Accessibility begins in planning: how to develop accessible websites?

Online Conference by Cédric Anderson
(Laboratoire NT2)
June 10, 2020

This conference is part of *Interrogating Access*, a series of talks and workshops on accessibility in art and media production developed by OBORO and Spectrum Productions with the support of the Canada Council for the Arts. OBORO and Spectrum Productions acknowledge that their activities take place in Tiohtià:ke, an unceded Kanien'kehá:ka territory.

(Beginning of transcription)

Hello. My name is Cédric Anderson, and today I'll talk about website accessibility in Quebec. Before we begin, I'd like to thank OBORO for inviting me. I hope you'll enjoy the presentation today.

On the agenda:

- We'll look at who I am and what I do.
- We'll take a look at website accessibility in Quebec.
- WCAG 2.0, which is the protocol we use in Canada.
- WAI-ARIA, which supports HTML tags.
- Accessible HTML.
- Online tools that you can use to check whether a website is accessible.
- Website accessibility in various scenarios.

I've been working with websites since 2003. I work as a full-stack developer, so I deal with the visible and background aspects when we're building a website. I work mainly for public and private institutions, small businesses, artists and self-employed individuals. I work with WordPress a lot of the time. I build custom themes for the WordPress platform. I also work at UQAM, at the NT2, which is a research lab in digital arts and literature.

Here is the definition of accessibility for the Web:

"A service or content provided over the Internet is considered accessible when any person, whatever their disability, can understand, browse through, and interact with it. Web accessibility is an important factor for social integration and community participation."

There's a reference at the bottom of the slide if you'd like to access the government website.

In Quebec, as we speak, very few organizations have to follow accessibility standards, aside from the government itself. Colleges and universities have to, but for the most part, only government organizations and ministries have to be accessible. These two slides with links at the bottom explain who has to apply accessibility standards. If you're an artist, an arts centre, a small business or a non-profit, you don't have to provide an accessible website.

WCAG 2.0 is a rule book that can be used to ensure that a website will be accessible by most people. There are many, many standards. It's a very large document. But we can sum it up in 4 points: a website has to be perceivable, operable, understandable, and robust. You'll find a link at the end of this document for an in-depth explanation of WCAG.

WAI-ARIA is a series of properties you can add to HTML tags to give the content some context. When building a website and creating a content box, developers can only say, "this is a box." But it's hard to define exactly what the content is. WAI-ARIA provides a series of HTML properties that can be used to specify the contents of each box used in a website. For example: "This is the opening hours" or "That's navigation." So with WAI-ARIA you can define roles, properties, statuses. ARIA properties should only be used when absolutely necessary. There are HTML tags that are called semantic

elements. On their own, they can give a general idea of what the contents are. For example, you have the tags: NAV, ARTICLE, FOOTER. These semantic elements should be used before applying WAI-ARIA rules.

Let's take a look at the tools we can use to create an accessible website. The first thing is the semantic elements I mentioned. They can give context to the content. They also help with keyboard navigation. For people who have a motor disability or use a screen reader, it's much easier to navigate without using a mouse, only with a keyboard.

It's also much simpler for developers. When creating a website, using semantic elements is very simple. It's also simple for responsive sites, which work equally well on a computer screen or a mobile screen. And it helps optimization for search engines. Google loves semantic websites, and will provide better search ratings when the right elements are used.

For text, although there's no fixed rule, it's best to follow logic. The language should be clear, for example, using "without" and not "w/o" with a slash, which doesn't mean anything for a screen reader. Abbreviations should be avoided, and if they're used, the ABBR tag should be used, which is the "abbreviation" tag.

A sufficient font size should be used, the minimum size being 16 pixels. Under 16 pixels, the text is very small. For low-vision users, it can be difficult. Text colour should be highly visible in contrast with the background colour. For example, this presentation uses white text of a dark grey background, which is easy to read. However, the blue letters used for the title may be harder to read for some.

Using the right tags. I often see developers using incorrect tags. The right tags should always be used. For example, for a page header, H1, H2, and H3 should be used instead of the paragraph tag. Also, HTML attributes shouldn't be used for design. It's really a question of separating visuals and content—contents are handled with HTML, and visuals by CSS.

