
24
2014
24 WAYS

Credits

24 ways is the advent calendar for web geeks. For twenty-four days each December we publish a daily dose of web design and development goodness to bring you all a little Christmas cheer.

- *24 ways* is brought to you by Perch CMS
- Produced by Drew McLellan, Brian Suda, Anna Debenham and Owen Gregory.
- Designed by Paul Robert Lloyd.
- eBook published by edgeofmyseat.com and produced by Rachel Andrew.
- Possible only with the help and dedication of our authors.

2014

The web turned twenty-five and showed no sign of settling down in semi-detached suburbia. In October, HTML5 was released as a W3C Recommendation. Back in May, 24 ways was very excited and grateful to win the net award for best collaborative project – a huge thank you to all our authors, readers and supporters!

1. What It Takes to Build a Website

Drew McLellan 24ways.org/201401

In 1994 we lost Kurt Cobain and got the world wide web as a weird consolation prize. In the years that followed, if you'd asked me if I knew how to build a website I'd have said yes, I know HTML, so I know how to build a website. If you'd then asked me *what it takes* to build a website, I'd have had to admit that HTML would hardly feature.

Among the design nerdery and dev geekery it's easy to think that the nuts and bolts of building a page just need to be multiplied up and Ta-da! There's your website. That can certainly be true with weekend projects and hackery for fun. It works for throwing something together on GitHub or experimenting with ideas on your personal site. But what about working professionally on client projects?

The web is important, so we need to build it right.

It's 2015 – your job involves people paying you money for building websites. What does it take to build a website and to do it right? What practices should we adopt to make really great, successful and professional web projects in 2015? I put that question to some friends and *24 ways* authors to see what they thought.

GETTING THE TECH RIGHT

Inevitably, it all starts with the technology. We work in a technical medium, after all. From Notepad and WinFTP through to continuous integration and deployment – how do you build sites?

Create a stable development environment

There's little more likely to send a web developer into a wild panic and a client into a wild rage than making a new site live and things just not working. That's why it's important to have realistic development and staging environments that mimic the live server as closely as possible.

Are you in the habit of developing new sites right on the client's server? Or maybe in a subfolder on your local machine? It's time to reconsider.

Charlie Perrins writes:

> Don't work on a live server – this feels like one of those gear-changing moments for a developer's growth. Build something that works just as well locally on your own machine as it does on a live server, and capture the differences in the code between the local and live version in a single config file. Ultimately, if you can get all the differences between environments down to a config level then you'll be in a really good position to automate the deployment process at some point in the future.

Anything that creates a significant difference between the development and the live environments has the potential to cause problems you won't know about until the site goes live – and at that point the problems are very public and very embarrassing, not to mention unprofessional.

A reasonable solution is to use a tool like MAMP PRO which enables you to set up an individual local website for each project you work on. Importantly, individual sites give you both consistency of paths between development and live, but also the ability to configure server options (like PHP versions and configuration, for example) to match the live site.

Better yet is to use a virtual machine, managed with a tool such as Vagrant. If you're interested in learning more about that, we have an article on that subject later in the series.

Use source control

Trent Walton writes:

> We use source control, and it's become the centerpiece for how we handle collaboration, enhancements, and issues. It drives our process.

I'm hoping by now that you're either using source control for all your work, or feeling a nagging guilt that you should be. Be it Git, Mercurial, Subversion (name your poison), a revision control system enables you to keep track of changes, revert anything that breaks, and keep rolling backups of your project.

The benefits only start there, and **Charlie Perrins** recommends using source control "not just as a personal backup of your code, but as a way to play nicely with other developers."

Noting the benefits when collaborating with other developers, he adds:

> Graduating from being the sole architect of your codebase to contributing to a shared codebase is a huge leap for a developer. Perhaps a practical way for people who tend to work on their own to do that would be to submit a pull request or a patch to an open source project or plugin."

Richard Rutter of Clearleft sees clear advantages for the client, too. He recommends using source control "preferably in some sort of collaborative environment that you can open up or hand over to the client" – a feature found with hosted services such as GitHub.

If you'd like to hone your Git skills, Emma Jane Westby wrote Git for Grown-ups in last year's 24 ways.

Don't repeat, automate!

Tim Kadlec is a big proponent of automating your build process:

> I've been hammering that home to every client I've had this year. It's amazing how many companies don't really have a formal build/deployment process in place. So many issues on the web (performance, accessibility, etc.) can be greatly improved just by having a layer of automation involved.
>
> For example, graphic editing software spits out ridiculously bloated images. Very frequently, that's what ends up getting put on a site. If you have a build process, you can have the compression automated and start seeing immediate gains for no effort. On a recent project, they were able to shave around 1.5MB from their site weight simply by automating compression.

Once you have your code in source control, some of that automation can be made easier. **Brian Suda** writes:

> We have a few bash scripts that run on git commit: they compile the less, jslint and remove white-space, basically the 3 Cs, Compress, Concatenate, Combine. This is now part of our workflow without even realising it.

One great way to get started with a build process is to use a tool like Grunt, and a great way to get started with Grunt is to read Chris Coyier's Grunt for People Who Think Things Like Grunt are Weird and Hard.

Tim reinforces:

> Issues like [image compression] — or simple accessibility issues like alt tags on images — should never be able to hit a live server. If you can detect it, you can automate it. And if you can automate it, you can free up time for designers and developers to focus on more challenging — and interesting — problems.

A clear call to arms to tighten up and formalise development and deployment practices. The less that has to be done manually or is susceptible to change, the less that can go wrong when a site is built and deployed. Any procedures that are automated are no longer dependant on a single person's knowledge, making it easier to build your team or just cope when someone important is out of the office or leaves.

If you're interested in kicking the FTP habit and automating your site deployments, we have an article later in the series just for you.

BUILD SYSTEMS, NOT SITES

One big theme arising this year was that of building websites as systems, not as individual pages.

Brad Frost:

> For me, teams making websites in 2015 shouldn't be working on just-another-redesign redesign. People are realizing that in order to make stable, future-friendly, scalable, extensible web experiences they're going to need to think more systematically. That means crafting deliberate and thoughtful design systems. That means establishing front-end style guides. That means killing the out-dated, siloed, assembly-line waterfall process and getting cross-disciplinary teams working together in meaningful ways. That means treating development as design. That means treating performance as design. That means taking the time out of the day to establish the big picture, rather than aimlessly crawling along quarter by quarter.

Designer and developer **Jina Bolton** also advocates the use of style guides, and recommends making the guide a project deliverable:

> Consider adding on a style guide/UI library to your project as a deliverable for maintainability and thinking through all UI elements and components.

Val Head agrees: "build and maintain a style guide for each project" she wrote. On the subject of approaching a redesign, she added:

> A UI inventory goes a long way to helping get your head around what a design system needs in the early stages of a redesign project.

So what about that old chestnut, responsive web design? Should we be making sites responsive by default? How about mobile first?

Richard Rutter:

> *Think* mobile first unless you have a very good reason not to. Remember to take the client with you on this principle, otherwise it won't work as a convincing piece of design.

Trent Walton adds:

> The more you can test and sort of skew your perception for what is typical on the web, the better. 4k displays hooked up to 100Mbps connections can make one extremely unsympathetic.

The value of testing with real devices is something **Ruth John** appreciates. She wrote:

> I still have my own small device lab at home, even though I work permanently for a well-established company (which has a LOT of devices at its disposal) – it just means I can get a good overview of how things are looking during development.

And speaking of systems, **Mark Norman Francis** recommends the use of measuring tools to aid the design process; "[U]se analytics and make decisions from actual data" he suggests, rather than relying totally on intuition.

Tim Kadlec adds a word on performance planning:

> I think having a performance budget in place should now be a given on any project. We've proven pretty conclusively through a hundred and one case studies that performance matters. And over the last year or so, we've really seen a lot of great tools emerge to help track and enforce performance budgets. There's not really a good excuse for not using one any more.

It's clear that in the four years since Ethan Marcotte's Responsive Web Design article the diversity of screen sizes, network connection speeds and input methods has

only increased. New web projects should presume visitors will be using anything from a watch up to a big screen desktop display, and from being offline, through to GPRS, 3G and fast broadband.

Will it take more time to design and build for those constraints? Yes, it most likely will. If Internet Explorer is brave enough to ask to be your default browser, you can be brave enough to tell your client they need to build responsively.

WORKING COLLABORATIVELY

A big part of delivering a successful website project is how we work together, both as a design team and a wider project team with the client.

Val Head recommends an open line of communication:

> Keep conversations going. With clients, with teammates. Talking is so important with the way we work now. A good team conversation place, like Slack, is slowly becoming invaluable for me too.

Ruth John agrees:

> We've recently opened up our lines of communication by using Slack. It has transformed the way we work. We're easily more productive and collaborative on projects, as well as making it a lot easier for us all to work remotely (including freelancers).

She goes on to point out how tools can be combined to ease team communication without adding further complications:

> We have a private GitHub organisation (which everyone who works with us is granted access to), which not only holds all our project code but also a team wiki. This has lots of information to get you set up within the team, as well as coding guidelines and best practices and other admin info, like contact numbers/ emails for the team.

Small-A agile is also the theme of the day, with **Mark Norman Francis** suggesting an approach of "small iterations with constant feedback around individual features, not spec-it-all-first". He also encourages you to review as you go, at each stage of the project:

> Always reflect on what went well and what went badly, and how you can learn from that, even if not Doing Agile™. Ultimately "best practices" should come from learning lessons (both good and bad).

Richard Rutter echoes this, warning against working in isolation from the client for too long:

> Avoid big reveals. Your engagement with the client should be participatory. In business no one likes surprises.

This experience rings true for **Ruth John** who recommends involving real users in the feedback loop, not just the client:

> We also try and get feedback on what we're building as soon and as often as we can with our stakeholders/clients and real users.

We should also remember that our role is to serve the client's needs, not just bill them for whatever we can. **Brian Suda** adds:

> Don't sell clients on things they don't need. We can spout a lot of jargon and scare clients into thinking you are a god. We can do things few can now, but you can't rip people off because they are unknowledgeable.

But do clients know what they're getting, even when they see it? **Trent Walton** has an interesting take:

> We focus on prototypes over image-based comps at all costs, especially when meetings are involved. It's much easier to assess a prototype, and too often with image-based comps, discussions devolve into how something might feel when actually live, or how a layout could change to fit a given viewport.

Val Head also likes to get work into the browser:

> Sketch design ideas with any software you like, but get to the browser as soon as possible.

Beyond your immediate team, **Emma Jane Westby** has advice for looking further afield:

> Invest time into building relationships within your (technical) community. You never know when you might be able to lend a hand; or benefit from someone who's able to lend theirs.

And when things don't go according to plan, **Brian Suda** has the following advice:

> If something doesn't work out, be professional and don't burn bridges. It will always come back to you.

The best work comes from working collaboratively, not just as a team within an agency or department, but with the client and stakeholders too. If doing your job and chucking it over the fence ever worked, it certainly doesn't fly any more. You *can* work in isolation, but doing really great work requires collaboration.

THE BUSINESS END

When you're building sites professionally, every team member has to think about the business aspects. Estimating time, setting billing rates, and establishing deliverables are all part of the job.

In 2008, **Andrew Clarke** gave us the Contract Killer sample contract we could use to establish a working agreement for a web design project. **Richard Rutter** agrees that contracts are still an essential part of business:

> They are there for both parties' protection. Make sure you know what will happen if you decide you don't want to work with the client any more (it happens) and, of course, what circumstances mean they can stop taking your services.

Having a contract is one thing, but does it adequately protect both you and the client? **Emma Jane Westby** adds:

> Find a good IP lawyer/legal counsel. I routinely had an IP lawyer read all of my contracts to find loopholes I wouldn't have noticed. I didn't always change the contract, but at least I knew what might come back to bite me.

So, you have a contract in place, and know what the project is. **Brian Suda** recommends keeping track of time and making sure you bill fairly for the hours the project costs you:

> If I go to a meeting and they are 15 minutes late, the billing clock has already started. They can't expect me to be in the 1h meeting and not bill for the extra 15–30 minutes they wasted. It goes both ways too. You need to do your best to respect their deadlines and time frame – this is always hard to get right.

As ever, it's good business to do good business. Perhaps we can at last shed the old image of web designers being snowboarding layabouts and demonstrate to clients that we care as much about conducting professional business as they do.

TIME TO REVIEW

It's a lot to take in. Some of these ideas and practices will be familiar, others new and yet to be evaluated. The web moves at a fast pace, and we need to be constantly reexamining our tools, techniques and working practices. The most important thing is not to blindly adopt any and all suggestions, but to carefully look at what the benefits might be and decide how they apply to your work.

Could you benefit from more formalised development and deployment procedures? Would your design projects run more smoothly and have a longer maintainable life if you approached the solution as a componentised system rather than a series of pages? Are your teams structured in a way that enables the most fluid communication, or are there changes you could make? Are your billing procedures and business agreements serving you and your clients in the best way possible?

