Walk a mile in someone else's shoes

> Empathy – the action of understanding, being aware of, being sensitive to, and vicariously experiencing the feelings, thoughts, and experience of another of either the past or present without having the feelings, thoughts, and experience fully communicated in an objectively explicit manner;
>
> — Merriam-Webster

How many times have we told a teammate or a potential client "that's easy" or "piece of cake" when it comes to doing something like a Wordpress theme or installing a web server from source? Even if you have the data to back up the assertion that it's easy we may still come across as arrogant or "know it all" and they would probably be right.

I saw this video after I had a conversation about Wordpress development where I claimed that customizing a Wordpress theme should not be complicated and how I've done multiple customizations using different techniques to do so.

Then it made me think... How hard was setting up a Wordpress installation in 2006 when I posted the first content in my blog? I remember why I moved away from Movable Type and I realized that I take those beginner experiences for

granted and dismiss them quickly because I've already done this a hundred times.

This is also when I hear when I hear Tal Oppenheimer and Bruce Lawson speak about the next Billion users. We take connectivity for granted and believe we will always be able to optimize our applications to get the best speed possible.

There are cultural issues: Bruce makes a point that there are some people in Thailand who use one name only; if we require first and last name in a form (for example to pay for an item) people will either lie or not use your application. What other similar gems hide behind your users?

So how do we become more empathetic as designers?

I know this will come like an arrogant asshole but please bear with me.

The first step is to remember who we are doing this for. It may be yourself in 6 months, your circle of developer friends in your town or developers halfway across the world. They are people and they may do things differently than we do.

I remember that when I first started teaching how to build online courses a trainer form the High Tech Center Training Unit of the California Community Colleges showed videos of users with disabilities working with online courses. It was an eye opening and humbling experience. Things I took for granted when using keyboard and mouse were a whole different experience when using the keyboard as the only navigation tool.

So we need to account for our users. We'll look at it from different aspects:

- research
- design
- accessibility
- process

Walk a mile in someone else's shoes: Research

Walk a mile on their shoes is more than a platitude. I have to admit I'm the first one who some times takes for granted that the page will load in x seconds because that's how long it takes for the content to load in my laptop with my connection at home or in my phone.

Some of the questions I should ask:

- What devices do they use to access your content?
- What accessibility tools do they use to work with your material?
- Does their network connectivity influence how they interact with your content? Your app?
- Do I need to provide for localization of the content to the users' language or will English work as a sensible default?
- Do I have people from the target audience available for interview and ongoing feedback?

Answers to these questions will help define the design and development process. Having access to the devices our target use help us get a better sense of performance for our application.

If you've heard Alex Rusell's presentations on performance and Javascript (or you've seen me put videos of his presentations on post and articles I write) you already know that DevTools device emulation and even Page Speed Test mobile testing are only a starting point as they will never fully emulate a device being used in your target area.

I will not go into the differences between desktop and mobile devices; Alex does that much better than I can. The important part is that we need to be mindful of how we test our application.

Walk a mile in someone else's

shoes: Design

When we talk about design it's easier to forget why we go through all these hoops to do design and development. We want to present visually stunning products but we also want to build experiences and uses interfaces that work for our target audiences.

The figure below shows a user centered design and development process where we include our target audiences as early as possible and throughout the design process.

