Users often navigate pages by using the tab key. The "tabindex" attribute can be used to specify the order in which focusable elements are visited. This allows designers to create logical order on a website. Problems arise when using some visual tools for creating web pages, where tab order is often determined by the order in which elements are added as opposed to their textual placement or grouping.

4.1 Tasks

Keeping the 4 common accessibility issues in mind, we designed a set of 4 tasks, 3 of which cover specific issues and 1 that covers all 4 issues. These 4 tasks are described below.

- **Tables Task** — The page for this task contains a table with U.S. state names and corresponding official fruits (Figure 5-a). Each picture has adequate alternative text, but because the of the ordering created by the tables, the states are read first and then the fruits. This problem can be fixed in a number of ways, all of which involve either moving the text or the picture next to each other in the source code.

- **Headings Task** — This task contained a page with a clear outline of headings (Figure 5-b). Each heading looked correct visually, but because of the associated CSS style sheets, certain headings were actually styled using tags instead of appropriate heading tags. To fix this, participants needed to locate all of the tags and change them to an appropriate heading tag.

- **Tab Order Task** — The page for this task included elements that were positioned using absolute positioning that resulted in their visual order not reflecting the order in the HTML source. To correct this, participants needed to rearrange the sections of the page to match a natural reading order. Some elements were also assigned `tabindex` values that overrode the default tab order in a way that differed from the visual semantics. If the ordering of the source is corrected, this can be solved by removing the `tabindex` values completely. Otherwise, the `tabindex` values needed to be change to reflect the visual flow of the page.

- **Combination Task** — Accessibility issues rarely exist on their own. The combination task combines all 4 of the accessibility problems outlined earlier (Figure 5-c). All sections of this page were positioned using absolute positioning and randomly arranged in the source code. Many elements on the page had `tabindex` values, also randomly distributed. Images were randomly assigned appropriate or missing alternative text, and headings were randomly assigned a mix of correctly used heading tags and visually-identical tags.

For each of the 4 tasks, there were 3 conditions: WebAnywhere-ABD, WAVE, and WCAG. In the WebAnywhere-ABD condition participants viewed an interactive recording of someone trying to access the web page with WebAnywhere. The recording was created manually based on common strategies of screen reader users [7]. In the WAVE extension, participants used the WAVE accessibility evaluation tool [28], and in the WCAG condition participants were given access to the WCAG guidelines [30]. For all conditions, participants were given a sentence describing the problem from a prospective of a non-technical user. For example, "I have difficulty navigating the page quickly" was provided for the Headings Task.

4.2 Procedure

We recruited a total of 15 participants who completed 45 total tasks and who were paid $2.50 per task. The participants consisted of 11 men and 4 women between the ages of 20 to 38 (Mean=28.6; SD=5.68). When asked to rate their own web design skills (1=not skilled and 7=very skilled) the all self-rated very highly (Mean=5.07; SD=0.73). Their self-ratings of accessibility experience (1=not experienced and 7=very experienced) were much lower (Mean=3.93; SD=1.98). The participants were therefore self-rated skilled web designers but were not that experienced with accessibility issues.

Participants were shown a page that described the problem on the page, presented with one of the 3 conditions outlined in the previous section, and asked to evaluate the page using the tool provided. Participants were asked to make

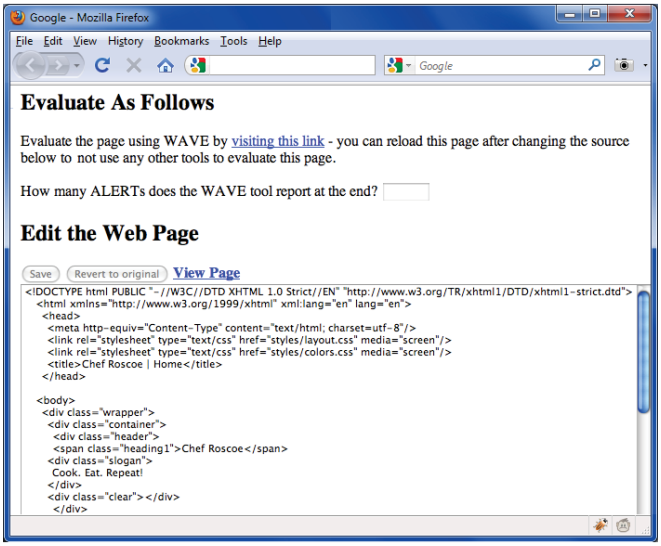

Figure 6: The web-based editor used by participants in the study.

appropriate improvements to the page using a web-based text editor that contained the web page source code (Figure 6). The participant could save the source of the page and view the rendered page in another window. Finally, participants were asked for a written description of the problems that they observed.

4.3 Evaluation Results

We began our analysis by tallying the number of problems fixed by each participant for each task, although we found that if a participant as able to correct one problem they were very likely to find them all. For example, if a participant noticed that the headings were not correctly applied, then they mostly found all such problems and corrected them. As a result, we normalized these results and tallied the problem types for each condition. This is presented in Figure 7.

