Webinar: 6988 ‑ 11. BLH Training: Automated Testing Approaches with Real Time Demonstrations
Date: 2/8/22
This is a Captioned transcript provided by CIDI to facilitate communication accessibility and is not a verbatim record of the session.

>> Johan Rempel: Good afternoon, everyone. This is Johan Rempel from CIDI at Georgia Tech. And today the presentation will cover the topic of automated testing approaches with real time demonstrations.

We have the privilege of having Rayianna Daniels and John Toles present on this topic. This is a topic they are very familiar with. They are deep in the trenches on a regular basis regarding accessibility testing, the automated testing tools certainly aren't everything. They do a lot of code inspection and as a team we look at the accessibility usability experience from many user perspectives with assistive technology as well. This is an important component for testing accessibility.

Next slide.

So, once again thank you to Heather for the live captions today. There's two ways to access it. She's already placed the StreamText link in the chat or you can also click on the closed captioning option on the toolbar of Zoom as well.

Next slide.

So, most of you have already heard about who we are at center for inclusive design and innovation. We provide a number of different services. All really related to disability awareness, assistive technology, and accessibility initiatives, research initiatives, braille, captioning and audio description, accessible document creation and remediation, and assistive technology solutions, as well as digital accessibility training and evaluations.

Next slide.

So the overview today is to ‑‑ a description of the tools that will be used, tool comparisons, the pros and cons of each and the testing methodologies and supplemental resources and then a live demonstration of these tools as well. With that I will pass it on to Rayianna and she will cover the goals for today's presentation as well.

>> Rayianna Daniels: All right. Thank you, Johan. So the goals for today's presentation number 1 is to provide an overview of free automated testing tools. Give you a general overview. Not too in‑depth. We're going to talk about the pros and cons of each one. And later John Toles will give real world examples of accessibility and then at the end we'll share resources to assist writers, designers, and developers to better understand and create online content.

So I am Rayianna Daniels. I'm a digital accessibility specialist here at CIDI. I also work in our customer support department as well, as well as ICT and this presentation is brought to you by ICT. John Toles is also a digital accessibility specialist and works in customer support and ICT and a member of our SAM team.

So, getting into the first tool is ANDI. Before I begin, I'm sorry, the order that I'm going to present the tools in today ‑‑ at the beginning is just alphabetical. That's the easiest way. Later on I will talk about our recommended test flow which are then I will order the tools to replicate how they may be used when evaluating a website or document.

First up is ANDI. ANDI stands for accessible name and description inspector. And that's exactly what it is. It was created by the social security administration. It sits on top of the page on your browser. It can be saved as a bookmarklet or favorite. It doesn't require you to download anything. You just drag it to your bookmark bar.

So, ANDI has the ability to detect issues within a webpage like insufficient color contrast and missing alt text. In the screen shot here on the right side of the slide there's a pinkish red boxes around certain elements on the page. It's pulling out the images that it sees on the page. So although those features are great like the highlighting and detecting issues, the best part of ANDI is its ability to anticipate what a screen reader would read or what it may say when it interacts with an element. Whenever an element like a link or button receives focus, the ANDI output will produce a text version of what a screen reader will say.

For example, if focus is shifted to an image with alt text, it will read the alt text. Just like I have here on this screen shot. I believe the logo there doesn't have alt text. So the ANDI output reads image has no accessible name, alt or title.

Because of its ability to anticipate screen reader output, the tool is especially useful for developers who are new to accessibility or new to using screen readers or who want to test out their websites.

Along with the output feature, ANDI also the ability to provide alerts of potential issues on the page. Over all, it's a really good tool to double back in an evaluation or just as a general spot checker when you're developing something. It ensures commonly overlooked elements are addressed. Like links makes sure they have labels and what not.

So aXe by Deque. AXe operates as an accessibility toolkit and is built on the accessibility rules library that they created themselves. It's built on WCAG success criteria. Because of this, when you use aXe, it accurately ties each automatically discovered violation to a WCAG success criteria. ANDI operates to a bookmarklet. AXe is actually an extension that you install within your browser. Whether you're using Chrome or Firefox. It can be accessed from your extensions tray in the upper‑right hand corner of your browser or using the inspector tool in your browser. You have a couple options to get to it.

It's worth noting that this toolkit is supported on Android operating systems as well. I haven't had a chance to use it on an Android operating system but that's something they talk about on their website.