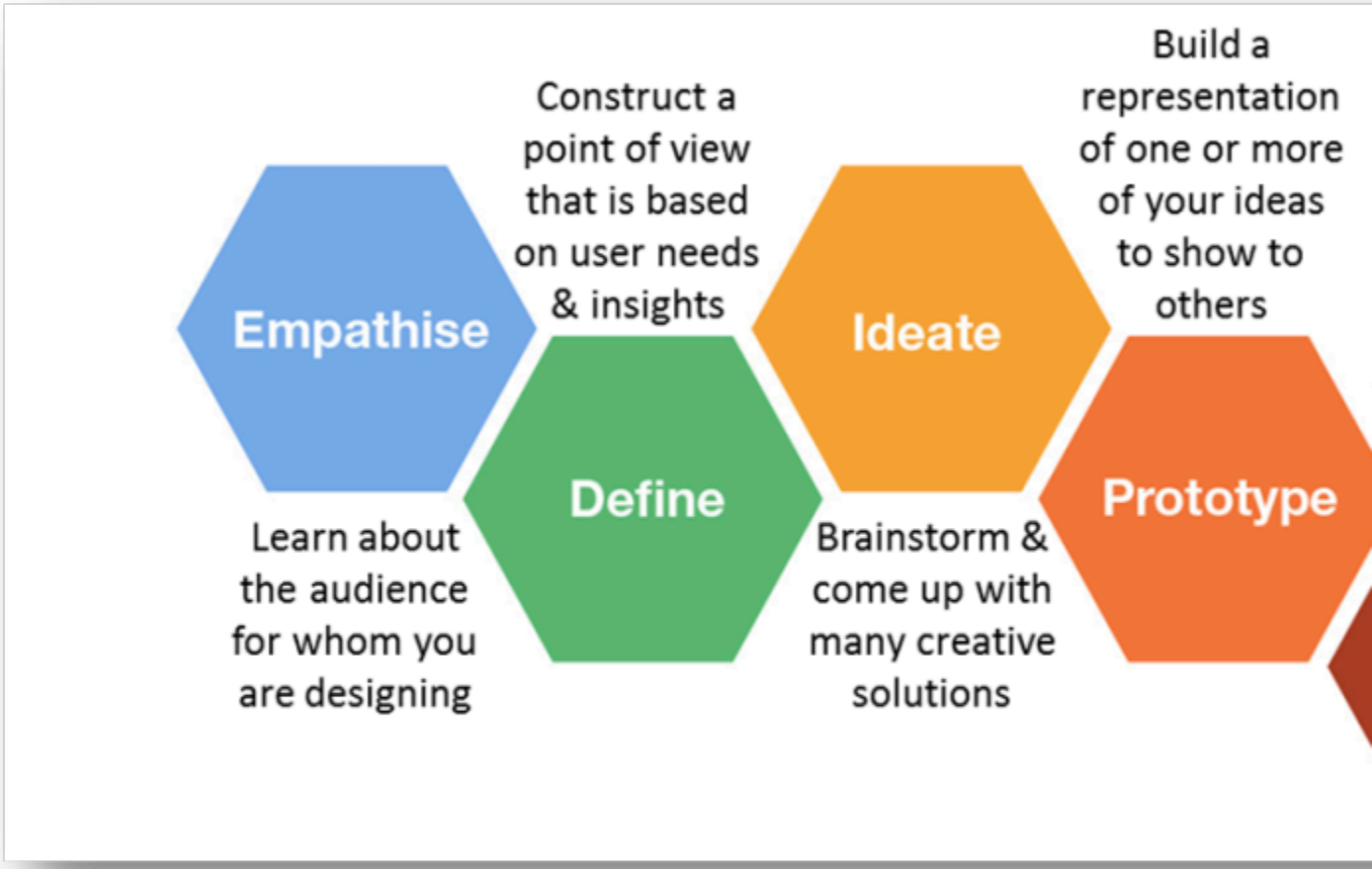

Figure 1: How empathy applies throughout the design process. Credit Shortie Designs

By including users at the earliest stages of the project we can get a better understanding of what they need and what our products (web sites, web apps and other deliverables) need to offer.

I accept that adding users from our target populations early in the design

process may not always be possible. If it's not then we should research the cultural norms of the country or countries we want to work in.

One of my biggest surprises in this area came when researching a different article when I called for us to pay attention to the next billion users and where they would come from.

The table below, taken from Creating Culturally Customized Content for Website Translation shows how different colors are perceived in different cultures.

COLOR	USA	China	India	Egypt	Japan
Red	Danger Love Stop	Good fortune Luck Joy	Luck Fury Masculine	Death	Anger Danger
Orange	Confident Dependable Corporate	Fortune Luck Joy	Sacred (the Color Saffron)	Virtue Faith Truth	Future Youth Energy
Yellow	Coward Joy Hope	Wealth Earth Royal	Celebration	Mourning	Grace Nobility
Green	Spring Money New	Health Prosperity Harmony	Romance New Harvest	Happiness Prosperity	Eternal life
Blue	Confident Dependability Corporate	Heavenly Clouds	Mourning Disgust Chilling	Virtue Faith Truth	Villainy
Purple	Royalty Imagination	Royalty	Unhappiness	Virtue	Wealth
White	Purity Peace Holy	Mourning	Fun Serenity Harmony	Joy	Purity Holiness
Black	Funeral Death Evil	Heaven Neutral High Quality	Evil	Death Evil	