Overall, developers correctly recognized and fixed 41.9% of problems using WebAnywhere-ABD as compared to only 25.0% using WCAG alone, which was a significant difference ($t(24)=2.32$ at $p<.05$). Participants in the WAVE condition fixed 44.1% of problems, a difference from WebAnywhere-ABD that was not statistically significant.

We did find, however, that WebAnywhere-ABD offered a significant difference over WAVE on the reading order task. While participants in the WebAnywhere-ABD condition fixed 50.0% of problems in the reading order task, WAVE users fixed only 36.4% of problems ($t(6)=2.32$ at $p<.05$). The results differences between WebAnywhere-ABD and WAVE were not detectably significant on the other tasks.

Upon analyzing the comments for each task, we found that for many cases the participants who did not have any evaluation tools at their disposal depended on guessing. One participant even described that her solution was a "total guess".

5. DISCUSSION & FUTURE WORK

Our study demonstrated the potential of using the ABD approach as part of a comprehensive accessibility evaluation. In particular, our results suggest that the ABD approach

	WA-ABD	WAVE	WCAG
Reading Order	0.50	0.36	0.14
Headings	0.33	0.42	0.00
Alt Text	0.50	0.75	0.75
Tab Order	0.33	0.50	0.33
Total	0.42	0.44	0.25

Figure 7: Fraction of problems fixed by participants per task and condition.

can help developers find what might best be classified as usability problems that present themselves when users employ certain assistive technology. In contrast to earlier tools, WebAnywhere-ABD requires little overhead for the developer, yet still allows disabled users to be involved in the accessibility evaluation process.

We plan to explore the limits of accessibility evaluation using this approach by publicly releasing the tool and studying how people use it "in the wild." Questions for future work include (i) How well can disabled users demonstrate different types of accessibility problems? (ii) How well can developers understand and fix the problems contained within the playbacks of different problem? and (iii) How can the playbacks be best augmented with interactive help to best lead to accessibility improvements?

We might also explore exposing common browsing patterns so that web developers can leverage WebAnywhere without requiring a disabled user to demonstrate problems. For instance, a standard pattern could be browsing through a page by headings, another by links, and another by tabbing through content. Eventually, we could extend WebAnywhere to showcase how other populations experience the web, such as making web content look as it would to people with low vision or color blindness. Such modifications have been created before as separate tools [23], but have required developers to both know about and install new software. WebAnywhere-ABD could help developers easily understand the experiences of diverse users.

WebAnywhere-ABD is currently limited in the types of accessibility problems that can be demonstrated to it. Some of the limitations of WebAnywhere-ABD are inherited from the underlying WebAnywhere platform. For example, WebAnywhere does not currently implement ARIA [2], so problems related to dynamic content are currently impossible to demonstrate. This limitation will improve as WebAnywhere catches up with or improves upon the other access technology that is available.

WebAnywhere currently does not allow its users to browse local files or files located behind a firewall. This means that developers who want to try the changes to their content would need to put it on a publicly-facing site. Sites that require users to login may also pose problems, although it seems reasonable to expect the site owner to have a login to their own site. In this case, content specific to a user's account might not be able to be evaluated using WebAnywhere-ABD. These problems are not fundamental limitations of WebAnywhere-ABD, but rather limitations of our current implementation.

6. CONCLUSION

We have presented and motivated a new approach to accessibility evaluation called *Accessibility by Demonstration* (ABD) that enables users to demonstrate accessibility problems and send recordings to developers who can play them back without installing new software. In an evaluation with developers who were not experienced with accessibility evaluation, participants created more accessible content when the problem was demonstrated to them using WebAnywhere-ABD. Based on these results, ABD could be an important complement to existing evaluation techniques. WebAnywhere-ABD is a relatively low-cost approach to involving users, one that may be likely to be used because it does not require new software to be installed. ABD also demonstrates the need for multi-faceted evaluation; combining multiple diverse evaluation techniques will likely yield the best results.

7. REFERENCES

[1] P. Abrahams. *Testing websites with the WebAnywhere Screen-Reader* http://www.bloorresearch.com/blog/, 2009.

[2] ARIA. http://www.w3.org/TR/wai-aria-roadmap/.

[3] J. P. Bigham. *Intelligent Interfaces Enabling Blind Web Users to Build Accessibility Into the Web*. PhD thesis, University of Washington, 2009.

[4] J. P. Bigham and R. Ladner. Accessmonkey: A collaborative scripting framework for web users and developers. In *Proc. of the Intl. Cross-Disciplinary Conf. on Web Accessibility (W4A '07)*, pp. 25–34, 2007.

[5] J. P. Bigham, T. Lau, and J. Nichols. Trailblazer: Enabling blind users to blaze trails through the web. In *Proc. of the 12th Intl. Conf. on Intelligent User Interfaces (IUI 2009)*, pp. 177–186, 2009.