So, aXe has the ability to highlight areas of the active page but its highlighting ability is limited to the areas it finds issues. So, where ANDI was able to pick out certain elements on a page, aXe will only highlight things that has a potential issue.

Another great feature of aXe is its ability to detect those accessibility issues and provide supplemental information for the violation.

As I stated before, the accessibility rules library of the toolkit is based on WCAG guidelines. Because of this, each violation links to a page that explains the corresponding success criteria in plain language. So no matter your level of expertise, you'll be able to understand. And it provides a suggestion on how to remediate the issue and links to other articles about the issue.

Another feature is that it also indicates which issues that have been detected might need a manual review from a tester or developer. This usually includes issues with color contrast. Those are typically the most common ones that need manual review. A computer can only do so much in determining color contrast. Especially if it's like text that's been positioned on the page using CSS it doesn't know the true background of the text. So it can't tell you the exact color contrast ratio.

Moving on, with any automated tool you can find that there are going to be some false positives, just because try as they might, developers aren't able to account for the many different ways a website can be designed. Deque have a low percentage of false positives. I can kind of attest to this because I can probably account on one hand the number of times, I have been giving a false positive.

Before I move into bookmarklets, am I going too fast or am I going too slow? Let me know in the chat. Am I coming through clearly?

>> Johan Rempel: You're doing great, someone says. You're very well paced, Rayianna.

>> Rayianna Daniels: Gotcha.

>> Johan Rempel: Catherine says you're clear to me.

>> Rayianna Daniels: Awesome. Thank you. On to bookmarklets. When evaluating websites and web applications I use a compiled list of bookmarklets created by John J. Adams.

Each bookmarklet uses Java script to highlight the roles, states and properties on a webpage. If you want to highlight all of the forms or images or tables, et cetera.

Here I have an example of two bookmarklets that I probably use the most: The forms bookmarklet ‑‑ there's an image here that says apply now. It's a form. The fields of the form are highlighted. It will tell you more information about the form fields. They have an input field but no IDs. Another bookmarklet I use a lot is the list bookmarklet. There are often things that look like lists. Before I dive into the code, I want to know whether or not it is a list. Something good for spot checking ‑‑ checking out things at a higher level.

The bookmarklets can be used to quickly spot-check for any markup on the page and they have flexibility in their options. With aXe and ANDI, those tools once you install them or add them to your bookmarks bar you get all the tools that come with them. With bookmarklets, each one is its own individual tool. So you can save as many as you want. You can interchange. If you want all 12, you can save all 12. I only have a few that I use. I save just those to my bookmarks bar and come back to them. The color contrast analyser, when testing with automated tools color contrast violations are one of the most common issues found. Some of the violations are actually false positives like I mentioned before. With the color contrast analyser, I can actually go in and determine the color ratio between two elements which will allow me to determine whether or not there's an issue or just a false positive that was flagged by another tool.

And this can be done by plugs in the color code of the text. You can see that on the screenshot here.

In the analyzer, hex code is the default code type but it accepts pretty much everything from hex to hsba. We most often use hex because that's most typically used in the code that we evaluate. If you prefer or don't have access to color codes ‑‑ for example if you're checking out the color contrast in a PDF ‑‑ you don't know the hex code off the top of your head, you can use the eye dropper tool to extract the color and compare them in the color contrast analyser. Depending on how small or large the element is and whether or not I have access to the color code I will use the eye dropper tool or go back and forth between using the tool.

The main functionality is to analyze the color contrast. It also has the ability to determine ‑‑ or to figure out whether or not the color contrast ratio passes WCAG guidelines. When a comparison is made, the ratio is evaluated against the WCAG AA and AAA success criteria. It will say it passes for large and regular text or for large text but not regular text or vice versa.

So we are able to detect what we need to change.

So, the color contrast analyser, like I said, it analyzing color contrast but it has a color blindness simulator which is really, really cool. When you compare two colors using the tool it will automatically analyze how that color can be seen through the lens of various different kinds of color blindness. I think this makes a great addition to designers and developers. You will be able to make decisions out of the way early in the design of your application or website and you don't have to worry about going back to fix them later.

On a final note about color contrast analyser, unlike the tools that I mentioned before that live within your browser, color contrast analyser is a local application. You have to install that on your local machine which is great especially if you are doing work and you don't have Wi‑Fi or internet connectivity. You don't have to rely on that to use this tool.