I know that this is a lot of work. We can find ourselves in three different categories of websites (From Creating Culturally Customized Content)

- **Standard websites**: this type of website has only one language (usually English) and the same content is intended to reach all countries. No effort was made in terms of website translation, website localization or website internationalization.
- **Localized websites**: this type of website ranges from websites with some translated landing pages, websites that are fully translated (localized), to websites that not only offer users with translated content, but also with content specific to their country or locale.
- **Culturally customized websites**: this type of website not only takes into account the language and locale of the target audience, but also one or more levels of cultural adaptation: perception, symbolism and behavior.

English is not bad a universal language but it does a diservice to people in the local community that cannot understand it. Assuming that you have the resources we'll explore a few ways you can use to display content in languages other than English

From a technical stand point it's easier to go from comps to visual design but how does the design address different languages, and different needs

Unicode and why it's important

Before we jump in to the mechanics of displaying content in languages other than English on the web, we need to take a little detour and speak about `character encodings`, `character sets`, `code points`, `ASCII`, `Unicode` and `fonts`. These definitions have helped me get a better handle on how to work with languages other than English.

Character Sets, Code Pages and Character encodings

Information derived from Character sets, coded character sets, and encodings

Whenever we work with text in a computer we need to worry about the language we're writing in and how will the interpreted and displayed to the user. Until Unicode came around the main problem was that the same number will represent different characters depending on what encoding we use.

A **character set** is the set of characters we use to support a given language, be it English, Mandarin or Celtic languages. This is independent of the medium we

use to write the characters.

A **coded character set** (also refered to as code page) is a set of characters where each character has a unique number assigned to it. Units of a coded character set are known as code points.

A **code point** value represents the position of a character in the coded character set. For example, the code point for the letter á in the Unicode coded character set is 225 in decimal, or E1 in hexadecimal notation.

The **character encoding** reflects the way the coded character set is mapped to bytes for manipulation in a computer.

It is also important to note that, depending on the encoding, a character may take more than one byte of storage space. We'll look at this in more detail when we discuss ISO-8859 and Unicodde.

ASCII

ASCII (American Standard Code for Information Interchange) is an older 7-bit encoding used in teleprinters and early Internet devices. ASCII encodes 128 specified characters into seven-bit integers. The characters encoded are numbers 0 to 9, lowercase letters a to z, uppercase letters A to Z, basic punctuation symbols, control codes that originated with Teletype machines, and a space.

ASCII was OK while the Internet was only used in Eglish (either native speakers or researchers who spoked the language) but it became an issue as the Internet grew more multicultural and exploded when the web became a public space and a graphical space with Mosaic and early versions of Netscape and Internet Explorer.

ISO/IEC 8859

The **ISO-8859** family of standard, or more specifically ISO/IEC 8859, seeks to address the shortcomings of ASCII when it comes to languages other than English. These other languages need additional characters that are outside the range of the English alphabet. Because ASCII already exhausted the number of characters that you can use with 7 bits so they added another 96 characters by using 2 additional group of characters. Wikipedia describes the 8859 encodings as:

> The ISO/IEC 8859-n encodings only contain printable

> characters, and were designed to be used in conjunction with control characters mapped to the unassigned bytes. To this end a series of encodings registered with the IANA add the C0 control set (control characters mapped to bytes 0 to 31) from ISO 646 and the C1 control set (control characters mapped to bytes 128 to 159) from ISO 6429, resulting in full 8-bit character maps with most, if not all, bytes assigned.
>
> From: ISO/IEC 8859

Because it is impossible to fit all characters from all languages the 8859 standard is broken in to 16 different code pages represented by a different number in the extension. For example: 8859-1 corresponds to the basic Latin alphabet used in English and most Western European countries; 8859-14 represents Celtic languages, such as Irish, Manx, Scottish Gaelic, Welsh, Cornish, and Breton.

Two final things to know about these encodings.