The new year is a good time to look at your working practices and see what can be improved, and maybe this time next year you'll look back and think "thank goodness we don't work like *that* any more".

ABOUT THE AUTHOR

Drew McLellan is lead developer on your favourite small CMS, Perch. He is Director and Senior Developer at UK-based web development agency edgeofmyseat.com, and formerly Group Lead at the Web Standards Project. When not publishing 24 ways, Drew keeps a personal site covering web development issues and themes, takes photos and tweets a lot.

2. Dealing with Emergencies in Git

Emma Jane Westby 24ways.org/201402

The stockings were hung by the chimney with care,
In hopes that version control soon would be there.

This summer I moved to the UK with my partner, and the onslaught of the Christmas holiday season began around the end of October (October!). It does mean that I've had more than a fair amount of time to come up with horrible Git analogies for this article. Analogies, metaphors, and comparisons help the learner hook into existing mental models about how a system works. They only help, however, if the learner has enough familiarity with the topic at hand to make the connection between the old and new information.

Let's start by painting an updated version of Clement Clarke Moore's Christmas living room. Empty stockings are hung up next to the fireplace, waiting for Saint

Nicholas to come down the chimney and fill them with small treats. Holiday treats are scattered about. A bowl of mixed nuts, the holiday nutcracker, and a few clementines. A string of coloured lights winds its way up an evergreen.

Perhaps a few of these images are familiar, or maybe they're just settings you've seen in a movie. It doesn't really matter what the living room looks like though. The important thing is to ground yourself in your own experiences before tackling a new subject. Instead of trying to brute-force your way into new information, as an adult learner constantly ask yourself: 'What is this like? What does this remind me of? What do I already know that I can use to map out this new territory?' It's okay if the map isn't perfect. As you refine your understanding of a new topic, you'll outgrow the initial metaphors, analogies, and comparisons.

With apologies to Mr. Moore, let's give it a try.

GETTING INTERRUPTED IN GIT

When on the roof there arose such a clatter!

You're happily working on your software project when all of a sudden there are freaking reindeer on the roof! Whatever you've been working on is going to need to wait while you investigate the commotion.

If you've got even a little bit of experience working with Git, you know that you cannot simply change what you're working on in times of emergency. If you've been doing work, you have a dirty working directory and you cannot change branches, or push your work to a remote repository while in this state.

Up to this point, you've probably dealt with emergencies by making a somewhat useless commit with a message something to the effect of 'switching branches for a sec'. This isn't exactly helpful to future you, as commits should really contain whole ideas of completed work. If you get interrupted, especially if there are reindeer on the roof, the chances are very high that you *weren't* finished with what you were working on.

You don't need to make useless commits though. Instead, you can use the `stash` command. This command allows you to temporarily set aside all of your changes so that you can come back to them later. In this sense, `stash` is like setting your book down on the side table (or pushing the cat off your lap) so you can go investigate the noise on the roof. You aren't putting your book away though, you're just putting it down for a moment so you can come back and find it exactly the way it was when you put it down.

Let's say you've been working in the branch *waiting-for-st-nicholas*, and now you need to temporarily set aside your changes to see what the noise was on the roof:

```
$ git stash
```

After running this command, all uncommitted work will be temporarily removed from your working directory, and you will be returned to whatever state you were in the last time you committed your work.

With the book safely on the side table, and the cat safely off your lap, you are now free to investigate the noise on the roof. It turns out it's not reindeer after all, but just your boss who thought they'd help out by writing some code on the project you've been working on. Bless. Rolling your eyes, you agree to take a look and see what kind of mischief your boss has gotten themselves into this time.

You `fetch` an updated list of branches from the remote repository, locate the branch your boss had been working on, and `checkout` a local copy:

```
$ git fetch
$ git branch -r
$ git checkout -b helpful-boss-branch origin/
helpful-boss-branch
```

You are now in a local copy of the branch where you are free to look around, and figure out exactly what's going on.

You sigh audibly and say, 'Okay. Tell me what was happening when you first realised you'd gotten into a mess' as you look through the log messages for the branch.

```
$ git log --oneline
$ git log
```

By using the `log` command you will be able to review the history of the branch and find out the moment right before your boss ended up stuck on your roof.

You may also want to compare the work your boss has done to the main branch for your project. For this article, we'll assume the main branch is named *master*.

```
$ git diff master
```

Looking through the commits, you may be able to see that things started out okay but then took a turn for the worse.

CHECKING OUT A SINGLE COMMIT

Using commands you're already familiar with, you can rewind through history and take a look at the state of the code at any moment in time by checking out a single commit, just like you would a branch.

Using the `log` command, locate the unique identifier (commit hash) of the commit you want to investigate. For example, let's say the unique identifier you want to checkout is 25f6d7f.

```
$ git checkout 25f6d7f

Note: checking out '25f6d7f'.

You are in 'detached HEAD' state. You can look around,
make experimental changes and commit them, and you can
discard any commits you make in this state without
impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you
create, you may do so (now or later) by using @-b@ with
the checkout command again. Example:

$ git checkout -b new_branch_name

HEAD is now at 25f6d7f... Removed first paragraph.
```

This is usually where people start to panic. Your boss screwed something up, and now your HEAD is detached. Under normal circumstances, these words would be a **very** good reason to panic.

Take a deep breath. Nothing bad is going to happen. Being in a detached HEAD state just means you've temporarily disconnected from a known chain of events. In other

words, you're currently looking at the middle of a story (or branch) about what happened – and you're not at the endpoint for this particular story.

Git allows you to view the history of your repository as a timeline (technically it's a *directed acyclic graph*). When you make commits which are not associated with a branch, they are essentially inaccessible once you return to a known branch. If you make commits while you're in a detached HEAD state, and then try to return to a known branch, Git will give you a warning and tell you how to save your work.

```
$ git checkout master

Warning: you are leaving 1 commit behind, not connected
to
any of your branches:

  7a85788 Your witty holiday commit message.

If you want to keep them by creating a new branch, this
may be a good time to do so with:

$ git branch new_branch_name 7a85788

Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
```

So, if you want to save the commits you've made while in a detached HEAD state, you simply need to put them on a new branch.

```
$ git branch saved-headless-commits 7a85788
```

With this trick under your belt, you can jingle around in history as much as you'd like. It's not like sliding around on a timeline though. When you checkout a specific commit, you will only have access to the history from that point backwards in time. If you want to move forward in history, you'll need to move back to the branch tip by checking out the branch again.

```
$ git checkout helpful-boss-branch
```

You're now back to the present. Your HEAD is now pointing to the endpoint of a known branch, and so it is no longer detached. Any changes you made while on your adventure are safely stored in a new branch, assuming you've followed the instructions Git gave you. That wasn't so scary after all, now, was it?

Back to our reindeer problem.

If your boss is anything like the bosses I've worked with, chances are very good that at least some of their work is worth salvaging. Depending on how your repository is structured, you'll want to capture the good work using one of several different methods.

Back in the living room, we'll use our bowl of nuts to illustrate how you can rescue a tiny bit of work.

SAVING JUST ONE COMMIT

About that bowl of nuts. If you're like me, you probably had some favourite kinds of nuts from an assorted collection. Walnuts were generally the most satisfying to crack open. So, instead of taking the entire bowl of nuts and dumping it into a stocking (merging the stocking and the bowl of nuts), we're just going to pick out one nut from the bowl. In Git terms, we're going to cherry-pick a commit and save it to another branch.

First, checkout the main branch for your development work. From this branch, create a new branch where you can copy the changes into.

```
$ git checkout master
$ git checkout -b rescue-the-boss
```

From your boss's branch, `helpful-boss-branch` locate the commit you want to keep.

```
$ git log --oneline helpful-boss-branch
```

Let's say the commit ID you want to keep is `e08740b`. From your rescue branch, use the command `cherry-pick` to copy the changes into your current branch.

```
$ git cherry-pick e08740b
```

If you review the history of your current branch again, you will see you now also have the changes made in the commit in your boss's branch.

At this point you might need to make a few additional fixes to help your boss out. (You're angling for a bonus out of all this. Go the extra mile.) Once you've made your additional changes, you'll need to add that work to the branch as well.

```
$ git add [filename(s)]
$ git commit -m "Building on boss's work to improve
feature X."
```

Go ahead and test everything, and make sure it's perfect. You don't want to introduce your own mistakes during the rescue mission!

UPLOADING THE FIXED BRANCH

The next step is to upload the new branch to the remote repository so that your boss can download it and give you a huge bonus for helping you fix their branch.

```
$ git push -u origin rescue-the-boss
```

CLEANING UP AND GETTING BACK TO WORK

With your boss rescued, and your bonus secured, you can now delete the local temporary branches.

```
$ git branch --delete rescue-the-boss
$ git branch --delete helpful-boss-branch
```

And settle back into your chair to wait for Saint Nicholas with your book, your branch, and possibly your cat.

```
$ git checkout waiting-for-st-nicholas
$ git stash pop
```

Your working directory has been returned to exactly the same state you were in at the beginning of the article.

HAVING FUN WITH ANALOGIES

I've had a bit of fun with analogies in this article. But sometimes those little twists on ideas can really help someone pick up a new idea (`git stash`: it's like when Christmas comes around and everyone throws their fashion sense out the window and puts on a reindeer sweater for the holiday party; or `git bisect`: it's like trying to find that one broken light on the string of Christmas lights). It doesn't matter if the analogy isn't perfect. It's just a way to give someone a temporary hook into a concept in a way that makes the concept accessible while the learner becomes comfortable with it. As the learner's comfort increases, the analogies can drop away, making room for the technically correct definition of how something works.

Or, if you're like me, you can choose to never grow old and just keep mucking about in the analogies. I'd argue it's a lot more fun to play with a string of Christmas lights and some holiday cheer than a directed acyclic graph anyway.

ABOUT THE AUTHOR

Emma Jane Westby is an author, an educator, and a part-time beekeeper. Her latest videos, Collaborating with Git: Crafting Workflows at the Command Line, are now available from O'Reilly. You can follow her adventures on Twitter at @emmajanehw.

3. JavaScript Modules the ES6 Way

Jack Franklin 24ways.org/201403

JavaScript admittedly has plenty of flaws, but one of the largest and most prominent is the lack of a module system: a way to split up your application into a series of smaller files that can depend on each other to function correctly.

This is something nearly all other languages come with out of the box, whether it be Ruby's `require`, Python's `import`, or any other language you're familiar with. Even CSS has `@import`! JavaScript has nothing of that sort, and this has caused problems for application developers as they go from working with small websites to full client-side applications. Let's be clear: it doesn't mean the new module system in the upcoming version of JavaScript won't be useful to you if you're building smaller websites rather than the next Instagram.

Thankfully, the lack of a module system will soon be a problem of the past. The next version of JavaScript, ECMAScript 6, will bring with it a full-featured module and dependency management solution for JavaScript. The bad news is that it won't be landing in browsers for a while yet – but the good news is that the specification for the module system and how it will look has been finalised. The *even better* news is that there are tools available to get it all working in browsers today without too much hassle. In this post I'd like to give you the gift of JS modules and show you the syntax, and how to use them in browsers today. It's much simpler than you might think.

WHAT IS ES6?

ECMAScript is a scripting language that is standardised by a company called Ecma International. JavaScript is an implementation of ECMAScript. ECMAScript 6 is simply the next version of the ECMAScript standard and, hence, the next version of JavaScript. The spec aims to be fully comfirmed and complete by the end of 2014, with a target initial release date of June 2015. It's impossible to know when we will have full feature support across the most popular browsers, but already some ES6 features are landing in the latest builds of Chrome and Firefox. You shouldn't expect to be able to use the new features across browsers without some form of additional tooling or library for a while yet.

THE ES6 MODULE SPEC

The ES6 module spec was fully confirmed in July 2014, so all the syntax I will show you in this article is not expected to change. I'll first show you the syntax and the new APIs being added to the language, and then look at how to use them today. There are two parts to the new module system. The first is the syntax for declaring modules and dependencies in your JS files, and the second is a programmatic API for loading in modules manually. The first is what most people are expected to use most of the time, so it's what I'll focus on more.

Module syntax

The key thing to understand here is that modules have two key components. First, they have *dependencies*. These are things that the module you are writing depends on to function correctly. For example, if you were building a carousel module that used jQuery, you would say that jQuery is a dependency of your carousel. You *import* these dependencies into your module, and we'll see how to do that in a minute. Second, modules have *exports*. These are the functions or variables that your module exposes publicly to anything that imports it. Using jQuery as the example again, you could say that jQuery exports the `$` function. Modules that depend on and hence import jQuery get access to the `$` function, because jQuery exports it.

Another important thing to note is that when I discuss a module, all I really mean is a JavaScript file. There's no extra syntax to use other than the new ES6 syntax. Once ES6 lands, modules and files will be analogous.