Of all of the tools that I mentioned thus far, HTML Code Sniffer is the one I use the most. It has the ability to check accessibility violations. You're given a list of standards to choose from. Whenever you run Code Sniffer on the page, you're presented with 3 different levels of violation detection which are errors, warnings and notices.

Errors are just clear violations of whichever stand you have chosen. For example, neglecting to characterize [indistinct speech]. Warnings may not necessarily be an issue. Things that fall into this category should be reviewed to ensure the markup is intentional. For example, you have empty alt attributes for an image, those will filter in this category. Images with empty alt attributes will be ignored. So the Code Sniffer wants you to know this image has an empty alt attribute.

And finally you have notices. These are things to be aware of and keep an eye out for. For example, if there are links on your page you might receive a notice saying make sure the links have purposeful texts. It's not necessarily finding an issue with the links it's just letting you know to check on this and make sure it's good to go.

The one thing about notices, is that you'll often have a lot of them. That's because every single link will have the same notice as an individual notice. So that's why it typically has numbers in the 3 digits.

For each violation that Code Sniffer finds, you're given a brief explanation of the issue and the related success criteria and technique or if you have Section 508 chosen, you'll be given the standard and then ‑‑ I'm sorry, you'll be given the related standard. It will link you to where you can find more information about that standard.

All right, so, the functionality of Code Sniffer is mostly coded in JavaScript. It's open source. You can go in and clone it and modify it to create other detection points outside of the standards that are already there. If you're proficient in JavaScript and you would like to use the code to evaluate other standards you can do that. It's flexible. Not only does it indicate errors, it makes sure where warnings and notices ‑‑ one thing I neglected to highlight is when there are warnings ‑‑ I'm sorry. Warnings and notices ‑‑ it will have you go back and check over your code and your work.

Finally, before I get off of this, on the screen shot here is a little bit of what I was explaining before. This is a brief description of the issue it found. Because I have WCAG AA selected it will give me the success criteria and link to that and link to suggested techniques on how to fix the issue. It doesn't necessarily provide a recommendation on how to fix it but it will tell you what's wrong and resources on the success criteria and the issue. And it will provide a code snippet as well.

So Nu Validator. Nu Validator is essential an HTML validator like Code Sniffer. It validates code and provides the user with a list all the errors found. You have the option to link to your code if the site is live or you can copy and paste the code or up load a source code file. The easiest method if you have a live site is link to it. The errors that are found in the validator don't always pertain to accessibility because it's not built for that. It's not built to focus on accessibility. So for that reason, whenever I use Nu Validator I use a filtering script that was created by the Paciello group. Those are the developers of the color contrast analyser. They have a script that you can use with the Nu Validator to parse out any issues unrelated to accessibility.

Unlike any of the other tools that I presented; Nu Validator is just a website. It doesn't live within your browser unless you save a link to the site.

You have to actually go there and it has to run another tab and you have to insert your code in some way and it will give you every issue it found related to parsing and issues within your code.

All right. So, WAVE by WebAIM can be defined as a suite of tools. It was created to help content authors and web developers make their content more accessible. It helps them do this by identifying accessibility and WCAG errors. So as you can see here, it overlays the tab in which it is activated on the left side we have the WAVE tool. And on the right side you see the website being tested.

Unlike the other tools, it does highlight but it uses badges to identify accessibility features like ARIA and page elements like headings and lists. And it uses badges to identify accessibility errors like missing alt text or insufficient color contrast. Things like that.

So, the badges if you can make them out on this screenshot here, all of those are interactive. Whenever they're activated a tool tip dialogue box will appear to provide information about the specific badge. On this website the accessibility university website here is flags the logo on the top. There's a red badge next to the logo. That's for an image. It will tell you this is an image and it's more than likely missing alt text. And then it will also link you to the source code to show you where in the code this is an issue and where you should focus on when you're attempting to remediate it.

No matter the type of badge you're given, whether for an error, an alert which are typically yellow, or a feature ‑‑ those are green or purple ‑‑ you will always be given two options. You'll be given the option to view the reference which is a fuller explanation of the issue or the feature or the alert or the ARIA feature or you'll be given the option to view the code as I mentioned before.

So no matter what type of badge it is, you'll be given those options.