Because they only include printable characters the 8859 character sets lack typographical tools such as ligatures, curly quotation marks, dashes and others. To work with high end typography many programs use Unicode or provide their own additions to the 8859 standards for a given language.

The specifications are no longer beig actively maintained. The working group that developed the 8859 standards disbanded in June, 2004. The only working group that works in character sets is concentrating efforts in Unicode's Universal Coded Character Set (UCS) defined in ISO/IEC 10646.

Unicode

So now we come to **Unicode**. Unlike ISO 8859 and ASCII, Unicode provides a unified registry for all languages. The codepoint for a letter in a given alphabet will not change regardless of the code page the character and its associated language is in.

> The Unicode Standard provides a unique number for every character, no matter what platform, device, application or language. It has been adopted by all modern software providers and now allows data to be transported through many different platforms, devices and applications without corruption. Support of Unicode forms the foundation for the

> representation of languages and symbols in all major operating systems, search engines, browsers, laptops, and smart phones—plus the Internet and World Wide Web (URLs, HTML, XML, CSS, JSON, etc.). Supporting Unicode is the best way to implement ISO/IEC 10646.
>
> From What is Unicode?

Unicode is not a font or a font system. It makes it easier for font developers to know what characters are contained in each page and make it easier to support one language or family of languagesper font and it also makes it easier to know that if a font supports Unicode for a given codde page the characters will be the same no matter what OS or what tool we're using.

Also, because we are dealing with languages other than English where most characters require one byte to be represented we may have to use more than one byte to represent a character.

That's where the UTF-8 enconding comes in.

UTF-8 (Unicode Transformation Format - 8 bit) encodes each of the 1,112,064 valid code points in Unicode using one to four 8-bit bytes or octet in Unicode parlance. The first 128 characters (equivalent to US-ASCII) need one byte. The next 1,920 characters need two bytes to encode, which covers the remainder of almost all Latin-script alphabets, and also Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac, Thaana and N'Ko alphabets, as well as Combining Diacritical Marks. Three bytes are needed for characters in the rest of the Basic Multilingual Plane, which contains virtually all characters in common use including most CJK (Chinese, Japanese and Korean) characters. Four bytes are needed for characters in the other planes of Unicode, which include less common CJK characters, various historic scripts, mathematical symbols, and emoji (pictographic symbols).

Paragraph adapted and condensed from Wikipedia's UTF-8 entry

So why is this important?

If we're going to cover a language in our web pages, we owe it to the users in that country to treat it correctly and to provide fonts that actually work with the languagees in question. It also forces Unicode to stay up to date with languages to avoid problems like those that happened in Myanmar/Burma.

Because there has been little or no development of the Internet in Myanmar there was no reaal desire to work on Fonts that supported the Myanmar languages as a commercial venture or for anyone outside Myanmar to support development of free fonts. Myanmar's cell phone usage was 1% in 2010, exploding to over 80% in 2016, and about 80% of those are smartphones. Google Search launched as google.com.mm in March 2013, including a Burmese user interface. Facebook added Burmese language support in June 2013, and the list of supporting companies continues to grow.

The solution was a locally developed font called Zawgyi. It became highly popular and the majority font use in Myanmar. We'll come back to Zawgyi when we compare it to Unicode.

It wasn't until Unincode 3.0 that a codeblock was assigned to Myanmar and related languages, Currrently (as of version 10.0.0) includes 160 code points (U+1000...U+109F) for Burmese, Mon, Karen, Kayah, Shan, and Palaung languages of Myanmar. It is also used to write Pali and Sanskrit in Myanmar.

The presentations below from IMUG's July Meeting illustrate the issue better than I can and in a lot more detail than I have space for in this article.

Unicode came into the picture after Zawgyi was released and the conflicts within them make the two fonts incompatible and working with one means that characters may not render well with the other. Zawgi takes the code points that Unicode uses for other languages on the code block and uses them for Burmese glyphs without providing replacement code points for those other languages.

So as front end developers targeting content to Myanmar we have three options. We use Zawqgyi and let our Unicode users suffer when the text in Myanmar or any of the other languages in the code block display properly, we use fonts that support Unicode and hope that our users have the same or similar fonts installed or we use English and hope our target audience can understand what we're saying.