NAMED EXPORTS

Modules can export multiple objects, which can be either plain old variables or JavaScript functions. You denote something to be exported with the `export` keyword:

```
export function double(x) {
  return x + x;
};
```

You can also store something in a variable then export it. If you do that, you have to wrap the variable in a set of curly braces.

```
var double = function(x) {
  return x + x;
}

export { double };
```

A module can then import the `double` function like so:

```
import { double } from 'mymodule';
double(2); // 4
```

Again, curly braces are required around the variable you would like to import. It's also important to note that `from 'mymodule'` will look for a file called *mymodule.js* in the same directory as the file you are requesting the import from. There is no need to add the *.js* extension.

The reason for those extra braces is that this syntax lets you export multiple variables:

```
var double = function(x) {
  return x + x;
}

var square = function(x) {
  return x * x;
}

export { double, square }
```

I personally prefer this syntax over the `export function ...`, but only because it makes it much clearer to me what the module exports. Typically I will have my `export {…}` line at the bottom of the file, which means I can quickly look in one place to determine what the module is exporting.

A file importing both `double` and `square` can do so in just the way you'd expect:

```
import { double, square } from 'mymodule';
double(2); // 4
square(3); // 9
```

With this approach you can't easily import an entire module and all its methods. This is by design – it's much better and you're encouraged to import just the functions you need to use.

DEFAULT EXPORTS

Along with named exports, the system also lets a module have a default export. This is useful when you are working with a large library such as jQuery, Underscore, Backbone and others, and just want to import the entire library. A module can define its default export (it can only ever have one default export) like so:

```
export default function(x) {
  return x + x;
}
```

And that can be imported:

```
import double from 'mymodule';
double(2); // 4
```

This time you do not use the curly braces around the name of the object you are importing. Also notice how you can name the import whatever you'd like. Default exports are not named, so you can import them as anything you like:

```
import christmas from 'mymodule';
christmas(2); // 4
```

The above is entirely valid.

Although it's not something that is used too often, a module can have both named exports and a default export, if you wish.

One of the design goals of the ES6 modules spec was to favour default exports. There are many reasons behind this, and there is a very detailed discussion on the ES Discuss site about it. That said, if you find yourself preferring named exports, that's fine, and you shouldn't change that to meet the preferences of those designing the spec.

Programmatic API

Along with the syntax above, there is also a new API being added to the language so you can programmatically import modules. It's pretty rare you would use this, but one obvious example is loading a module conditionally based on some variable or property. You could easily import a polyfill, for example, if the user's browser didn't support a feature your app relied on. An example of doing this is:

```
if(someFeatureNotSupported) {
  System.import('my-polyfill').then(function(myPolyFill)
{
    // use the module from here
  });
}
```

`System.import` will return a promise, which, if you're not familiar, you can read about in this excellent article on HTMl5 Rocks by Jake Archibald. A promise basically lets you attach callback functions that are run when the asynchronous operation (in this case, `System.import`), is complete.

This programmatic API opens up a lot of possibilities and will also provide hooks to allow you to register callbacks that will run at certain points in the lifetime of a module. Those hooks and that syntax are slightly less set in stone, but when they are confirmed they will provide really useful functionality. For example, you could write code that would run every module that you import through something like JSHint before importing it. In development that would provide you with an easy way to keep your code quality high without having to run a command line watch task.

HOW TO USE IT TODAY

It's all well and good having this new syntax, but right now it won't work in any browser – and it's not likely to for a long time. Maybe in next year's 24 ways there will be an article on how you can use ES6 modules with no extra work in the browser, but for now we're stuck with a bit of extra work.

ES6 module transpiler

One solution is to use the ES6 module transpiler, a compiler that lets you write your JavaScript using the ES6 module syntax (actually a subset of it – not quite everything is supported, but the main features are) and have it compiled into either CommonJS-style code (CommonJS is the module specification that NodeJS and Browserify use), or into AMD-style code (the spec RequireJS uses). There are also plugins for all the popular build tools, including Grunt and Gulp.

The advantage of using this transpiler is that if you are already using a tool like RequireJS or Browserify, you can drop the transpiler in, start writing in ES6 and not worry about any additional work to make the code work in the browser, because you should already have that set up already. If you don't have any system in place for handling modules in the browser, using the transpiler doesn't really make sense. Remember, all this does is convert ES6 module code into CommonJS- or AMD-compliant JavaScript. It doesn't do anything to help you get that code running in the browser, but if you have that part sorted it's a really nice addition to your workflow. If you would like a tutorial on how to do this, I wrote a post back in June 2014 on using ES6 with the ES6 module transpiler.

SystemJS

Another solution is SystemJS. It's the best solution in my opinion, particularly if you are starting a new project from scratch, or want to use ES6 modules on a project where you have no current module system in place. SystemJS is a spec-compliant universal module loader: it loads ES6 modules, AMD modules, CommonJS modules, as well as modules that just add a variable to the global scope (`window`, in the browser).

To load in ES6 files, SystemJS also depends on two other libraries: the ES6 module loader polyfill; and Traceur. Traceur is best accessed through the bower-traceur package, as the main repository doesn't have an easy to find downloadable version. The ES6 module load polyfill implements `System.import`, and lets you load in files using it. Traceur is an ES6-to-ES5 module loader. It takes code written in ES6, the newest version of JavaScript, and *transpiles* it into ES5, the version of JavaScript widely implemented in browsers. The advantage of this is that you can play with the new features of the language today, even though they are not supported in browsers. The drawback is that you have to run all your files through Traceur every time you save them, but this is easily automated. Additionally, if you use SystemJS, the Traceur compilation is done automatically for you.

All you need to do to get SystemJS running is to add a `<script>` element to load SystemJS into your webpage. It will then automatically load the ES6 module loader and Traceur files when it needs them. In your HTML you then need to use `System.import` to load in your module:

```
<script>
  System.import('./app');
</script>
```

When you load the page, *app.js* will be asynchronously loaded. Within *app.js*, you can now use ES6 modules. SystemJS will detect that the file is an ES6 file, automatically load Traceur, and compile the file into ES5 so that it works in the browser. It does all this dynamically in the browser, but there are tools to bundle your application in production, so it doesn't make a lot of requests on the live site. In development though, it makes for a really nice workflow.

When working with SystemJS and modules in general, the best approach is to have a main module (in our case *app.js*) that is the main entry point for your application. *app.js* should then be responsible for loading all your application's modules. This forces you to keep your application organised by only loading one file initially, and having the rest dealt with by that file.

SystemJS also provides a workflow for bundling your application together into one file.

CONCLUSION

ES6 modules may be at least six months to a year away (if not more) but that doesn't mean they can't be used today. Although there is an overhead to using them now – with the work required to set up SystemJS, the module transpiler, or another solution – that doesn't mean it's not worthwhile. Using any module system in the browser, whether that be RequireJS, Browserify or another alternative, requires extra tooling and libraries to support it, and I would argue that the effort to set up SystemJS is no greater than that required to configure any other tool. It also comes with the extra benefit that when the syntax is supported in browsers, you get a free upgrade. You'll be able to remove SystemJS and have everything continue to work, backed by the native browser solution.

If you are starting a new project, I would strongly advocate using ES6 modules. It is a syntax and specification that is not going away at all, and will soon be supported in browsers. Investing time in learning it now will pay off hugely further down the road.

FURTHER READING

If you'd like to delve further into ES6 modules (or ES6 generally) and using them today, I recommend the following resources:

- ECMAScript 6 modules: the final syntax by Axel Rauschmayer
- Practical Workflows for ES6 Modules by Guy Bedford
- ECMAScript 6 resources for the curious JavaScripter by Addy Osmani
- Tracking ES6 support by Addy Osmani
- ES6 Tools List by Addy Osmani
- Using Grunt and the ES6 Module Transpiler by Thomas Boyt
- JavaScript Modules and Dependencies with jspm by myself
- Using ES6 Modules Today by Guy Bedford

ABOUT THE AUTHOR

Jack Franklin is a developer, speaker and author who blogs at javascriptplayground.com and has authored “Beginning jQuery”. Jack works as an engineer at GoCardless and is also a Google Developer Expert for the Chrome HTML 5 platform. He tweets as @jack_franklin and spends far too much time thinking about JavaScript.

4. Developing Robust Deployment Procedures

Rachel Andrew 24ways.org/201404

Once you have developed your site, how do you make it live on your web hosting? For many years the answer was to log on to your server and upload the files via FTP. Over time most hosts and FTP clients began to support SFTP, ensuring your files were transmitted over a secure connection. The process of deploying a site however remained the same.

There are issues with deploying a site in this way. You are essentially transferring files one by one to the server without any real management of that transfer. If the transfer fails for some reason, you may end up with a site that is only half updated. It can then be really difficult to work out what hasn't been replaced or added, especially where you are updating an existing site. If you are updating some third-party software your update may include files that should be removed, but that may not be

obvious to you and you risk leaving outdated files littering your file system. Updating using (S)FTP is a fragile process that leaves you open to problems caused by both connectivity and human error. Is there a better way to do this?

You'll be glad to know that there is. A modern professional deployment workflow should have you moving away from fragile manual file transfers to deployments linked to code committed into source control.

THE BENEFITS OF GOOD PRACTICE

You may never have experienced any major issues while uploading files over FTP, and good FTP clients can help. However, there are other benefits to moving to modern deployment practices.

No surprises when you launch

If you are deploying in the way I suggest in this article you should have no surprises when you launch because the code you committed from your local environment should be the same code you deploy – and to staging if you have a staging server. A missing vital file won't cause things to start throwing errors on updating the live site.

Being able to work collaboratively

Source control and good deployment practice makes working with your clients and other developers easy. Deploying first to a staging server means you can show your client updates and then push them live. If you subcontract some part of the work, you can give your subcontractor the ability to deploy to staging, leaving you with the final push to launch, once you know you are happy with the work.

Having a proper backup of site files with access to them from anywhere

The process I will outline requires the use of hosted, external source control. This gives you a backup of your latest commit and the ability to clone those files and start working on them from any machine, wherever you are.

Being able to jump back into a site quickly when the client wants a few changes

When doing client work it is common for some work to be handed over, then several months might go by without you needing to update the site. If you don't have a good process in place, just getting back to work on it may take several hours for what could be only a few hours of work in itself. A solid method for getting your local copy up to date and deploying your changes live can cut that set-up time down to a few minutes.

THE TOOL CHAIN

In the rest of this article I assume that your current practice is to deploy your files over (S)FTP, using an FTP client. You would like to move to a more robust method of deployment, but without blowing apart your workflow and spending all Christmas trying to put it back together again. Therefore I'm selecting the most straightforward tools to get you from A to B.

Source control

Perhaps you already use some kind of source control for your sites. Today that is likely to be Git but you might also use Subversion or Mercurial. If you are not using any source control at all then I would suggest you choose Git, and that is what I will be working with in this article.

When you work with Git, you always have a local repository. This is where your changes are committed. You also have the option to push those changes to a remote repository; for example, GitHub. You may well have come across GitHub as somewhere you can go to download open source code. However, you can also set up private repositories for sites whose code you don't want to make publicly accessible.

A hosted Git repository gives you somewhere to push your commits to and deploy from, so it's a crucial part of our tool chain.

A deployment service

Once you have your files pushed to a remote repository, you then need a way to deploy them to your staging environment and live server. This is the job of a deployment service.

This service will connect securely to your hosting, and either automatically (or on the click of a button) transfer files from your Git commit to the hosting server. If files need removing, the service should also do this too, so you can be absolutely sure that your various environments are the same.

Tools to choose from

What follows are not exhaustive lists, but any of these should allow you to deploy your sites without FTP.

HOSTED GIT REPOSITORIES

- GitHub
- Beanstalk
- Bitbucket

STANDALONE DEPLOYMENT TOOLS

- Deploy
- dploy.io
- FTPloy

I've listed Beanstalk as a hosted Git repository, though it also includes a bundled deployment tool. Dploy.io is a standalone version of that tool just for deployment. In this tutorial I have chosen two separate services to show how everything fits together, and because you may already be using source control. If you are setting up all of this for the first time then using Beanstalk saves having two accounts – and I can personally recommend them.

PUTTING IT ALL TOGETHER

The steps we are going to work through are:

1. Getting your local site into a local Git repository
2. Pushing the files to a hosted repository
3. Connecting a deployment tool to your web hosting
4. Setting up a deployment

Get your local site into a local Git repository

Download and install Git for your operating system.

Open up a Terminal window and tell Git your name using the following command (use the name you will set up on your hosted repository).

```
> git config --global user.name "YOUR NAME"
```

Use the next command to give Git your email address. This should be the address that you will use to sign up for your remote repository.

```
> git config --global user.email "YOUR EMAIL ADDRESS"
```

Staying in the command line, change to the directory where you keep your site files. If your files are in */Users/rachel/Sites/mynicewebite* you would type:

```
> cd /Users/rachel/Sites/mynicewebsite
```

The next command tells Git that we want to create a new Git repository here.

```
> git init
```

We then add our files:

```
> git add .
```

Then commit the files:

```
> git commit -m “Adding initial files”
```

The bit in quotes after `-m` is a message describing what you are doing with this commit. It's important to add something useful here to remind yourself later why you made the changes included in the commit.