If you can't already tell, WAVE is kind of busy with what it puts on a page and this is pretty light compared to what I've seen with the number of badges that are here. It allows you to sort and turn off what you don't want. If you only want to look for errors, you can uncheck all of the badges that you don't want to see and have the errors display. That's how I use it. I know John says he uses it to look for ARIA structure. If you want you can have all the badges on or turn on only the ones you want to see.

Another cool tool ‑‑ feature of this tool is the ability to desaturate the page. On WAVE on the screenshot that I have here under the summary there's a few different things. There's a summary tab, a reference tab, a structure tab, a contrast tab. With that on the contrast tab you can desaturate the page and turn everything back and white and that will let you know if you're using color to convey information. That's something tool to check out.

So, as I mentioned before, it will provide you a lot of information about the errors that it finds. It will provide you with recommendations. As I said, it gets pretty busy on the webpage. Because of all the things it finds, if it finds ‑‑ if it has over 100 badges on the page it tends to slow down. That's a down side to it. Which is typically why I like to use it on small webpages. If I do use it on large webpages, I turn off the badges that I don't want to see.

All right, so on to the Web Developer Extension. The Web Developer Extension has over 100 tools within it. It's essentially an extension that can you install in your browser. Within it has stools that range from highlighting content on the page, outlining content or double check whether you have alt text on images, the states of certain things. It's a pretty much all in one type of tool. Because it has over 100 tools, it's something that I use the most. Along with Code Sniffer. It helps me identify certain elements on a page. If I'm looking for heading structure, this is helpful. What it does not do, however, is it doesn't necessarily provide information about violations to WCAG or any accessibility standard. It will just help you identify whether an issue may potentially be but it won't give you hey this is an issue because of XYZ.

One thing to note about this because it has so many tools is it's not customizable at all. Whenever you install it, you will get all of the tools. There are a couple you can probably turn off but only one or two of those.

Here's examples of some of the tools. For forms, looking at images, outlining tables, frames. It's cool at highlighting elements on a page.

So getting into some comparisons ‑‑ Nu Validator verses HTML Code Sniffer, they're tools that kind of perform similar functions but also very different at the same time. Nu Validator checks and validates HTML code but doesn't necessarily focus on accessibility. HTML Code Sniffer, however, was built to check for accessibility violations. That's at its core. That's what it does.

Whether Nu Validator finds parsing issues using the filtering script, HTML Code Sniffer is more well-rounded and find a wide variety of violations and breaks the violations into three categories. Another difference is how they're accessed. Nu Validator requires you to open up the website in a new tab. HTML Code Sniffer lives within a tab that you activate it in.

So it doesn't require you to jump back and forth between different tabs when you're evaluating which makes it go by faster.

The Web Developer Extension and bookmarklets are very similar. They highlight and identify elements within a page. One key difference with the Web Developer Extension you're getting all of the tools and you don't have an option to pick and choose what you want which can be good or bad. With the bookmarklets you're limited to just about 12 tools but you have options to pick and choose, especially if you're more comfortable with one over the other or don't want to clutter your bookmarks bar.

So then finally ANDI, aXe and WAVE. All of these operate in a similar way within the browser, and they're all able to run in the same browser that the page is being testing. ANDI and aXe use boxes to highlight elements. WAVE uses badges. As you'll see in the live demonstration sometimes this causes the page to become very cluttered and busy. With ANDI and aXe, the use of boxes to highlight keeps the pages a little clutter free.

Another difference between ANDI and aXe and WAVE is where they are located. ANDI is located in the bookmark bar. AXe can be launched from either in your extension tray or from your inspector tools. Wave is an extension. That can be launched from your extension tray as well.

All of these tools have the ability to detect violations, but ANDI is set apart because it does this but it also anticipates what a screen reader may say. So that kind of makes it a little more unique.

So ‑‑

>> Johan Rempel: You're doing great on time. I think we're right on target. I want to let you know it's 3:35.

>> Rayianna Daniels: Awesome. I'm finishing up right here, I think. I think this is the last one. So, here's our recommended testing workflow. On the left we have tools that are used frequently. The bookmarklets are used to high light elements. From there HTML Code Sniffer helps you discover issues, to document during your evaluation. The color contrast analyser lets you verify color contrast ratio. I frequently use this in conjunction and Code Sniffer. If you're going in the order of this list here you can then use aXe to find an issue that might have been missed by Code Sniffer and to round it out, I use Nu Validator to find any miss parsing errors. On the right we have tools that are dependent upon the type of website you're evaluating or the type of content. As I mentioned, WAVE ‑‑ if the site is really busy, WAVE will lag and slow down a bit. It will cause your browser to do that. If you're evaluating a basic site, WAVE might be a great option. The Web Developer Extension has so many tools, many of which are replicated by some of the tools I mentioned before it's only used for something I need that's unique. For example there's a form fields tool which is cool.