[6] J. P. Bigham, C. M. Prince, and R. E. Ladner. Webanywhere: A screen reader on-the-go. In *Proc. of the Intl. Cross-Disciplinary Conf. on Web Accessibility (W4A 2008)*, pp. 73–82, 2008.

[7] Y. Borodin, J. P. Bigham, Glenn Dausch, and I. V. Ramakrishnan. More than Meets the Eye: A Survey of Screen-Reader Browsing Strategies. In *Proc. of the Intl. Cross-Disciplinary Conf. on Web Accessibility (W4A 2010)*, 2010.

[8] J. T. Brudvik, J. P Bigham, A. C. Cavender, and R. E. Ladner. Hunting for headings: sighted labeling vs. automatic classification of headings. In *Proc. of the Intl. Conf. on Comp. and Accessibility (ASSETS 2008)*, pp. 201-208, 2008.

[9] T. C. Craven. Some features of alt text associated with images in web pp.. *Information Research*, 11, 2006.

[10] A. Cypher. Eager: programming repetitive tasks by example. In *Proc. of the SIGCHI Conf. on Human factors in Comp. Sys. (CHI '91)*, pp. 33–39, 1991.

[11] Firefox accessibility extension, 2006. Illinois Center for Information Technology.

[12] Google-AxsJAX. Accessed April 15, 2009. http://code.google.com/p/google-axsjax/.

[13] S. Henry. *Just Ask: Integrating Accessibility Throughout Design*. Lulu.com, London, United Kingdom, 21 February 2007.

[14] D. Hupp and R. C. Miller. Smart bookmarks: automatic retroactive macro recording on the web. In *Proc. of the 20th annual ACM Symposium on User Interface Software and Technology (UIST '07)*, pp. 81–90, 2007.

[15] Jaws 8.0 for windows. Freedom Scientific, 4 May 2009. http://www.freedomscientific.com.

[16] C. Kelleher and R. Pausch. Stencils-based tutorials: design and evaluation. In *Proc. of the SIGCHI Conf. on Human factors in Comp. Sys. (CHI '05)*, pp. 541–550, 2005.

[17] J. Mankoff, H. Fait, and T. Tran. Is your web page accessible?: a comparative study of methods for assessing web page accessibility for the blind. In *Proc. of the SIGCHI Conf. on Human factors in Comp. Sys. (CHI '05)*, pp. 41–50, 2005.

[18] H. Petrie, F. Hamilton, N. King, and P. Pavan. Remote usability evaluations with disabled people. In *Proc. of the SIGCHI Conf. on Human Factors in Comp. Sys. (CHI '06)*, pp. 1133–1141, 2006.

[19] H. Petrie and O. Kheir. The relationship between accessibility and usability of websites. In *Proc. of the SIGCHI Conf. on Human factors in Comp. Sys. (CHI '07)*, pp. 397–406, 2007.

[20] Roadmap for accessible rich internet applications (wai-aria roadmap). World Wide Web Consortium, 2007. http://www.w3.org/TR/WCAG20/.

[21] Selenium, 2009. http://seleniumhq.org/.

[22] T. Selker. Cognitive adaptive computer help (coach). In *Proc. of the Intl. Conf. on Artificial Intelligence*, pp. 25–34, IOS, Amsterdam, 1989.

[23] H. Takagi, C. Asakawa, K. Fukuda, and J. Maeda. Accessibility designer: visualizing usability for the blind. *SIGACCESS Accessibility and Comp.*, (77-78):177–184, 2004.

[24] H. Takagi, S. Kawanaka, M. Kobayashi, T. Itoh, and C. Asakawa. Social accessibility: achieving accessibility through collaborative metadata authoring. In *Proc. of the 10th Intl. ACM SIGACCESS Conf. on Comp. and accessibility (ASSETS 2008)*, pp. 193–200, 2008.

[25] H. Takagi, S. Kawanaka, M. Kobayashi, D. Sato, and C. Asakawa. Collaborative web accessibility improvement: Challenges and possibilities. In *Proc. of the 11th Intl. ACM SIGACCESS Conf. on Comp. and accessibility (ASSETS 2009)*, 2009.

[26] S. Trewin, B. Cragun, C. Swart, J. Brezi and J. Richards. Accessibility Challenges and Tool Features: An IBM Web Developer Perspective. In *Proc. of the Intl. Web4All Conf. (W4A 2010)*, 2010.

[27] Usatf - america's running routes - map it. Accessed May 1, 2009. http://www.usatf.org/routes/map/.

[28] Wave web accessibility evaluation tool, 2009. http://wave.webaim.org/.

[29] Webanywhere open source project, 2009. http://webanywhere.googlecode.com/.

[30] Web Content Accessibility Guidelines. World Wide Web Consortium. http://www.w3.org/TR/WCAG20/, 2009.

[31] Window-eyes. GW Micro, 3 April 2009. http://www.gwmicro.com/Window-Eyes/.

[32] Watchfire bobby. http://www.watchfire.com/products/webxm/bobby.aspx.