Your local files are now in a Git repository! However, everything should be just the same as before in terms of working on the files or viewing them in a local web server. The only difference is that you can add and commit changes to this local repository.

Want to know more about Git? There are some excellent resources in a range of formats here.

Setting up a hosted Git repository

I'm going to use Atlassian Bitbucket for my first example as they offer a free hosted and private repository.

Create an account on Bitbucket. Then create a new empty repository and give it a name that will identify the repository easily.

Click **Getting Started** and under **Command Line** select "I have an existing project". This will give you a set of instructions to run on the command line. The first instruction is just to change into your working directory as we did before. We then add a remote repository, and run two commands to push everything up to Bitbucket.

```
cd /path/to/my/repo
git remote add origin https://myuser@bitbucket.org/
myname/24ways-tutorial.git
git push -u origin --all
git push -u origin --tags
```

When you run the push command you will be asked for the password that you set for Bitbucket. Having entered that, you should be able to view the files of your site on Bitbucket by selecting the navigation option **Source** in the sidebar.

You will also be able to see commits. When we initially committed our files locally we added the message "Adding initial files". If you select **Commits** from the sidebar you'll see we have one commit, with the message we set locally. You can imagine how useful this becomes when you can look back and see why you made certain changes to a project that perhaps you haven't worked on for six months.

Before working on your site locally you should run:

```
> git pull
```

in your working directory to make sure you have all of the most up-to-date files. This is especially important if someone else might work on them, or you just use multiple machines.

You then make your changes and add any changed or modified files, for example:

```
> git add index.php
```

Commit the change locally:

```
> git commit -m "updated the homepage"
```

Then push it to Bitbucket:

```
> git push origin master
```

If you want to work on your files on a different computer you `clone` them using the following command:

```
> git clone https://myuser@bitbucket.org/myname/
24ways-tutorial.git
```

You then have a copy of your files that is already a Git repository with the Bitbucket repository set up as a remote, so you are all ready to start work.

Connecting a deployment tool to your repository and web hosting

The next step is deploying files. I have chosen to use a deployment tool called Deploy as it has support for Bitbucket. It does have a monthly charge – but offers a free account for open source projects.

Sign up for your account then log in and create your first project. Select **Create an empty project**. Under **Configure Repository Details** choose Bitbucket and enter your username and password.

If Deploy can connect, it will show you your list of projects. Select the one you want.

The next screen is **Add New Server** and here you need to configure the server that you want to deploy to. You might set up more than one server per project. In an ideal world you would deploy to a staging server for your client preview changes and then deploy once everything is signed off. For now I'll assume you just want to set up your live site.

Give the server a name; I usually use *Production* for the live web server. Then choose the protocol to connect with. Unless your host really does not support SFTP (which is pretty rare) I would choose that instead of FTP.

You now add the same details your host gave you to log in with your SFTP client, including the username and password. The **Path on server** should be where your files are on the server. When you log in with an SFTP client and you get put in the directory above *public_html* then you should just be able to add *public_html* here.

Once your server is configured you can deploy. Click **Deploy now** and choose the server you just set up. Then choose the last commit (which will probably be selected for you) and click **Preview deployment**. You will then get a preview of which files will change if you run the deployment: the files that will be added and any that will be removed. At the very top of that screen you should see the commit message you entered right back when you initially committed your files locally.

If all looks good, run the deployment.

You have taken the first steps to a more consistent and robust way of deploying your websites. It might seem like quite a few steps at first, but you will very soon come to realise how much easier deploying a live site is through this process.

YOUR NEW PROCEDURE STEP BY STEP

1. Edit your files locally as before, testing them through a web server on your own computer.
2. Commit your changes to your local Git repository.
3. Push changes to the remote repository.
4. Log into the deployment service.
5. Hit the **Deploy now** button.
6. Preview the changes.
7. Run the deployment and then check your live site.

TAKING IT FURTHER

I have tried to keep things simple in this article because so often, once you start to improve processes, it is easy to get bogged down in all the possible complexities. If you move from deploying with an FTP client to working in the way I have outlined above, you've taken a great step forward in creating more robust processes. You can continue to improve your procedures from this point.

Staging servers for client preview

When we added our server we could have added an additional server to use as a staging server for clients to preview their site on. This is a great use of a cheap VPS server, for example. You can set each client up with a

subdomain – clientname.yourcompany.com – and this becomes the place where they can view changes before you deploy them.

In that case you might deploy to the staging server, let the client check it out and then go back and deploy the same commit to the live server.

Using Git branches

As you become more familiar with using Git, and especially if you start working with other people, you might need to start developing using branches. You can then have a staging branch that deploys to staging and a production branch that is always a snapshot of what has been pushed to production. This guide from Beanstalk explains how this works.

Automatic deployment to staging

I wouldn't suggest doing automatic deployment to the live site. It's worth having someone on hand hitting the button and checking that everything worked nicely. If you have configured a staging server, however, you can set it up to deploy the changes each time a commit is pushed to it.

If you use Bitbucket and Deploy you would create a deployment hook on Bitbucket to post to a URL on Deploy when a push happens to deploy the code. This can save you a few steps when you are just testing out changes.

Even if you have made lots of changes to the staging deployment, the commit that you push live will include them all, so you can do that manually once you are happy with how things look in staging.

Further Reading

- The tutorials from Git Client Tower, already mentioned in this article, are a great place to start if you are new to Git.
- A presentation from Liam Dempsey showing how to use the GitHub App to connect to Bitbucket
- Try Git from Code School
- The Git Workbook a self study guide to Git from Lorna Mitchell

GET SET UP FOR THE NEW YEAR

I love to start the New Year with a clean slate and improved processes. If you are still wrangling files with FTP then this is one thing you could tick off your list to save you time and energy in 2015. Post to the comments if you have suggestions of tools or ideas for ways to enhance this type of set-up for those who have already taken the first steps.

ABOUT THE AUTHOR

Rachel Andrew is a Director of edgeofmyseat.com, a UK web development consultancy and creators of the small content management system, Perch. She is the author of a number of books, most recently The Profitable Side Project Handbook and CSS3 Layout Modules, and is a regular columnist for A List Apart.

When not writing about business and technology on her blog at rachelandrew.co.uk or speaking at conferences, you will usually find Rachel running up and down one of the giant hills in Bristol.

5. What Is Vagrant and Why Should I Care?

Darren Beale 24ways.org/201405

If you run a web server, a database server and your scripting language(s) of choice on your main machine and you have not yet switched to using virtualisation in your workflow then this essay may be of some value to you.

I know you exist because I bump into you daily: freelancers coming in to work on our projects; internet friends complaining about reinstalling a development environment because of an operating system upgrade; fellow agency owners who struggle to brief external help when getting a particular project up and running; or even hardcore back-end developers who "don't do ops" and prefer to run their development stack of choice locally.

There are many perfectly reasonable arguments as to why you may not have already made the switch, from being simply too busy, all the way through to a distrust of the new. I'll admit that there are many new technologies or workflows that I hear of daily and instantly disregard

because I have tool overload, that feeling I get when I hear about a new shiny thing and think "Well, what I do now works – I'll leave it for others to play with." If that's you when it comes to Vagrant then I hope you'll hear me out. The business case is compelling enough for you to make that switch; as a bonus it's also really easy to get going.

In this article we'll start off by going through the high level, the tools available and how it all fits together. Then we'll touch on the justification for making the switch, providing a few use cases that might resonate with you. Finally, I'll provide a very simple example that you can follow to get yourself up and running.

WHAT?

You already know what virtualisation is. You use the ability to run an operating system within another operating system every day. Whether that's Parallels or VMware on your laptop or similar server-based tools that drive the 'cloud', squeezing lots of machines on to physical hardware and making it really easy to copy servers and even clusters of servers from one place to another. It's an amazing technology which has changed the face of the internet over the past fifteen years.

Simply put, Vagrant makes it really easy to work with virtual machines. According to the Vagrant docs:

> If you're a designer, Vagrant will automatically set everything up that is required for that web app in order for you to focus on doing what you do best: design. Once a developer configures Vagrant, you don't need to worry about how to get that app running ever again. No more bothering other developers to help you fix your environment so you can test designs. Just check out the code, `vagrant up`, and start designing.

While I'm not sure I agree with the implication that all designers would get others to do the configuring, I think you'll agree that the "Just check out the code… and start designing" premise is very compelling.

You don't need Vagrant to develop your web applications on virtual machines. All you need is a virtualisation software package, something like VMware Workstation or VirtualBox, and some code. Download the half-gigabyte operating system image that you want and install it. Then download and configure the stack you'll be working with: let's say Apache, MySQL, PHP. Then install some libraries, CuRL and ImageMagick maybe, and finally configure the ability to easily copy files from your machine to the new virtual one, something like Samba, or install an FTP server. Once this is all done, copy the code over, import the database, configure Apache's virtual host, restart and cross your fingers.

If you're a bit weird like me then the above is pretty easy to do and secretly quite fun. Indeed, the amount of traffic to one of my more popular blog posts proves that a lot of people have been building themselves development servers from scratch for some time (or at least trying to anyway), whether that's on virtual or physical hardware.

Or you could use Vagrant. It allows you, or someone else, to specify in plain text how the machine's virtual hardware should be configured and what should be installed on it. It also makes it insanely easy to get the code on the server. You check out your project, type `vagrant up` and start work.

WHY?

It's worth labouring the point that Vagrant makes it really easy; I mean look-no-tangle-of-wires-or-using-vim-and-loads-of-annoying-command-line-stuff easy to run a development environment.

That's all well and good, I hear you say, but there's a steep learning curve, an overhead to switch. You're busy and this all sounds great but you need to get on; you've got a career to build or a business to run and you don't have time to learn new stuff right now.

In short, what's the business case?

The business case involves saved time, a very low barrier to entry and the ability to give the exact same environment to somebody else.

Getting your first development virtual machine running will take minutes, not counting download time. Seriously, use pre-built Vagrant files and provisioners (we'll touch on this below) and you can start developing immediately.

Once you've finished developing you can check in your changes, ask a colleague or freelancer to check them out, and then they run the code on the exact same machine – even if they are on the other side of the world and regardless of whether they are on Windows, Linux or Apple OS X.

The configuration to build the machine isn't a huge binary disk image that'll take ages to download from Git; it's two small text files *that can be version controlled too*, so you can see any changes made to the config and roll back if needed.

No more 'It works for me' reports; no 'Oh, I was using PHP 5.3.3, not PHP 5.3.11' – you're both working on exact same copies of the development environment. With a tested and verified provisioning file you'll have the confidence that when you brief your next freelancer in to your team there won't be that painful to and fro of getting

the system up and running, where you're on a Skype call and they are uttering the immortal words, 'It still doesn't work'. You know it works because you can run it too.

This portability becomes even more important when you're working on larger sites and systems. Need a load balancer? Multiple front-end servers and a clustered database back-end? No problem. Add each server into the same Vagrant file and a single command will build all of them. As you'll know if you work on larger, business critical systems, keeping the operating systems in sync is a real problem: one server with a slightly different library causing sporadic and hard to trace issues is a genuine time black hole. Well, the good news is that you can use the same provisioning files to keep test and production machines in sync using your current build workflow.

Let's also not forget the most simple use case: a single developer with multiple websites running on a single machine. If that's you and you switch to using Vagrant-managed virtual machines then the next time you upgrade your operating system or do a fresh install there's no chance that things will all stop working. The server config is all tucked away in version control with your code. Just pull it down and carry on coding.

OK, GOT IT. SHOW ME ALREADY

If you want to try this out you'll need to install the latest VirtualBox and Vagrant for your platform. If you already have VMware Workstation or another supported virtualisation package installed you can use that instead but you may need to tweak my Vagrant file below. Depending on your operating system, a reboot might also be wise.

Note: the commands below were executed on my MacBook, but should also work on Windows and Linux. If you're using Windows make sure to run the command prompt as Administrator or it'll fall over when trying to update the hosts file.

As a quick sanity check let's just make sure that we have the `vagrant` command in our path, so fire up a terminal and check the version number:

```
$ vagrant -v
Vagrant 1.6.5
```

We've one final thing to install and that's the vagrant-hostsupdater plugin. Once again, in your terminal:

```
$ vagrant plugin install vagrant-hostsupdater
Installing the 'vagrant-hostsupdater' plugin. This can
take a few minutes...
Installed the plugin 'vagrant-hostsupdater (0.0.11)'!
```

Hopefully that wasn't too painful for you.