The point is, again, to walk a mile on our users' shoes. How would you handle this disconnect between the font that people use in Myanmar and how you handle the people who chose not to use your idea (Zawgyi or Unicode) to view your content?

We write different characters in

different ways

Figures with sideways writing modes, will not work on Chrome. If you want to see what they look like, try viewing the page in Firefox.

People who I deeply admire and whose work I respect have been working on how to layout languages other than English and Latin languages on the web and how we can combine these languages in the same layout.

I will concentrate on direction, writing modes and text orientation as these are the easiest ones to work with and the ones that will have a big impact on how text appears on your web content.

The examples taken from Jen Simmons Workshop Website for AEA San Francisco, 2016 show different writing modes, direction and vertical alignment.

Figure 1 is the standard that we're used for English and most European languages, including Greek and Cyrillic. We read from top to bottom and left to right.

Basic defaults

(1) Around 1040, the first known movable type system was created in China by Bi Sheng out of porcelain. Sheng used clay type, which broke easily, but Wang Zhen by 1298 had carved a more durable type from wood.

(2) Copper movable type printing originated in China at the beginning of the 12th century. It was used in large-scale printing of paper money issued by the Northern Song dynasty. Movable type spread to Korea during the Goryeo Dynasty.

(3) Around 1230, Koreans invented a metal type movable printing using bronze. The Jikji, published in 1377, is the earliest known metal printed book.

(4) Around 1450, Johannes Gutenberg introduced what is regarded as the first

modern movable type system in Europe.

Figure 2: direction: ltr; writing-mode: horizontal-tb;

Something like **figure 2** was my first experience with languages and layouts other than English. Arabic and Hebrew write text from right to left. Figure 2 also shows what English looks like when written from right to left instead our traditional.

Set Direction to RTL

Around 1040, the first (1) known movable type system was created in China by Bi Sheng out of porcelain. Sheng used clay type, which broke easily, but Wang Zhen by 1298 had carved a more durable type from wood.

(2) النحاس نوع المنقولة الطباعة نشأت في الصين في بداية القرن 12th. انه كان يستخدم في الطباعة على نطاق واسع من النقود الورقية التي تصدرها سلالة سونغ الشمالية. نوع المنقولة انتشار لكوريا خلال عصر مملكة كوريو.

(3) حول 1230، اخترع الكوريين الطباعة المنقولة نوع المعدن باستخدام البرونزية. وجيكجي، التي نشرت في 1377، هو أقرب المعادن طبع كتاب معروف.

(4) حوالي 1450، قدم يوهانس غوتنبرغ ما يعتبر أول نظام نوع المنقولة الحديثة في أوروبا.

Figure 3: *direction: rtl;* writing-mode: horizontal-tb;

The first big surprise came when I saw figure 3 and later in Jen’s presentation When working with languages other than Latin and European languages. We can write languages like Japanese like we write English (horizontal, left to write) or we can write them vertically (vertical top to bottom, right to left).

Set Writing Mode to vertical-rl

(1) Around 1040, the first known movable type system was created in China by Bi Sheng out of porcelain. Sheng used clay type, which broke easily, but Wang Zhen by 1298 had carved a more durable type from wood.

(2) Copper movable type printing originated in China at the beginning of the 12th century. It was used in large-scale printing of paper money issued by the Northern Song dynasty. Movable type spread to Korea during the Goryeo Dynasty.

(3) 大约1230年，韩国人发明了一种使用青铜的金属型可移动印刷。吉吉，出版于1377年，是最早知道的金属印刷书。

(4) 约1450年左右，约翰内斯古登伯格介绍了被认为是欧洲第一个现代可移动式系统。

Figure 4: direction: ltr; *writing-mode: vertical-rl;* text-orientation: mixed;

We can do the same thing while flipping the text to go from left to right. The English reads a little weird but it’s not the target language for this layout.

Set Writing Mode to vertical-lr;

(1) Around 1040, the first known movable type system was created in China by Bi Sheng out of porcelain. Sheng used clay type, which broke easily, but Wang Zhen by 1298 had carved a more durable type from wood.

(2) Copper movable type printing originated in China at the beginning of the 12th century. It was used in large-scale printing of paper money issued by the Northern Song dynasty. Movable type spread to Korea during the Goryeo Dynasty.