For hyperlinks, the link text should be short, simple, clear and descriptive. For example, a link that says "Click Here" doesn't mean anything out of context. It should read instead, for example, "To buy a ticket, please click here." Using complete sentences for links is absolutely fine.

When possible, avoid using the same text for different links. For those of you who don't know about screen readers, they're used mainly by blind and low-vision users. The screen reader is a voice that reads what's on the page, line after line. You can imagine that if you have 5 "Click Here" buttons on a page, it'll be hard to figure out which does what. So different text should be used for each link of the page.

Use the TITLE attribute whenever possible. The Title attribute is used for images and links. It's not visible for screen readers, so shouldn't contain important text. The Title attribute is the text that pops up when you hover over a link with the mouse. Make sure to use a description to specify where the link goes.

When you add a link to a file, specify its size and use the DOWNLOAD attribute. There's an attribute in HTML called "download," and it should be used. You should also add, if possible, the size of the downloadable file. It'll help the user decide if they have the necessary resources.

Don't repeat the URL in the link text. For example, if you have a link to google.com, don't use "www.google.com" as the text. Instead, use: "Click here to access the Google search engine."

When the link opens in a new tab, mention it. Screen readers don't alert the user that a new tab has been opened. So it's useful to mention in the link that it will open in a new tab.

And use "skip links." It's important, we'll talk about it later.

TABINDEX: when someone is using the keyboard to navigate the site, they'll use the Tab key a lot. It's the key that moves the selection from one item to the next on a web page, and is used far more than the arrow keys. The tabindex can be used to change the default navigation flow in a page. Imagine the navigation in the top menu of a page. When you hover on "About," a submenu pops up. Let's say the default navigation flow when I press Tab is: Home > About > Products > Services > Contact, for example, but when the user gets to "About," instead of going to the next item, I want them to access the submenu. I can change this in the tabindex. With this change, when the user presses on Tab, the flow will now be Home > About

> The Team > The Company > Products > Services > Contact, etc.

The tabindex can make an item accessible with keyboard navigation. Normally, when I press the Tab key, mainly links will be selected. If I press Tab, it will go through the navigation. At the end, it'll bring me to the first link on the page, etc. But sometimes, you may want the user to access an item on the page, and you can do that by setting the tabindex at 0. So the section I coded as tabindex="0" will be visible for the tabindex. I can also do the opposite, and make an item unavailable for keyboard navigation, by setting the tabindex at -1.

For images: Always use the ALT attribute. If you use the IMG tag, writing IMG (space) ALT should become automatic. The ALT attribute contains the alternate text for an image. It's a short description of what the image is. So, for example: "Traffic sign indicating a left turn." Or it could be: "Artwork: Painting, oil on canvas," etc. So, it's one sentence or two short sentences that convey what the image contains.

For decorative images, leave the ALT attribute blank. These are all the images that aren't useful for the content, and have a purely aesthetic function. You should still use the ALT attribute, but leave it blank. That way, screen readers will skip them.

The Title attribute for help bubbles—I mentioned this earlier.

Use the LONGDESC attribute (meaning "long description") when including diagrams or complex images. However, note that these are not read by all screen readers, though most can. Long descriptions are used when a diagram is used to illustrate statistics, for example, or pie charts. These are hard to describe well in just one sentence. With a single sentence in the ALT attribute, you'll have something like "Statistical Diagram of Visitors on the Site in 2019," for example. This doesn't read what the statistics are for a blind person. Using the long description attribute will enable you to add a link to a longer description. When the user accesses this long description, it should be a complete description explaining that 10% is attributed to one thing, 25% to another, etc. in our statistics example.

Use descriptive file names: If I have a .JPEG image, for example. Some screen readers can read file names. So it's a good idea to use file names

that are descriptive, instead of just using “img001.jpeg,” for example.

If the image description is in a side paragraph, use the ARIA-LABELEDBY attribute. I mentioned ARIA earlier. As an example, this is an attribute that enables you to indicate that a paragraph next to the image is used to describe that image.

You should also use the tags FIGURE and FIG CAPTION, which are semantic elements that say: this figure contains an image, a description, a paragraph and an MP3, for example. This can be used to group several elements that are part of the image.