There are two things that you need to manage a virtual machine with Vagrant:

1. a Vagrant file: this tells Vagrant what hardware to spin up
2. a provisioning file: this tells Vagrant what to do on the machine

To save you copying and pasting I've supplied you with a simple example (ZIP) containing both of these. Unzip it somewhere sensible and in your terminal make sure you are inside the Vagrant folder:

```
$ cd where/you/placed/it/24ways

$ ls -l
-rw-r--r--@  1 bealers  staff  11055  9 Nov 09:16
bealers-24ways.md
-rw-r--r--@  1 bealers  staff  118152 9 Nov 10:08
it-works.png
drwxr-xr-x   5 bealers  staff  170    8 Nov 22:54 vagrant

$ cd vagrant/

$ ls -l
-rw-r--r--@ 1 bealers  staff  1661  8 Nov 21:50
Vagrantfile
-rwxr-xr-x@ 1 bealers  staff  3841  9 Nov 08:00
provision.sh
```

The Vagrant file tells Vagrant how to configure the virtual hardware of your development machine. Skipping over some of the finer details, here's what's in that Vagrant file:

```
www.vm.box = "ubuntu/trusty64"
```

Use Ubuntu 14.04 for the VM's OS. Vagrant will only download this once. If another project uses the same OS, Vagrant will use a cached version.

```
www.vm.hostname = "bealers-24ways.dev"
```

Set the machine's hostname. If, like us, you're using the vagrant-hostsupdater plugin, this will also get added to your hosts file, pointing to the virtual machine's IP address.

```
www.vm.provider :virtualbox do |vb|
    vb.customize ["modifyvm", :id, "--cpus", "2" ]
end
```

Here's an example of configuring the virtual machine's hardware on the fly. In this case we want two virtual processors.

Note: this is specific for the VirtualBox provider, but you could also have a section for VMware or other supported virtualisation software.

```
www.vm.network "private_network", ip: "192.168.13.37"
```

This specifies that we want a private networking link between your computer and the virtual machine. It's probably best to use a reserved private subnet like 192.168.0.0/16 or 10.0.0.0/8

```
www.vm.synced_folder "../", "/var/www/24ways",
        owner: "www-data", group: "www-data"
```

A particularly handy bit of Vagrant magic. This maps your local *24ways* parent folder to */var/www/24ways* on the virtual machine. This means the virtual machine already has direct access to your code and so do you. There's no messy copying or synchronisation – just edit your files and immediately run them on the server.

```
www.vm.provision :shell, :path => "provision.sh"
```

This is where we specify the provisioner, the script that will be executed on the machine.

If you open up the provisioner you'll see it's a bash script that does things like:

- install Apache, PHP, MySQL and related libraries
- configure the libraries: set permissions, enable logging
- create a database and grant some access rights
- set up some code for us to develop on; in this case, fire up a vanilla WordPress installation

To get this all up and running you simply need to run Vagrant from within the *vagrant* folder:

```
$ vagrant up
```

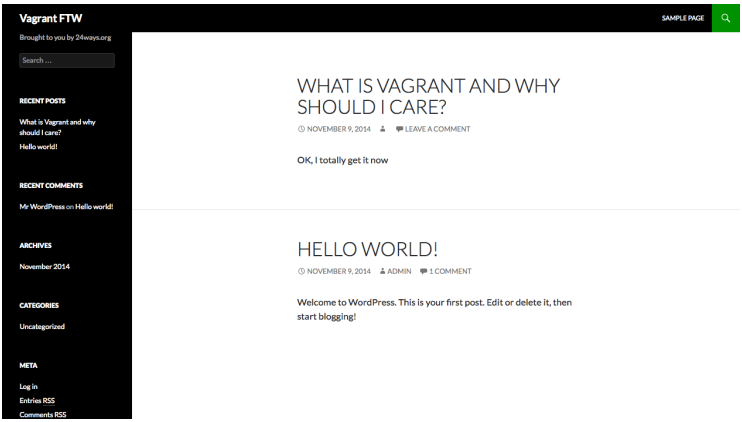
You should now get a Matrix-like stream of stuff shooting up the screen. If this is the first time Vagrant has used this particular operating system image – remember we've specified the latest version of Ubuntu – it'll download the disc image and cache it for future reuse. Then all the packages are downloaded and installed and finally all our configuration steps occur incluing the download and configuration of WordPress.

Halfway through proceedings it's likely that the process will halt at a prompt something like this:

```
==> www: adding to (/etc/hosts) : 192.168.13.37
bealers-24ways.dev  # VAGRANT:
2dbfbced1b1e79d2a0942728a0a57ece (www) /
899bd80d-4251-4f6f-91a0-d30f2d9918cc
Password:
```

You need to enter your password to give vagrant sudo rights to add the IP address and hostname mapping to your local hosts file.

Once finished, fire up your browser and go to http://bealers-24ways.dev. You should see a default WordPress installation. The username for *wp-admin* is *admin* and the password is *24ways*.



If you take a look at your local filesystem the *24ways* folder should now look like:

```
$ cd ../

$ ls -l

-rw-r--r--@  1 bealers  staff    13074  9 Nov 10:14
bealers-24ways.md
drwxr-xr-x  21 bealers  staff      714  9 Nov 10:06 code
drwxr-xr-x   3 bealers  staff      102  9 Nov 10:06 etc
-rw-r--r--@  1 bealers  staff   118152  9 Nov 10:08
it-works.png
drwxr-xr-x   5 bealers  staff      170  9 Nov 10:03
vagrant
-rwxr-xr-x   1 bealers  staff  1315849  9 Nov 10:06
wp-cli

$ cd vagrant/

$ ls -l
```

```
-rw-r--r--@ 1 bealers  staff  1661  9 Nov 09:41
Vagrantfile
-rwxr-xr-x@ 1 bealers  staff  3836  9 Nov 10:06
provision.sh
```

The *code* folder contains all the WordPress files. You can edit these directly and refresh that page to see your changes instantly.

Staying in the *vagrant* folder, we'll now SSH to the machine and have a quick poke around.

```
$ vagrant ssh
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux
3.13.0-39-generic x86_64)

* Documentation:  https://help.ubuntu.com/

System information as of Sun Nov  9 10:03:38 UTC 2014

System load:  1.35              Processes:           102
Usage of /:   2.7% of 39.34GB   Users logged in:     0
Memory usage: 16%               IP address for eth0:
10.0.2.15
Swap usage:   0%

Graph this data and manage this system at:
https://landscape.canonical.com/

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

0 packages can be updated.
```

```
0 updates are security updates.

vagrant@bealers-24ways:~$
```

You're now logged in as the Vagrant user; if you want to become root this is easy:

```
vagrant@bealers-24ways:~$ sudo su -
root@bealers-24ways:~#
```

Or you could become the webserver user, which is a good idea if you're editing the web files directly on the server:

```
root@bealers-24ways:~# su - www-data
www-data@bealers-24ways:~$
```

www-data's home directory is */var/www* so we should be able to see our magically mapped files:

```
www-data@bealers-24ways:~$ ls -l
total 4
drwxr-xr-x 1 www-data www-data  306 Nov  9 10:09 24ways
drwxr-xr-x 2 root     root     4096 Nov  9 10:05 html

www-data@bealers-24ways:~$ cd 24ways/

www-data@bealers-24ways:~/24ways$ ls -l
total 1420
-rw-r--r-- 1 www-data www-data   13682 Nov  9 10:19
bealers-24ways.md
drwxr-xr-x 1 www-data www-data     714 Nov  9 10:06 code
drwxr-xr-x 1 www-data www-data     102 Nov  9 10:06 etc
-rw-r--r-- 1 www-data www-data  118152 Nov  9 10:08
it-works.png
```

```
drwxr-xr-x 1 www-data www-data     170 Nov  9 10:03
vagrant
-rwxr-xr-x 1 www-data www-data 1315849 Nov  9 10:06
wp-cli
```

We can also see some of our bespoke configurations:

```
www-data@bealers-24ways:~/24ways$ cat /etc/php5/
mods-available/siftware.ini
upload_max_filesize = 15M
log_errors = On
display_errors = On
display_startup_errors = On
error_log = /var/log/apache2/php.log
memory_limit = 1024M
date.timezone = Europe/London

www-data@bealers-24ways:~/24ways$ ls -l /etc/apache2/
sites-enabled/
total 0
lrwxrwxrwx 1 root root 43 Nov  9 10:06
bealers-24ways.dev.conf -> /var/www/24ways/etc/
bealers-24ways.dev.conf
```

If you want to leave the server, simply type **Ctrl+D** a few times and you'll be back where you started.

```
www-data@bealers-24ways:~/24ways$ logout
root@bealers-24ways:~# logout
vagrant@bealers-24ways:~$ logout
Connection to 127.0.0.1 closed.
$
```

You can now `halt` the machine:

```
$ vagrant halt
==> www: Attempting graceful shutdown of VM...
==> www: Removing hosts
```

BONUS LEVEL

The example I've provided isn't very realistic. In the real world I'd expect the Vagrant file and provisioner to be included with the project and for it not to create the directory structure, which should already exist in your project. The same goes for the Apache VirtualHost file. You'll also probably have a default SQL script to populate the database.

As you work with Vagrant you might start to find bash provisioning to be quite limiting, especially if you are working on larger projects which use more than one server. In that case I would suggest you take a look at Ansible, Puppet or Chef. We use Ansible because we like YAML but they all do the same sort of thing. The main benefit is being able to use the same Vagrant provisioning scripts to also provision test, staging and production environments using your build workflows.

Having to supply a password so the hosts file can be updated gets annoying very quicky so you can give Vagrant sudo rights:

```
$ sudo visudo
```

Add these lines to the bottom (**Shift+G** then **i** then **Ctrl+V** then **Esc** then **:wq**)

```
Cmnd_Alias VAGRANT_HOSTS_ADD = /bin/sh -c echo "*" >>
/etc/hosts
Cmnd_Alias VAGRANT_HOSTS_REMOVE = /usr/bin/sed -i -e /*/
d /etc/hosts
%staff ALL=(root) NOPASSWD: VAGRANT_HOSTS_ADD,
VAGRANT_HOSTS_REMOVE
```

Vagrant caches the operating system images that you download but it'll download the installed software packages every time. You can get around this by using a plugin like vagrant-cachier or, if you're really keen, maintain local Apt repositories (or whatever the equivalent is for your server architecture).

At some point you might start getting a large number of virtual machines running on your poor hardware all at the same time, especially if you're switching between projects a lot and each of those projects use lots of servers. We're just getting to that stage now, so are considering a medium-term move to a containerised option like Docker, which seems to be maturing now.

If you are keen not to use any command line tools whatsoever and you're on OS X then you could check out Vagrant Manager as it looks quite shiny.

Finally, there are a huge amount of resources to give you pre-built Vagrant machines from the likes of VVV for Wordpress, something similar for Perch, PuPHPet for generating various configurations, and a long list of pre-built operating systems at VagrantBox.es.

WRAPPING UP

Hopefully you can now see why it might be worthwhile to add Vagrant to your development workflow. Whether you're an agency drafting in freelancers or a one-person band running lots of sites on your laptop using MAMP or something similar.

Vagrant makes it easy to launch exact copies of the same machine in a repeatable and version controlled way. The learning curve isn't too steep and, once configured, you can forget about it and focus on getting your work done.

ABOUT THE AUTHOR

Darren Beale is a Linux sysadmin and backend developer in Shropshire who these days spends most of his time in cashflow projections and talking to the clients of his agency, Siftware.

His dream is to retire to the woods and make furniture all day but until he can find a well paying consulting gig doing this I'm afraid we're stuck with him.

Darren tweets and blogs about things like business, self-funded product development and chainsaws.

6. Don’t Push Through the Pain

Carolyn Wood 24ways.org/201406

In 2004, I lost my web career. In a single day, it was gone. I was in too much pain to use a keyboard, a Wacom tablet (I couldn’t even click the pen), or a trackball. Switching my mouse to use my left (non-dominant) hand only helped a bit; then that hand went, too. I tried all the easy-to-find equipment out there, except for expensive gizmos with foot pedals. I had tingling in my fingers—which, when I was away from the computer, would rhythmically move as if some other being controlled them. I worried about Parkinson’s because the movements were so dramatic. Pen on paper was painful. Finally, I discovered one day that I couldn’t even turn a doorknob.

The only highlight was that I couldn't dust, scrub, or vacuum. We were forced to hire someone to come in once a week for an hour to whip through the house. You can imagine my disappointment.

My injuries had gradually slithered into my life without notice. I'd occasionally have sore elbows, or my wrist might ache for a day, or my shoulders feel tight. But nothing to keyboard home about. That's the critical bit of news. One day, you're pretty fine. The next day, you don't have your job—or any job that requires the use of your hands and wrists.

I had to walk away from the computer for over four months—and partially for several months more. That's right: no income. If I hadn't found a gifted massage therapist, the right book of stretches, the equipment I should have been using all along, and learned how to pay attention to my body—even just a little bit more—I quite possibly wouldn't be writing this article today. I wouldn't be writing anything, anywhere.

Most of us have heard of (and even claimed to have read all of) Mihaly Csikszentmihalyi, author of *Flow: The Psychology of Optimal Experience*, who describes the state of flow—the place our minds go when we are fully engaged and in our element. This lovely state of highly focused activity is deeply satisfying, often creative, and quite familiar to many of us on the web who just can't quit

until the copy sings or the code is untangled or we get our highest score yet in Angry Birds. Our minds may enter that flow, but too often as our brains take flight, all else recedes. And we leave something very important behind.