(3) Around 1230, Koreans invented a metal type movable printing using bronze. The Jikji, published in 1377, is the earliest known metal printed book.

(4) Around 1450, Johannes Gutenberg introduced what is regarded as the first modern movable type system in Europe.

Figure 5: direction: ltr; *writing-mode: vertical-lr;* text-orientation: mixed;

Sideways text addresses another set of non western / non european languages. We can also use sideways text to create interesting layouts for our content.

Set Writing Mode to sideways-rl

(1) Around 1040, the first known movable type system was created in China by Bi Sheng out of porcelain. Sheng used clay type, which broke easily, but Wang Zhen by 1298 had carved a more durable type from wood.

(2) Copper movable type printing originated in China at the beginning of the 12th century. It was used in large-scale printing of paper money issued by the Northern Song dynasty. Movable type spread to Korea during the Goryeo Dynasty.

(3) 大约1230年，韩国人发明了一种使用青铜的金属型可移动印刷。吉吉，出版于1377年，是最早知道的金属印刷书。

(4) Around 1450, Johannes Gutenberg introduced what is regarded as the first modern movable

Figure 6: direction: ltr; *writing-mode: sideways-rl;*

type
system
in
Europe.

Set Writing Mode to sideways-lr

(1) Around 1040, the first known movable type system was created in China by Bi Sheng out of porcelain. Sheng used clay type, which broke easily, but Wang Zhen by 1298 had carved a more durable type from wood.

(2) Copper movable type printing originated in China at the beginning of the 12th century. It was used in large-scale printing of paper money issued by the Northern Song dynasty. Movable type spread to Korea during the Goryeo Dynasty.

(3) 大约1230年，韩国人发明了一种使用青铜的金属型可移动印刷。吉吉，出版于1377年，是最早知道的金属印刷书。

(4) Around 1450, Johannes Gutenberg introduced what is regarded as the first modern movable

Figure 7: direction: ltr; *writing-mode: sideways-lr;*

type
system
in
Europe!

The final piece of surprise was the vertical layout for languages like Japanese. It’ll work with English and other western languages but, as you can seein figures 8 and 9, it looks odd when used with western languages but looks as intended when used wwith Japanese and other eastern languages.

Upright

1 Around 1040, the first known movable type.
2 Copper movable type printing originated in China.
3 1230 左右，韩国人发明了一种金属型活动印刷
4 Around 1450, Johannes Gutenberg.

Figure 8: direction: rtl;
writing-mode: vertical-rl;
text-orientation: upright;

Upright

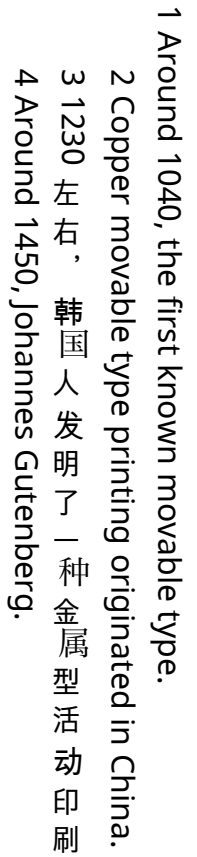

Figure 9: direction: ltr;
writing-mode: vertical-lr;
text-orientation: upright;

You may wonder why we went into this excursion into writing modes, writing directions and text orientations. It’s important to know that we can write text in the direction, writing mode and orientation that is appropriate to the languages you’re working on. We can also use these writing modes combined to do direct quotations in one language versus another. Knowing about this also helps when working with localized landing pages or we need to understand how languages other than English and European languages will impact our layouts.

Again, put the users (not just English speakers and European languages) first.

Understanding analytics and their impact

Another thing to do when looking at what our users do in our site is to see where they come from and what device they are browsing with. I used to not care as much about where users where coming from as much as I cared about what browsers they were using.

But then I started looking at the logs and some strange browsers appeared in the logs. Not many and not often enough to warrant making changes but enough to take a look at these funky browsers: Opera Mini and UC browser.