Forms: They should be simple, clear, and logical. You can sometimes see forms on several pages: Section 1, Section 2, Section 3 … with a “Next” button on each page. It’s already frustrating for someone who doesn’t have a visual disability. Imagine what it’s like for someone who uses a screen reader.

Ensure that forms work with keyboard navigation by using the tabindex. Forms are a great example for the tabindex that I was explaining earlier. If I design a form on two columns, normally the Tab function will go down the left column first, then down the right column. But maybe I’d like for the navigation to go line by line: left-right, left-right, left-right. I’ll be able to set that up with the tabindex.

Using the LABEL, or ARIA-LABEL attributes: Another mistake that’s common on websites is when someone doesn’t like to have the field name right above the field. They prefer placing the field name inside the field. By default, you have to use a placeholder in the INPUT tag, but the placeholder isn’t read by screen readers. So the Label attribute should be used for each field. You can use Float Label instead if you want to be able to move it, and place it inside the field, for example.

Use ARIA-DESCRIBEDBY for long explanations: If I have a complicated form with a long explanation, I’ll use the aria-describedby property to make it clear that this is the description of the form.

Also, errors should be kept in context: Another common mistake is when the user forgets to input their email address in a form, or types it incorrectly,

some forms will bring the user back to the top of the page where all the error messages are grouped together. This shouldn't happen. Errors should be written in at each field. So if I forget to type in my email address, the error should appear next to that field.

Tables: Use the TH tags and the SCOPE attribute for headers. I rarely see these elements used. They're used to define the function of the first column, the second column, etc. They give some context for lines and columns in a table. Use THEAD, TBODY and TFOOTER: These don't help screen readers, but help to keep items in context, like I just explained.

More notes on HTML elements:

Items hidden with VISIBILITY: HIDDEN or DISPLAY: NONE aren't visible for screen readers. Use positioning with Z-INDEX instead. One of the things that visibility: hidden is often used for is accordions. If I have an accordion on a page and I click the arrow, it'll hide the previous item and display the next item. Often the visibility: hidden property will be used. This is a problem, as the content won't be seen by screen readers. Instead, a z-index or another method should be used. Content shouldn't be hidden if that content is relevant, for users with a visual disability.

Same with a canvas: if you have a highly dynamic site with a lot of animations, often a canvas will be used. It should be referenced via JavaScript, or with ARIA to ensure that there is at least a text explaining to users with a visual disability what is supposed to happen on this canvas.

Use the BUTTON tag. Don't use the DIV tag to create buttons. This is, again, to ensure that contents are in the correct context and that semantic HTML is used. A DIV box can contain anything and shouldn't be used instead of a Button tag, where the name says it—it's a button, an interactive item.

There should be an adequate contrast between the foreground and the background. A pale blue title on a dark blue background will make it hard to read for a large number of people with vision issues.

Don't place important content in JavaScript. Same thing for screen readers. A JavaScript script normally shouldn't contain text. It should be available to

everyone, so ensure to keep text in HTML elements.

Here I have 3 online tools: The first is a website that can be used to check the colour contrast between the foreground and the background. The next 2 are extensions that can be used to check the colour contrast and a number of other items, for example whether images all have their ALT properties. I specified the Firefox extension and the Chrome extension. I don't use Safari—I searched for it, but didn't find anything. I recommend using one of those 2 browsers if you want to check a website's accessibility.

Scenarios: What is an accessible website…

For users on the autistic spectrum:

Use simple colours, plain language, short sentences, bullet lists, descriptive buttons, and simple and consistent layouts. User experience should be as simple and clear as possible. This is the most important aspect.

For screen readers, and for blind and low-vision users:

Describe images and provide transcripts for videos—I know that video transcripts and subtitles can be a problem in some cases, but if you want to ensure that a website is 100% accessible, videos should be subtitled. Follow a linear, logical layout—for example, this means that forms on two columns should be easy to navigate using the Tab key. Use semantic elements, keyboard navigation only and descriptive links and headings. We looked at these items previously.