Our bodies.

My body wasn't made to make the same minute movements thousands of times a day, most days of the year, for decades, and neither was yours. The wear and tear sneaks up on you, especially if you're the obsessive perfectionist that we all pretend not to be. Oh? You're not obsessed? I wasn't like this *all* the time, but I remember sitting across from my husband, eating dinner, and I didn't hear a word he said. I'd left my brain upstairs in my office, where it was wrestling in a death match with the box model or, God help us all, IE 5.2. I was a writer, too, and I was having my first inkling that I was a content strategist. Work was exciting. I could sit up late, in the flow, fingers flying at warp speed. I could sit until those wretched birds outside mocked me with their damn, cheerful "Hurray, it's morning!" songs. Suddenly, while, say, washing dishes, the one magical phrase that captured the essence of a voice or idea would pop up, and I would have mowed down small animals and toddlers to get to my computer and hammer out that website or article, to capture that thought before it escaped. Note my use of the word *hammer*. Sound at all familiar?

But where was my body during my work? Jaw jutting forward to see the screen, feet oddly positioned—and then left in place like chunks of marble—back unsupported, fingers pounding the keys, wrists and arms permanently twisted in unnatural angles that we thought were natural. And clicking. Clicking, clicking, clicking that mouse. Thumbing tiny keyboards on phones. A lethal little gesture for tiny little tendons. Though I was fine from, say 1997 to 2004, by the end of 2004 this behavior culminated in disaster. I had repetitive stress injuries, aka repetitive motion injuries. As the Apple site says, "A brief exposure to these conditions would not cause harm. But a prolonged exposure may, in some people, result in reduced ability to function." I'll say.

I frantically turned to people on lists and forums. "Try a track ball." Already did that. "Try a tablet." Worse. One person wrote, "I still come here once in a while and can type a couple sentences, but I've permanently got thoracic outlet syndrome and I'll never work again." Oh, beauteous web, oh, long-distance friends, farewell.

THE WRIST BONE'S CONNECTED TO THE BRAIN BONE

That variation on the old song tells part of the story. Most people (and many of their physicians) believe that tingling fingers and aching wrists MUST be carpel tunnel syndrome. Nope. If your neck juts forward, it tenses and

stays tense the entire time you work in that position. Remember how your muscles felt after holding a landline phone with your neck tilted to one side for a long client meeting? Regrettable. Tensing your shoulders because your chair's not designed properly puts you at risk for thoracic outlet syndrome, a career-killer if ever there was one. The nerves and tendons in your neck and shoulder refer down your arms, and muscles swell around nerves, causing pain and dysfunction. Your elbows have a tendon that is especially vulnerable to repetitive movements (think tennis elbow). Your wrists are performing something akin to a circus act with one thousand shows a day.

So, all the fine tendons and ligaments in your fingers have problems that may not start at your wrists at all. Though some people truly do have carpal tunnel syndrome, my finger and wrist problems weren't solved by heavily massaging my fingers (though, that was helpful, too) or my wrists. They were fixed by work on my neck, upper back, shoulders, arms, and elbows. This explains why many people have surgery for carpal tunnel syndrome and just months later say, "What?! How can I possibly have it again? I had an operation!" Well, fellow buckaroo, you may never have had carpel tunnel syndrome. You may have had—or perhaps will have—one long disaster area from your neck to your fingertips.

HOW TO CRAWL BACK

Before trying extreme measures, you may be able to function again even if you feel hopeless. I managed to heal, and so have others, but I'll always be at risk.

As Jen Simmons, of The Web Ahead podcast and other projects told me, "It took a long time to injure myself. It took a long time to get back to where I was. My right arm between my elbow and wrist would start aching intermittently. Eventually, my arm even ached at night. I started each day with yesterday's pain." Simple measures, used consistently, helped her back.

1. Massage therapy

I don't remember what the rest of the world is like, but in Portland, Oregon, we have more than one massage therapy college. (Of course we do.) I saw a former teacher at the most respected school. This is not your "It was all so soothing. Why, I fell asleep!" massage. This is "Holy crap, he's grinding his elbow into my armpit!" massage therapy, with the emphasis on **therapy**. I owe him everything. Make sure you have someone who really knows what they're doing. Get many referrals. Try a question, "Does my psoas muscle affect my back?" If they can't answer it, flee. Regularly see the one you choose and after a while, depending on how injured you are, you may be able to taper off.

2. Change your equipment

You may need to be hands-on with several pieces of equipment before you find the ones that don't cause more pain. Many companies have restocking fees, charges to ship the equipment you want to return, and other retail atrocities. Always be sure to ask what the return policies are at any company before purchasing.

MICE

You may have more success than I did with equipment such as the Wacom tablet. Mine came with a pen, and it hurt to repetitively click it. Trackballs are another option but, for many, they are better at prevention than recovery. But let's get to the really effective stuff. One of the biggest sources of pain is using your mouse. One major reason is that your hand and wrist are in a perpetually unnatural position and you're also moving your arm quite a bit. Each time you move the mouse, it is placing stress on your neck, shoulders and arms, because you need to lift them slightly in order to move the mouse and you need to angle your wrist. You may also be too injured to use the trackpad all the time, and this mouse, the vertical mouse is a dandy preventative measure, too. Shaking up your patterns is a wise move. I have long fingers, not especially thin, yet the small size works best for me. (They have larger choices available.) What?! A sideways mouse? Yep. All the weight of your hand will be resting on it in the

handshake position. Your forearms aren't constantly twisting over hill and dale. You aren't using any muscles in your wrist or hand. They are relaxing. You'll adapt in a day, and oh, oh, what a relief it is.

KEYBOARDS

I really liked doing business with the people at Kinesis-Ergo. (I'm not affiliated with them in any way.) They have the vertical mouse and a number of keyboards. The one that felt the most natural to me, and, once again, it only takes a day to adapt, is the Freestyle2 for the Mac. They have several options. I kept the keyboard halves attached to each other at first, and then spread them apart a little more. I recommend choosing one that slants and can separate. You can adjust the angle. For a little extra, they'll make sure it's all set up and ready to go for you. I'm guessing that some Googling will find you similar equipment, wherever you live.

Warning: if you use the ergonomic keyboards, you may have fewer USB ports. The laptop will be too far away to see unless you find a satisfactory setup using a stand. This is the perfect excuse for purchasing a humongous display.

You may not look cool while jetting coast to coast in your skinny jeans and what appears to be the old-time orthopedic shoe version of computing gear. But once you

have rested and used many of these suggestions consistently, you may be able to use your laptop or other device in all its lovely sleekness during the trip.

OTHER DOOHICKIES

The Kinesis site and The Human Solution have a wide selection of ergonomic products: standing desks, ergonomically correct chairs, and, yes, even things with foot pedals. Explore!

3. Stop clicking, at least for a while

Use keyboard shortcuts, but use them slowly. This is not the time to show off your skillz. You'll be sort of like a recovering alcoholic, in that you'll be a recovering repetitive stress survivor for the rest of your life, once you really injure yourself. Always be vigilant. There's also a bit of software sold by The Human Solution and other places, and it was my salvation. It's called the McNib for Macs, and the Nib for PCs. (I've only used the McNib.) It's for click-free mousing. I found it tricky to use when writing markup and code, but you may become quite adept at it. A little rectangle pops up on your screen, you mouse over it and choose, let's say, "Double-click." Until you change that choice, if you mouse over a link or anything else, it will double-click it for you. All you do is glide your mouse around. Awkward for a day or two, but you'll pick it up quickly. Though you can use it all day for work, even if you

just use this for browsing LOLcats or Gary Vaynerchuk's YouTube videos, it will help you by giving your fingers a sweet break.

But here's the sad news. The developer who invented this died a few years ago. (Yes, I used to speak to him on the phone.) While it is for sale, it isn't compatible with Mac OS X Lion or anything subsequent. PowerPC strikes again. His site is still up. Demos for use with older software can be downloaded free at his old site, or at The Human Solution. Perhaps an enterprising developer can invent something that would provide this help, without interfering with patents. Rumor has it among ergonomic retailers (yes, I'm like a police dog sniffing my way to a criminal once I head down a trail) that his company was purchased by a company in China, with no update in sight.

4. Use built-in features

That little microphone icon that comes up alongside the keyboard on your iPhone allows you to speak your message instead of incessantly thumbing it. I believe it works in any program that uses the keyboard. It's not Siri. She's for other things, like having a personal relationship with an inanimate object. Apple even has a good section on ergonomics. You think I'm intense about this subject? To improve your repetitive stress, Apple doesn't want you

to use oral contraceptives, alcohol, or tobacco, to which I say, "Have as much sex, bacon, and chocolate as possible to make up for it."

Apple's info even has illustrations of things like a faucet dripping into what is labeled a bucket full of "TRAUMA." Sounds like upgrading to Yosemite, but I digress.

5. Take breaks

If it's a game or other non-essential activity, take a break for a month. Fine, now that I've called games non-essential, I suppose you'll all unfollow me on Twitter.

6. Whether you are sore or not, do stretches throughout the day

This is a big one. Really big. The best book on the subject of repetitive stress injuries is *Conquering Carpal Tunnel Syndrome and Other Repetitive Strain Injuries: A Self-Care Program* by Sharon J. Butler. Don't worry, most of it is illustrations. Pretend it's a graphic novel.

I'm notorious for never reading instructions, and who on earth reads the introduction of a book, unless they wrote it? I wrote a book a long time ago, and I bet my house, husband, and life savings that my own parents never read the intro. Well, I did read the intro to this book, and you should, too. Stretching correctly, in a way that doesn't further hurt you, that keeps you flexible if you aren't

injured, that actually heals you, calls for precision. Read and you'll see. The key is to stretch just until you start to feel the stretch, even if that's merely a tiny movement. Don't force anything past that point. Kindly nurse yourself back to health, or nurture your still-healthy body by stretching. Over the following days, weeks, months, you'll be moving well past that initial stretch point.

The book is brimming with examples. You only have to pick a few stretches, if this is too much to handle. Do it every single day. I can tell you some of the best ones for me, but it depends on the person. You'll also discover in Butler's book that areas that you think are the problem are sometimes actually adjacent to the muscle or tendon that is the source of the problem. Add a few stretches or two for that area, too.

But please follow the instructions in the introduction. If you overdo it, or perform some other crazy-ass hijinks, as I would be tempted to do, I am not responsible for your outcome. I give you fair warning that I am not a healthcare provider. I'm just telling you as a friend, an untrained one, at that, who has been through this experience.

7. Follow good habits

Develop habits like drinking lots of water (which helps with lactic acid buildup in muscles), looking away from the computer for twenty seconds every twenty to thirty

minutes, eating right, and probably doing everything else your mother told you to do. Maybe this is a good time to bring up flossing your teeth, and going outside to play instead of watching TV. As your mom would say, "It's a beautiful day outside, what are you kids doing in here?"

8. Speak instead of writing, if you can

Amber Simmons, who is very smart and funny, once tweeted in front of the whole world that, "@carywood is a Skype whore." I was always asking people on Twitter if we could Skype instead of using iChat or exchanging emails. (I prefer the audio version so I don't have to, you know, do something drastic like comb my hair.) Keyboarding is tough on hands, whether you notice it or not at the time, and when doing rapid-fire back-and-forthing with people, you tend to speed up your typing and not take any breaks. This is a hand-killer. Voice chats have made such a difference for me that I am still a rabid Skype whore. Wait, did I say that out loud?

Speak your text or emails, using Dragon Dictate or other software. In about 2005, accessibility and user experience design expert, Derek Featherstone, in Canada, and I, at home, chatted over the internet, each of us using a different voice-to-text program. The programs made so many mistakes communicating with each other that we began that sort of endless, tearful laughing that makes you think someone may need to call an ambulance. This

type of software has improved quite a bit over the years, thank goodness. Lack of accessibility of any kind isn't funny to Derek or me or to anyone who can't use the web without pain.

9. Watch your position

For example, if you lift up your arms to use the computer, or stare down at your laptop, you'll need to rearrange your equipment. The internet has a lot of information about ideal ergonomic work areas. Please use a keyboard drawer. Be sure to measure the height carefully so that even a tented keyboard, like the one I recommend, will fit. I also recommend getting the version of the Freestyle with palm supports. Just these two measures did much to help both Jen Simmons and me.

10. If you need to take anti-inflammatories, stop working

If you are all drugged up on ibuprofen, and pounding and clicking like mad, your body will not know when you are tired or injuring yourself. I don't recommend taking these while using your computing devices. Perhaps just take it at night, though I'm not a fan of that category of medications. Check with your healthcare provider. At least ibuprofen is an anti-inflammatory, which may help you. In contrast, acetaminophen (paracetamol) only makes your body think it's not in pain. Ice is great, as is

switching back and forth between ice and heat. But again, if you need ice and ibuprofen **you really need to take a major break**.