And here's some supplemental resources that we gathered. First is the WCAG checklist which we refer to this during an evaluation to make sure we're hitting every success criteria. Making sure where designation fall and did, we check everything and make sure we cover all of our bases. Tutorial is good for people getting into accessibility. It will teach you about semantic markup, different accessibility issues and things like that.

And now I will pass it over to John for a live demonstration.

>> John Toles: Okay. Can everybody see my screen?

>> Johan Rempel: Sure can. Yup.

>> John Toles: Okay. So, I am working for this demonstration most of the time how I start any test is by opening the site in Firefox. I usually do my work work in chrome and I have Firefox set up so it clears cookies and it is a cleaner experience when I'm going through a website.

I've got bookmarks across the top to all the tools I use most commonly.

So usually starting an evaluation I will run either aXe or Code Sniffer ‑‑ usually both because I want to find what one misses and the other doesn't. To run aXe that has to be done through the developer page. The inspector tools and then there's a tab once you install the aXe tool extension. There's a tab specifically for that. So I'll bring that up and scan the page and it generates a report of all of the issues it finds on the page. And then I'll generally go through these to double check that there aren't any false positives. Usually if there's a false positive it's going to be a color contrast issue and that's most of the time not really a false positive. It's just reporting it because it can't determine what the foreground color it or the background color. So it found four of this one.

Let me pull this down so I can show the highlighting feature.

You have an option to highlight any issues that you're finding. I'll turn that on so ‑‑ on this menu across the top it's saying this is a contrast issue. That's probably correct. I would want to double check it because it's not giving me an exact value in the report.

We can see that looking at the report it's saying it's 2.5 to 1. So it's well below the 4.5:1 ratio.

So you can tab through each issue. It looks like it's pulling up each incident of that saying color combination. When I'm writing up the report, I would only report that once. It's going to report it every time because it doesn't realize it's part of a menu.

So the next thing is the same. No lang tag. That's rarely false positive because it analyze the code and determines whether or not there's a proper tag all or not. It's finding the same thing that Rayianna pointed out earlier that one of the images doesn't have alt text. The most it's finding is the form elements. There are no landmarks. That's generally ‑‑ it's saying there's a div structure on the page and none of the divs have been assigned a role for a structure or for landmarking purposes.

If you have any questions or want to go through any particular point, feel free to post that in the chat and then John or Rayianna will let me know. That's a brief overview of aXe. It's got the most straight forward interface. It lists everything and you want to go through point by point either adding them to your report or checking that they're false positives or not.

Next, I want to use the HTML Code Sniffer to find anything that aXe would have missed. Generally I like to go through point by point. So I do all the errors first. It's saying there's no language attribute. There's not sufficient color contrast, image missing alt attribute. I'm not seeing anything that wasn't pick said up by aXe but if we look at say this particular form field it's saying this form field should be labeled in some way use the label element with a for attribute or title or ARIA label. So it tells you what the area is and like Rayianna showed earlier it gives you the success criteria to compare against and list a suggested technique. It's saying there's no associated label with this form field.

This is slightly different way of saying it. I'll go through the warnings now. Whereas aXe is going to indicate that heading markup should be used if content is intended as a heading. AXe will report that as an error. In Code Sniffer that's a warning because it ‑‑ two different interpretations of the WCAG criteria. If you have headings they need to be marked up as headings but there's nothing saying you have to have headings. There's another interpretation that says content should be marked up with headings if it's a certain criteria but that's an AAA so that's not usually included. AXe includes it as best practice but Code Sniffer is including it as a warning.

The notices are generally minor stuff that usually contains a lot of false positives or doesn't relate to document accessibility and it's just points to check and certain things where the automated tool can't check that the title element describes the document for example. So it's saying make sure the ‑‑ check that the title element describes the document. That's something you have to do manually. Generally the title element will relate to the tab title I guess is the best way to say that. And this one is accessibility university demo site is a pretty good title.