For users with low vision and other visual impairments, including colour blindness:

Use good colour contrasts and a readable font size. Publish all information on the web page (not in images, videos or PDF files)—the contents of a web page should never be in a PDF file; you can convert the PDF content to HTML and add a link to that PDF. Use a logical combination of colours, shapes and text. Follow a linear, logical layout. Keep buttons and notifications in context—don't place form error messages at the top of the page, for example, as I explained earlier.

For users with dyslexia:

Use images and diagrams to support text. Align text to the left and keep a consistent layout—complex visuals may be fun, but with dyslexia, they're not helpful. Consider producing materials in other formats—if someone finds reading difficult, a video, a podcast or an audio recording can help. Keep content short, clear and simple. Let users change the colours of the background and foreground—if you use WordPress, there are a number of plug-ins that let the user change the site interface to suit their needs, and many are free.

For users with physical or motor disabilities:

Make easily clickable actions—I often see buttons that are right up close to some text or the edge of the screen, making them hard to reach. Give clickable elements space—same, for example with someone using a mobile phone with the thumb: if the space around it is too narrow, it's hard to select an item. Design for keyboard or speech-only use. Mobile First: Start your design with the mobile version, and continue with tablet and PC versions. Minimize interactions—for example, ask users for their postal code only, instead of a complete address. Do you really need to ask for complete contact information to subscribe a user to a newsletter? Probably not. So, don't ask users for information you don't need, and try to minimize interactions. It's been said that a whole site should be accessible in 3 clicks. In this case, it's particularly important.

For users who are deaf or hard of hearing:

Use plain language, keep it simple. Use subtitles or provide transcripts for videos. Use linear, logical layouts. Break up content with subheadings, images and videos which can simplify reading. Let users add a comment when booking appointments (for example, to ask for an interpreter).

For users with anxiety:

Give users enough time to complete an action—on some government sites or other sites where confidential information is submitted, there will often be a timer on forms. For example, a user may be required to complete a form in 20 minutes. Why not make it 60 minutes? Users should have enough

time to easily complete an action, without the added stress of running out of time.

Explain what will happen after completing an action. When you have a form with a “Submit” button, perhaps before this button, add a message that says, “You will be redirected to a page with a confirmation that your form has been received.”

Emphasize important information with bold or large characters. Give users the support they need to complete an action—for example, if you provide a highly complex form on the site, perhaps add a video on how to fill out this form. Let users check their answers before they submit a form—when the user clicks on “Send,” this could be a pop-up that reads: “Have you double-checked such and such section?”

Here are a number of references for WCAG 2.0 and government standards for website accessibility. If you go to quebec.ca/en/accessibility, you can read about the process used by the Quebec government to make their website accessible.

a11yqc.org is a Quebec non-profit that helps organizations make their websites accessible. ukhomeoffice—This is a series of posters about accessibility. The scenarios I mentioned were sourced from this document. And then WAI-ARIA 1.0, etc.

That’s all! That’s what I had to say about website accessibility in Quebec.

As I mentioned at the beginning, it isn’t mandatory in Quebec to have an accessible website. However, it should be included in your plans when creating a site. In fact, it should be one of the first topics of discussion.

Making a website accessible isn’t difficult. It doesn’t require much more work than a non-accessible site. It isn’t the part of website development that’s the most costly.

So making an accessible site is a must. There are so many advantages to having an accessible site, and very, very few disadvantages. We should all be making accessible sites, and ensure that everyone can access our site.

That’s all. If I’m not mistaken, there will be a discussion planned with OBORO. I’ll let them fill you in.

Thank you very much.

Transcription and translation: Marie Lauzon, trad. a. (Canada)

(End of transcription)

To cite this conference: Cédric Anderson “Accessibility begins in planning: how to develop accessible websites [L’accessibilité commence dans la planification : comment développer des sites Web accessibles?]” (10 June 2020). Conference presented as part of OBORO and Spectrum Productions’ *Interrogating Access* series. Available online: http://www.oboro.net/en/activity/accessibility-begins-planning-how-develop-accessible-websites

OBORO

www.oboro.net

SPECTRUM PRODUCTIONS

www.productionsspectrum.com

Conseil des arts du Canada | Canada Council for the Arts

(End of document)