11. Don't forget the rest of your body

I've zeroed in on my personal area of knowledge and experience, but you may be setting yourself up for problems in other areas of your body. There's what is known to bad writers as "a veritable cornucopia" of information on the web about how to help the rest of your body. A wee bit of research on the web and you'll discover simple exercises and stretches for the rest of your potential catastrophic areas: your upper back, your lower back, your legs, ankles, and eyes. Do gentle stretches, three or four times a day, rather than powering your way through. Ease into new equipment such as standing desks. Stretch those newly challenged areas until your body adapts. Pay attention to your body, even though I too often forget mine.

12. Remember the children

Kids are using equipment to play highly addictive games or to explore amazing software, and if these call for repetitive motions, children are being set up for future injuries. They'll grab hold of something, as parents out there know, and play it 3,742 times. That afternoon. Perhaps by the time they are adults, everything will just

be holograms and mind-reading, but adult fingers and hands are used for most things in life, not just computing devices and phones with keyboards sized for baby chipmunks.

I'LL BE WATCHING YOU

Quickly now, while I (possibly) have your attention. Don't move a muscle. Is your neck tense? Are you unconsciously lifting your shoulders up? How long since you stopped staring at the screen? How bright is your screen? Are you slumping (c'mon now, 'fess up) and inviting sciatica problems? Do you have to turn your hands at an angle relative to your wrist in order to type? Uh-oh. That's a bad one. Your hands, wrists, and forearms should be one straight line while keyboarding. Future you is begging you to change your ways. Don't let your #ThrowbackThursday in 2020 say, "Here's a photo from when I used to be able to do so many wonderful things that I can't do now." And, whatever you do, don't try for even a nanosecond to push through the pain, or the next thing you know, you'll be an unpaid extra in The Expendables 7.

ABOUT THE AUTHOR

Carolyn Wood is a writer, editor, and content strategist who often starts her work with individuals and companies all the way back at “Who are you, really?” so they’ll speak with a clear, true voice. She was also Editor of Digital Web Magazine and then on the staff of A List Apart for three years as acquisitions editor. Carolyn also helped create The Manual, where she was Editor in Chief for the first year, and did the same for Codex, journal of typography, with John Boardley. She does experimental writing at inthespacebetween.com. She loves both laughing and taking a stand.

7. Collaborative Responsive Design Workflows

Sibylle Weber

24ways.org/201407

Much has been written about workflow and designer–developer collaboration in web design, but many teams still struggle with this issue; either with how to adapt their internal workflow, or how to communicate the need for best practices like mobile first and progressive enhancement to their teams and clients. Christmas seems like a good time to have another look at what doesn't work between us and how we can improve matters.

WHY IS IT SO DIFFICULT?

We're still beginning to understand responsive design workflows, acknowledging the need to move away from static design tools and towards best practices in development. It's not that we don't want to change – so why is it so difficult?

Changing the way we do something that has become routine is always problematic, even with small things, and the changes today's web environment requires from web design and development teams are anything but small.

Although developers also have a host of new skills to learn and things to consider, designers are probably the ones pushed furthest out of their comfort zones: as well as graphic design, a web designer today also needs an understanding of interaction design and ergonomics, because more and more websites are becoming tools rather than pages meant to be read like a book or magazine. In addition to that there are thousands of different devices and screen sizes on the market today that layout and interactions need to work on.

These aspects make it impossible to design in a static design tool, so beyond having to learn about new aspects of design, the designer has to either learn how to code or learn to work with a responsive design tool.

WHY DO IT

That alone is enough to leave anyone overwhelmed, as learning a new skill takes time and slows you down in a project – and on most projects time is in short supply. Yet we have to make time or fall behind in the industry as

others pitch better, interactive designs. For an efficient workflow, both designers and developers must familiarise themselves with new tools and techniques.

A designer has to be able to play with ideas, make small adjustments here and there, look at the result, go back to the settings and make further adjustments, and so on. You can only realistically do that if you are able to play with all the elements of a design, including interactivity, accessibility and responsiveness.

Figuring out the right breakpoints in a layout is one of the foremost reasons for designing in a responsive design tool. Even if you create layouts for three viewport sizes (i.e. smartphone, tablet and the most common desktop size), you'd only cover around 30% of visitors and you might miss problems like line breaks and padding at other viewport sizes.

Another advantage is consistency. In static design tools changes will not be applied across all your other layouts. A developer referring back to last week's comps might work with outdated metrics. Furthermore, you cannot easily test what impact changes might have on previously designed areas. In a dynamic design tool such changes will be applied to the entire design and allow you to test things in site areas you had already finished.

No static design tool allows you to do this, and having somebody else produce a mockup from your static designs or wireframes will duplicate work and is inefficient.

HOW TO DO IT

When working in a responsive design tool rather than in the browser, there is still the question of how and when to communicate with the developer. I have found that working with Sass in combination with a visual style guide is very efficient, but it does need careful planning: fundamental metrics for padding, margins and font sizes, but also design elements like sliders, forms, tabs, buttons and navigational elements, should be defined at the beginning of a project and used consistently across the site. Working with a grid can help you develop a consistent design language across your site.

Create a visual style guide that shows what the elements look like and how they behave across different screen sizes – and when interacted with. Put all metrics on paddings, margins, breakpoints, widths, colours and so on in a text document, ideally with names that your developer can use as Sass variables in the CSS. For example:

```
$padding-default-vertical: 1.5em;
```

Developers, too, need an efficient workflow to keep code maintainable and speed up the time needed for more complex interactions with an eye on accessibility and performance. CSS preprocessors like Sass allow you to work with variables and mixins for default rules, as well as style sheet partials for different site areas or design elements. Create your own boilerplate to use for your projects and then update your variables with the information from your designer for each individual project.

HOW TO GET BUY-IN

One obstacle when implementing responsive design, accessibility and content strategy is the logistics of learning new skills and iterating on your workflow. Another is how to sell it. You might expect everyone on a project (including the client) to want to design and develop the best website possible: ultimately, a great site will lead to more conversions. However, we often hear that people find it difficult to convince their teammates, bosses or clients to implement best practices.

Why is that? Well, I believe a lot of it is down to how we sell it. You will have experienced this yourself: some people you trust to know what they are talking about, and others you don't. Think about why you trust that first person but don't buy what the other one is telling you. It is likely because person A has a self-assured, calm and

assertive demeanour, while person B seems insecure and apologetic. To sell our ideas, we need to become person A! For a timid designer or developer suffering from imposter syndrome (like many of us do in this industry) that is a difficult task. So how can we become more confident in selling our expertise?

Write

We need to become experts. And I mean not just in writing great code or coming up with beautiful designs but at explaining why we're doing what we're doing. Why do you code this way or that? Why is this the best layout? Why does a website have to be accessible and responsive? Write about it. Putting your thoughts down on paper or screen is a really efficient way of getting your head around a topic and learning to make a case for something. You may even find that you come up with new ideas as you are writing, so you'll become a better designer or developer along the way.

Talk

Then, talk about it. Start out in front of your team, then do a lightning talk at a web event near you, then a longer talk or workshop. Having to talk about a topic is going to help you put into spoken words the argument that you've previously put together in writing. Writing comes more easily when you're starting out but we use a different

register when writing than talking and you need to learn how to speak your case. Do the talk a couple of times and after each talk make adjustments where you found it didn't work well. By this time, you are more than ready to make your case to the client. In fact, you've been ready since that first talk in front of your colleagues ;)

Pitch

Pitches used to be based on a presentation of static layouts for for three to five typical pages and three different designs. But if we want to sell interactivity, structure, usability, accessibility and responsiveness, we need to demonstrate these things and I believe that it can only do us good. I have seen a few pitches sitting in the client's chair and static layouts are always sort of dull. What makes a website a website is the fact that I can interact with it and smooth interactions or animations add that extra sparkle.

I can't claim personal experience for this one but I'd be bold and go for only one design. One demo page matching the client's corporate design but not any specific page for the final site. Include design elements like navigation, photography, typefaces, article layout (with **real** content), sliders, tabs, accordions, buttons, forms, tables (yes, tables) – everything you would include in a style tiles document, only interactive. Demonstrate how the elements behave when clicked, hovered and touched, and

how they change across different screen sizes. You may even want to demonstrate accessibility features like tabbed navigation and screen reader use.

Obviously, there are many approaches that will work in different situations but don't give up on finding a process that works for you and that ultimately allows you to build delightful, accessible, responsive user experiences for the web. Make time to try new tools and techniques and don't just work on them on the side – start using them on an actual project. It is only when we use a tool or process in the real world that we become true experts. Remember your driving lessons: once the instructor had explained how to operate the car, you were sent to practise driving on the road in actual traffic!

ABOUT THE AUTHOR

Sibylle Weber is a front-end developer and designer living in Munich. After being a marketing manager in e-commerce she moved to the production side of the web where she is an evangelist for usability, accessibility and responsive design.

8. Websites of Christmas Past, Present and Future

Josh Emerson

24ways.org/201408

THE WEBSITES OF CHRISTMAS PAST

The first website was created at CERN. It was launched on 20 December 1990 (just in time for Christmas!), and it still works today, after twenty-four years. Isn't that incredible?!

Why does this website still work after all this time? I can think of a few reasons.

First, the authors of this document chose HTML. Of course they couldn't have known back then the extent to which we would be creating documents in HTML, but HTML always had a lot going for it. It's built on top of plain text, which means it can be opened in any text editor, and it's pretty readable, even without any parsing.

Despite the fact that HTML has changed quite a lot over the past twenty-four years, extensions to the specification have always been implemented in a backwards-compatible manner. Reading through the 1992 W3C document HTML Tags, you'll see just how it has evolved. We still have `h1` – `h6` elements, but I'd not heard of the `<plaintext>` element before. Despite being deprecated since HTML2, it still works in several browsers. You can see it in action on my website.

As well as being written in HTML, there is no run-time compilation of code; the first website simply consists of HTML files transmitted over the web. Due to its lack of complexity, it stood a good chance of surviving in the turbulent World Wide Web.

That's all well and good for a simple, static website. But websites created today are increasingly interactive. Many require a login and provide experiences that are tailored to the individual user. This type of dynamic website requires code to be executed somewhere.

Traditionally, dynamic websites would execute such code on the server, and transmit a simple HTML file to the user. As far as the browser was concerned, this wasn't much different from the first website, as the additional complexity all happened before the document was sent to the browser.

Doing it all in the browser

In 2003, the first single page interface was created at slashdotslash.com. A single page interface or single page app is a website where the page is created in the browser via JavaScript. The benefit of this technique is that, after the initial page load, subsequent interactions can happen instantly, or very quickly, as they all happen in the browser.

When software runs on the client rather than the server, it is often referred to as a fat client. This means that the bulk of the processing happens on the client rather than the server (which can now be thin).

A fat client is preferred over a thin client because:

- It takes some processing requirements away from the server, thereby reducing the cost of servers (a thin server requires cheaper, or fewer servers).
- They can often continue working offline, provided no server communication is required to complete tasks after initial load.
- The latency of internet communications is bypassed after initial load, as interactions can appear near instantaneous when compared to waiting for a response from the server.

But there are also some big downsides, and these are often overlooked:

- They can't work without JavaScript. Obviously JavaScript is a requirement for any client-side code execution. And as the UK Government Digital Service discovered, 1.1% of their visitors did not receive JavaScript enhancements. Of that 1.1%, 81% had JavaScript enabled, but their browsers failed to execute it (possibly due to dropping the internet connection). If you care about 1.1% of your visitors, you should care about the non-JavaScript experience for your website.
- The browser needs to do all the processing. This means that the hardware it runs on needs to be fast. It also means that we require all clients to have largely the same capabilities and browser APIs.
- The initial payload is often much larger, and nothing will be rendered for the user until this payload has been fully downloaded and executed. If the connection drops at any point, or the code fails to execute owing to a bug, we're left with the non-JavaScript experience.
- They are not easily indexed as every crawler now needs to run JavaScript just to receive the content of the website.

These are not merely edge case issues to shirk off. The first three issues will affect some of your visitors; the fourth affects everyone, including you.

What problem are we trying to solve?

So what can be done to address these issues? Whereas fat clients solve some inherent issues with the web, they seem to create as many problems. When attempting to resolve any issue, it's always good to try to uncover the original problem and work forwards from there. One of the best ways to frame a problem is as a user story. A user story considers the who, what and why of a need. Here's a template:

> As a {who} I want {what} so that {why}

I haven't got a specific project in mind, so let's refer to the who as *user*. Here's one that could explain the use of thick clients.

> As a user I want the site to respond to my actions quickly so that I get immediate feedback when I do something.

This user story could probably apply to a great number of websites, but so could this:

> As a user I want to get to the content quickly, so that I don't have to wait too long to find out what the site is all about or get the content I need.