And one of the best things about Code Sniffer is it sits on your screen here but you can move it around. If you're looking through ‑‑ if you're comparing what Code Sniffer is reporting on the page you can drag it around. You don't have to have it in the same place.

After I've done that, I will go through and check individual ‑‑ go through the user journey on the page. I'll do that with ANDI open. That way I can watch as I tab through the page. I want to find the first interactable element on the page. The ANDI provides the highlight. Then the first is the about link. And I tab through and check the output that ANDI is providing, and make sure that it matches the text on the screen or that it's there.

We have a "click here" link there. I'll do that for several elements on the page. You can choose what you want to go through in the select module. You can also go through element by element using the tab here. I like to tab through the page so I can check the layout of the page, the order is logical. For instance, if I got to the about and tab through the menu and it immediately jumped over to the apply now that means the page doesn't have a logical layout because you want to go through the content first and then the sidebar.

I don't think those are accessible.

Then I'll go through each thing that ANDI will report. I will go through graphics. That's where you have to use the tab here. You can't tab through graphics. ANDI will read the outline. It's saying there's no alt. So it won't describe ‑‑ provide feedback to the user. See the horizontal line graphic. Those should all be alter out because they're not important. They should have ‑‑ these are used for layout. They should have loan tags.

So I go through and check the graphics, links, buttons. Something useful with ANDI is checking structural elements like tables and headings. For instance, I know there's a table on this ‑‑ in this section. I will use ANDI to check that this is marked up as a table. So that it includes all the proper table elements. This is heading. These are marked as TD tags. They should be TH tags. This should also be H tags. These would all be reported as individual data points reported by the screen reader. That's something you would only be able to tell ‑‑ it seems ANDI seems to be outputting the correct thing. The output is in here. It will read correctly but additional information provided by the screen reader would be that this is a column heading. If it's marked up correctly. I can tell just from looking that it's not but it might be difficult testing it without a screen reader. So I would probably mark that in my report to come back and check that table with a screen reader.

And for structural elements it will check for headings. We'll see zero headings. So we can click on these and it will take us to each one. Element visually conveys heading meaning but is not using semantic mark up. That's one of the more useful things is checking structural elements.

As I'm going through and tabbing through, I will have made note of color contrast wise from aXe and spot check those with color contrast analyser. I'll bring the color contrast analyser and I remember from the report that these were reported as having issues. If there's any question, I can tell just by looking at it that aXe was definitely correct. It will be a violation. If I need to really check if I want to double check that aXe was correct I would either find the hex value in the code and input the hex value ‑‑ so go to inspect and then look for the actual hex color in the code and copy that. So 2.5:1. So it doesn't meet minimum for regular text or enhance text and doesn't meet for UI components either.

And then if I can't find the exact value of the code, I will use the eye dropper tool. The eye dropper tool is a little difficult because there's modern browsers will try to create a smoother image around the text. So you can see there's a slight color variation. So you want to try to pick the darkest part of any individual letter. So I will go for this color here. The back ground is usually easier to do. So it's slightly different value but it's close enough that it's probably the same contrast ratio. Like Rayianna was saying, you have choices in the drop down for how you want to set up the color. If you're testing something that doesn't have hex value, you can use RGB or saturation as well which is what the HSL and HSD are.

That's generally very rarely do we use that. We almost always do hex or use the eye dropper to pick the actual color.

Any questions so far?

>> Johan Rempel: No questions have come up yet. This is a good opportunity if you want to unmute yourself or type in the chat. Just let us know if you have any questions or comments.

>> John Toles: That's most of the stuff that has visual elements to look at. I think we're at 4 o'clock. If anybody has any specific questions or they want to see any features demonstrated specifically, feel free to let me know.

>> Johan Rempel: I'll add one thing here that I find with the color contrast analyser that you demonstrated very helpful with if you're evaluating documents like PDF's or Word documents or PowerPoint it's handy because it's not web based. It's a powerful tool if you don't have internet access. All the trainings that you received with document accessibility it's a useful tool with that. Even with the define of websites at the earliest stages it can provide a quick reality check of color contrast.

>> John Toles: I just realized I didn't demo WAVE.

With the WAVE tool ‑‑ the badges like Rayianna explained apply to each element that it finds. If it finds a lot of stuff, it can get really busy. This is pretty minimal for what you'll find in WAVE. You can pull up code at the bottom. So if you want to go to a specific badge it will jump to that point in your code on the page.