Opera Mini and UC browsers are proxy browsers. Rather than do all the processing in the device they send the URL to a central server to do the rendering and, limited, Javascript and CSS. The server sends a single file containing the server rendered version of your page (without most of your CSS and Javascript) back to the client at a fraction of the size of the individual resources put together. As you can imagine this helps when you access the web with a low end devices that will not handle your 2MB of Javascript graceffully.

The reason why this is important is **learning about your user**. It helps you figure out the combination of georgraphical location, browser and (mobile) devices for your users. If you're working on emerging markets or if you have users from countries you were not expecting it may help you to research those browsers.

Accessibility

Accessibility is like dirty laundry we all know we should worry about it but we never do until we have no choice (and I'm the first one to admit that I suck at being proactive about addressing accessibility issues in my projects).

Let's take this hipothetical example:

Your team close to releasing an application that uses the latest and greatest technologies. It's a progressive web application that uses web componets for the front end and Express to talk to MongoDB and IndexedDB to store data in the user's browser.

The app works wonderfully and your team gets a lot of kudos from users and

management.

Then you get a call from a coworker that tells you the application is not quite working as expected. After you panic and freak out for a few seconds you ask your coworker what is not working as expected.

What he tells you floors you... *the keyboard tab order doesn't work as expected*.

You invite your coworker to your desk and ask him to explain what he means by his comment about tab order. He sits at your computer and starts navigating (or trying to navigate) the app and you realize that it jumps all over the place, not in the order you'd expect it.

The two of you hunker down and figure out that the `tabindex` attribute wasn't used properly. After looking over the code you discovered the following issue

- Developers used it in some elements and not others

Further more when revieing how to use `tabindex` you discovered that:

- An element with tabindex="0", an invalid value, or no tabindex attribute should get focus after elements with positive tabindex values in the sequential keyboard navigation order
- Setting tabindex attribute on a `<div>` element forces us to set tabindex on the div children content, otherwise the content inside the div is ignored when navigating with they keyvboard

Your work together in fixing the tabindex issues and pushed the changes. They work and your coworker is happy that the changes work.

The accessibility evaluation firm that you hired to do an audit of your site calls you a few days later and they ask to meet with you and your development team to discuss some of their findings.

The meeting was surprising. They ran pages of your application through a screen reaader and you team was amazed at how it sounded. When you look at the report you realize that there are many images with no `alt` attributes and form fields without labels.

They further suggested developers use a screen reader to work oveer the site.

This would help them understand some of the issues facing vision impaired users.

The actual world

So how do we address accessibility?

In an ideal world we would have users with disabilities test your site and give you feedback but the range of accessility issues is too large to expect every aspect of it to be tested.

We don't realize that the web platform, the technology stack that lives under our frameowrks and APIs, does a lot to make the web accessible to people using assistive technologies. When we build out own components or extend the platform it becomes our responsibility to make sure the content remains accessible.

This may cause extra work for you as a developer. For example you'll have to add accessibility attributes and things that we take for granted when working on the web. Things like keyboard navigation, focus, and the functionality of custom versions of web elements like buttons, checkboxes and other elements.

The W3C Web Accessibiliity Initiative provides a set of authoring practices that give you an idea of the amount of work that you need to do to make accessible elements. Some times this is by choice but sometimes you don't really have an option and need to build a custom version of an existing element or create something completely new for your application.

If you're using custon elements, the howto-tabs sample element shows how to incorporate labels for customm elements.

If you're working with React you can check Facebook Accessibility will show you the basics and an example of an accessible accordion will show you a specific example of how to build accessibility on your React elements and components.

An interesting option is to include tools like Axe Core command line or build system plugins as part of your build process. This will help you identify the issues but it's up to you and your team to fix them. It takes away the excuse that you didn't know how to do it.

Lighthouse, a tool from Google provides a series of reports that cover performance and accessibility. YOu can run this as command line utility, a Chrome browser extension or a build system plugin

Going back to our central premise of walking on our users' shoes let's try using a screen reader for our content. Rob Dodson has created basic tutorials for Voice Over (screen reader built in to macOS) and NVDA (a free screen reader for Windows).

Once we start working with accessibility it's tempting to concentrate in a small area of accessibility and not pay attention to areas that we might not consider accessibility related.

Seren Davies' presentation from Render 2017 covers some of these non visual and non permanent conditions. This is important because all our users have suffered some form of non permanent accessibility impairment and we don't always take these issues into consideration.