So that's handy for when I need to pull out a Code Sniffer to put in the report or if I'm not sure why it's been tagged and I need to look at the code itself. So I'll look at badges. I want to turn on because it can get busy in the details tab on the left, I will turn off everything that I don't want to look at. So if I want to start off looking at errors, I will just have the errors turned off. If there's a lot of errors I will go through category by category. This isn't too bad. So you can click on it and each one will jump to each individual point in the code and that will move the page at the top to that point as well and it will make it flash. It's hard to ‑‑ WAVE is intended for someone who can see all this stuff. For me it's helpful because I'm a visual person. The badges are helpful for me. If you can't see all the stuff then it doesn't do a lot of good. So the assumptions is you are someone who can make use of the badges and want that visual feedback.

As I go through, I will turn off errors and leave color contrast on. I like to use WAVE specifically to test ARIA because it's hard to find ARIA on pages a lot of times. It gets included in various ways because a lot of ARIA ends up on a page because it gets copied from stack over flow or a code repository. It's not set up correctly and creates problems when you're using a screen reader but you don't get what's going on because the ARIA is deep in the page. So I like to use WAVE to jump specifically to ARIA errors on a page. But this is not finding any ARIA. Then it's not reported at all.

A back up for a lot of screen reader users if I'm trying to test something specifically like a workflow or some element on a page that I can get to or some task in the page to complete, a fall back for a lot of screen reader users to turn off styling. It removes the CSS and creates a text page. WAVE offers an easy way to do that. A way to as a last stitch effort to get to it is turn off styling. I flip that on and it will display the page as base text. Then I'll try to complete the issue ‑‑ complete the task with my screen reader with absolutely no styling. That's just a quick way of doing it. It's hard to do in modern browsers, especially this quickly especially just having a switch.

I think that's it for WAVE. We didn't find any ARIA. So I'm testing structural elements is easy in WAVE because it does put the badge on it. It differentiates the beiges. If these were marked up as headings it will include a different badge for ‑‑ to indicate this is a heading. Since I know this should be a heading, I know this isn't marked up as headings. If WAVE was the only thing, I was using that would be a good indicator. It's another point to point out that these are headings but WAVE is not putting a badge on it because WAVE doesn't know they're headings because they're not marked up as headings. The badges are helpful because they're there and they're helpful because they're not there. I know they should be there.

Okay. So quick demo of WAVE. If you have any questions feel free to post them in the chat or unmute yourself. And we'll take the last 10 minutes here for questions.

>> Johan Rempel: Great. John f you can just unshare your screen and if Rayianna you want to share the resources there. I did post the bookmarklets by Paul Adams for accessibility testing in the chat but those will be included in the resources as well. I misspoke. I thought they were in the resources but I ‑‑ are they on the next page? No, I don't think so. Those will be included when you get the PowerPoint as well. Just an FYI.

All right. Hopefully this was helpful. Rayianna and John really know their stuff. They've been at this for a while. Any questions from anybody?

>> John Toles: One point I would like to make we had a question about what was the best combination or the best workflow for using all of these tools and it's really going to depend on your own comfort level. We use multiple tools because we've been doing this a while and we have a workflow built around using different tools. If you're just starting out and getting used to the whole concept of testing for accessibility and what you need to look for you may want to start with one tool. You may want to start with aXe or ANDI and get used to using that and as you get more comfortable and your workflow becomes easier then you add more tools so you can find more issues. The more ‑‑ it's really a question of the more you look the more issues you will find. It's really your comfort level with the tools and the principals of accessibility.

>> Johan Rempel: Well stated, John. All right. Well we will go ahead and wrap it up then unless there are any questions coming in. Thank you all for your time. Thank you Rayianna and John for a very detailed and comprehensive overview of these automated testing tools. Next week I will present on the functionality end of screen readers so we'll address more of the wide question and how this impacts the end user from a screen reader perspective. Thank you all. And thank you Molly. Molly says thanks CIDI. Useful stuff. We appreciate you taking time for this. Once again this is being recorded. I will send all of this to Allie McDougall. The recording, the accessible PowerPoint and the transcript and a big thank you to Heather again for providing the transcripts today. So, any other final words from Rayianna or John before we close out?

>> Rayianna Daniels: Thank you, everyone for attending. I hope you learned something.

>> John Toles: I don't have anything additional.

>> Johan Rempel: Thank you. Enjoy the rest of your day.