Process and tooling

The final empathy area I want to discuss is process and tooling. Who are we doing this for and what tools we're using to develop

Who?

There are three groups of people we should emphatize with: Fellow team members, External developers who use our code and the end users of our product.

Our fellow team memebers are typically the smallest group (some times it's just us), but they are most affected by the code we write. Because we share the same context it's easier for us to write code that they can work with easily. If we miss the mark here they will most definitely grill us with feedback and suggestions for improvement. If it's just us we can certainly make the changes to improve the project.

External Developers build on top of the code and APIs we create. The typical example is a Javascript framework we create and they consume. We shouldn't assume that they know as much about our framework as we do. Instead of being having direct access to the Framework's developers for questions and feedback, this group often has our code and its supporting documentation to solve issues with our code.

The final group affected by our code is our end users. These are the people using applications that developers (internal or external) create. This is usually the

largest group of people our code will affect, and they probably not developers and don't share much context with us. While we may never interact directly with end users, we need to consider how our code can help the integration of features, including – but not limited to – full accessibility and perceived performance.

So how do we walk on the shoes of each of these groups. I believe that the first part of the answer is communication both internally within the team and with external developers. How we document the code we write (both inline comments in the code and documentation outside the code itself) and the interfaces we provide to our libraries and APIs should be easy to understand and well documented without having to be a Javascript expert.

Tooling

Another way to make things easier for developers and, eventually, end users is to provide good tooling for the tools you create.

One thing I've always loved about tools like NPM, Yeoman, Polymer CLI, Imagemin and other command line tools is that they are very clear in what they do and what they want the user to do to configure the project using the tool. They also give you ways to edit or overwrite the configuration files created in the CLI so you can change the way the tool behaves when it comes to building the application.

Make it as easy as possible for users who are using your tools. Provide documentation and remember that the user of your project may be you in six months and may not remember how you implemented it.

Conclusion

I know we've jumped all over the place but I want to distill some final thoughts

Put the user first when designing and developing content. Don't make changes without testing the new changes. As the quote below shows, even the best intentioned changes can can have unexpected consequences.

> Seamless is a food delivery service that's incredibly popular on the East Coast. In 2013, they merged with GrubHub, and recently, the two companies combined their backends and released a refreshed user interface.

> The reaction to these updates was really strong. It was a total “who moved my cheese” scenario, as customers had gotten used to the application’s flow, regardless of its inefficiencies. Customers had already expended the mental how to get from Point A (logging in, searching, placing order), to Point B (purchasing food), and it worked for them despite how counter-intuitive it may have seemed.
>
> We need to remember that familiarity is a more powerful concept than intuition. While we should always strive to make easy-to-use software, we need to remember that this doesn’t serve the people who already know how to use the software as it exists today.
>
> From: No Flex Zone: Empathy Driven Development

Accessibility is important. Accessibility should be an essential component of any design and development process. And we should be working on accessibility acommodations for visually impaired or hearing impaired users. More accessibility issues exist than those we normally pay attention to when working with accessibility.

We may need to accessibility features ourselves for short periods of time. Imagine that you have an accident or surgery that will change the way(s) how you access your content, even if it’s temporary.

So I have to constantly remind myself that it’s not always **easy**. I’m just more experienced.

Links, resources and further ideas

- Character encodings: Essential concepts
- ASCII
- A Closer Look At Personas: What They Are And How They Work (Part 1)
- A Closer Look At Personas: A Guide To Developing The Right Ones (Part 2)
- Informing Your Empathy for More Human Designs and Communities
- Why Empathy is the Secret Sauce for Good Software Development
- EMPATHY IS THE NEW KILLER APP
- No Flex Zone: Empathy Driven Development
- Why You Must Use Empathy Effectively in UX Design

- USER EMPATHY IN WEB DESIGN
- Empathy: an essential skill in Software Development
- Why empathy is a crucial skill for web designers
- Building websites and empathy my experience as a frontend design apprentice
- 8 Ways To Become A More Empathic Designer
- Speed up your development process by coding with empathy
- Developing empathy
- Practicing Empathy in Product Design
- Using VoiceOver to Evaluate Web Accessibility
- Using NVDA to Evaluate Web Accessibility