A better solution

How can we balance both these user needs? How can we have a website that loads fast, and also reacts fast? The solution is to have a thick server, that serves the complete document, and then a thick client, that manages subsequent actions and replaces parts of the page. What we're talking about here is simply progressive enhancement, but from the user's perspective.

The initial payload contains the entire document. At this point, all interactions would happen in a traditional way using links or form elements. Then, once we've downloaded the JavaScript (asynchronously, after load) we can *enhance* the experience with JavaScript interactions. If for whatever reason our JavaScript fails to download or execute, it's no biggie – we've already got a fully functioning website. If an API that we need isn't available in this browser, it's not a problem. We just fall back to the basic experience.

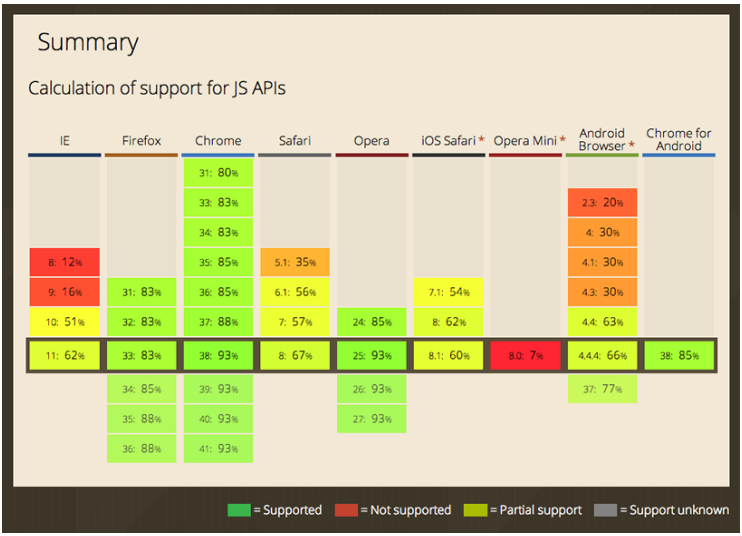
This second point, of having some minimum requirement for an enhanced experience, is often referred to as *cutting the mustard*, first used in this sense by the BBC News team. Essentially it's an `if` statement like this:

```
if('querySelector' in document
 && 'localStorage' in window
 && 'addEventListener' in window) {
 // bootstrap the JavaScript application
 }
```

This code states that the browser must support the following methods before downloading and executing the JavaScript:

- `document.querySelector` (can it find elements by CSS selectors)
- `window.localStorage` (can it store strings)
- `window.addEventListener` (can it bind to events in a standards-compliant way)

These three properties are what the BBC News team decided to test for, as they are present in their website's JavaScript. Each website will have its own requirements. The last method, `window.addEventListener` is in interesting one. Although it's simple to bind to events on IE8 and earlier, these browsers have very inconsistent support for standards. Making any JavaScript-heavy website work on IE8 and earlier is a painful exercise, and comes at a cost to all users on other browsers, as they'll download unnecessary code to patch support for IE.



JavaScript API support by browser.

I discovered that IE8 supports 12% of the current JavaScript APIs, while IE9 supports 16%, and IE10 51%. It seems, then, that IE10 could be the earliest version of IE that I'd like to develop JavaScript for. That doesn't mean that users on browsers earlier than 10 can't use the website. On the contrary, they get the core experience, and because it's just HTML and CSS, it's much more likely to be bug-free, and could even provide a better experience than trying to run JavaScript in their browser. They receive the thin client experience.

By reducing the number of platforms that our enhanced JavaScript version supports, we can better focus our efforts on those platforms and offer an even greater

experience to those users. But we can only do that if we use progressive enhancement. Otherwise our website would be completely broken for all other users.

So what we have is a thick server, capable of serving the entire website to our users, complete with all core functionality needed for our users to complete their tasks; and we have a thick client on supported browsers, which can bring an even greater experience to those users.

This is all transparent to users. They may notice that the website seems snappier on the new iPhone they received for Christmas than on the Windows 7 machine they got five years ago, but then they probably expected it to be faster on their iPhone anyway.

Isn't this just more work?

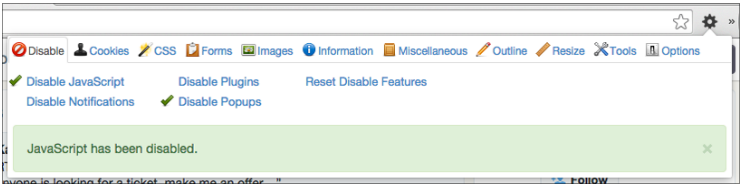
It's true that making a thick server *and* a thick client is more work than just making one or the other. But there are some big advantages:

- The website works for everyone.
- You can decide when users get the enhanced experience.
- You can enhance features in an iterative (or agile) manner.
- When the website breaks, it doesn't break down.

- The more you practise this approach, the quicker you will become.

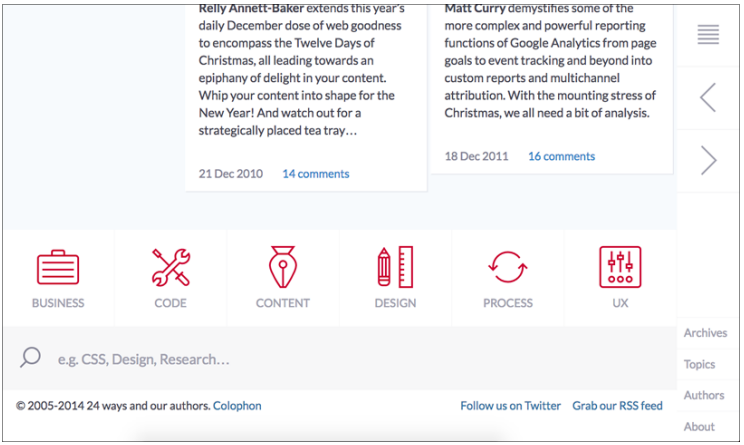
THE WEBSITES OF CHRISTMAS PRESENT

The best way to discover websites using this technique of progressive enhancement is to disable JavaScript and see if the website breaks. I use the Web Developer extension, which is available for Chrome and Firefox. It lets me quickly disable JavaScript.

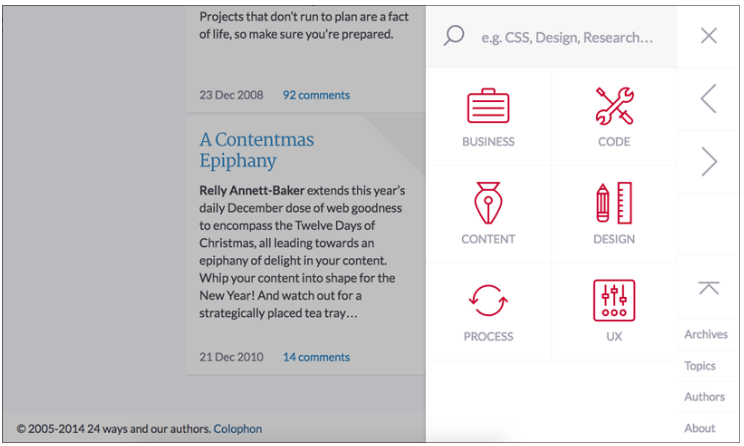


Web Developer extension.

24 ways works with and without JavaScript. Try using the menu icon to view the navigation. Without JavaScript, it's a jump link to the bottom of the page, but with JavaScript, the menu slides in from the right.



24 ways navigation with JavaScript disabled.



24 ways navigation with working JavaScript.

Google search will also work without JavaScript. You won't get instant search results or any prerendering, because those are enhancements.

For a more app-like example, try using Twitter. Without JavaScript, it still works, and looks nearly identical. But when you load JavaScript, links open in modal windows and all pages are navigated much quicker, as only the content that has changed is loaded. You can read about how they achieved this in Twitter's blog posts Improving performance on twitter.com and Implementing pushState for twitter.com.

Unfortunately Facebook doesn't use progressive enhancement, which not only means that the website doesn't work without JavaScript, but it takes longer to load. I tested it on WebPagetest and if you compare the load times of Twitter and Facebook, you'll notice that, despite putting similar content on the page, Facebook takes two and a half times longer to render the core content on the page.

Facebook takes two and a half times longer to load than Twitter.

WEBSITES OF CHRISTMAS YET TO COME

Every project is different, and making a website that enjoys a long life, or serves a larger number of users may or may not be a high priority. But I hope I've convinced

you that it certainly is possible to look to the past and future simultaneously, and that there can be significant advantages to doing so.

ABOUT THE AUTHOR

Josh Emerson is a front end developer working at Mendeley. He cares about progressive enhancement and enjoys challenging assumptions about user needs and device capabilities.

9. Responsive Enhancement

Jeremy Keith 24ways.org/201409

24 ways has been going strong for ten years. That's an aeon in internet timescales. Just think of all the changes we've seen in that time: the rise of Ajax, the explosion of mobile devices, the unrecognisably changed landscape of front-end tooling.

Tools and technologies come and go, but one thing has remained constant for me over the past decade: progressive enhancement.

Progressive enhancement isn't a technology. It's more like a way of thinking. Instead of thinking about the specifics of how a finished website might look, progressive enhancement encourages you to think about the fundamental meaning of what the website is providing. So instead of thinking of a website in terms of its ideal state in a modern browser on a nice widescreen device, progressive enhancement allows you to think about the core functionality in a more abstract way.

Once you've figured out what the core functionality is – adding an item to a shopping cart, posting a message, sharing a photo – then you can enable that functionality in the simplest possible way. That usually means starting with good old-fashioned HTML. Links and forms are often all you need. Then, once you have the core functionality working in a basic way, you can start to enhance to make a progressively better experience for more modern browsers.

The advantage of working this way isn't just that your site will work in older browsers (albeit in a rudimentary way). It also ensures that if anything goes wrong in a modern browser, it won't be catastrophic.

There's a common misconception that progressive enhancement means that you'll spend your time dealing with older browsers, but in fact the opposite is true. Putting the basic functionality into place doesn't take very long at all. And once you've done that, you're free to spend all your time experimenting with the latest and greatest browser technologies, secure in the knowledge that even if they aren't universally supported yet, that's OK: you've already got your fallback in place.

The key to thinking about web development this way is realising that there isn't one final interface – there could be many, slightly different interfaces depending on the

properties and capabilities of any particular user agent at any particular moment. And that's OK. Websites do not need to look the same in every browser.

Once you truly accept that, it's an immensely liberating idea. Instead of spending your time trying to make websites look the same in wildly varying browsers, you can spend your time making sure that the core functionality of what you build works everywhere, while providing the best possible experience for more capable browsers.

Allow me to demonstrate with a simple example: navigation.

STEP ONE: CORE FUNCTIONALITY

Let's say we have a straightforward website about the twelve days of Christmas, with a page for each day. The core functionality is pretty clear:

1. To read about any particular day.
2. To browse from day to day.

The first is easily satisfied by marking up the text with headings, paragraphs and all the usual structural HTML elements. The second is satisfied by providing a list of good ol' hyperlinks.

Now where's the best place to position this navigation list? Personally, I'm a big fan of the jump-to-footer pattern. This puts the content first and the navigation second. At the top of the page there's a link with an `href` attribute pointing to the fragment identifier for the navigation.

```
<body>
  <main role="main" id="top">
    <a href="#menu" class="control">Menu</a>
    ...
  </main>
  <nav role="navigation" id="menu">
    ...
    <a href="#top" class="control">Dismiss</a>
  </nav>
</body>
```

See the footer-anchor pattern in action.

Because it's nothing more than a hyperlink, this works in just about every browser since the dawn of the web. Following hyperlinks is what web browsers were made to do (hence the name).

STEP TWO: LAYOUT AS AN ENHANCEMENT

The footer-anchor pattern is a particularly neat solution on small-screen devices, like mobile phones. Once more screen real estate is available, I can use the magic of CSS

to reposition the navigation above the content. I could use `position: absolute`, flexbox or, in this case, `display: table`.

```
@media all and (min-width: 35em) {
  .control {
    display: none;
  }
  body {
    display: table;
  }
  [role="navigation"] {
    display: table-caption;
    columns: 6 15em;
  }
}
```

See the styles for wider screens in action

STEP THREE: ENHANCE!

Right. At this point I'm providing core functionality to everyone, and I've got nice responsive styles for wider screens. I could stop here, but the real advantage of progressive enhancement is that I don't have to. From here on, I can go crazy adding all sorts of fancy enhancements for modern browsers, without having to worry about providing a fallback for older browsers – the fallback is already in place.

What I'd really like is to provide a swish off-canvas pattern for small-screen devices. Here's my plan:

1. Position the navigation under the main content.
2. Listen out for the `.control` links being activated and intercept that action.
3. When those links are activated, toggle a class of `.active` on the body.
4. If the `.active` class exists, slide the content out to reveal the navigation.

Here's the CSS for positioning the content and navigation:

```
@media all and (max-width: 35em) {
  [role="main"] {
    transition: all .25s;
    width: 100%;
    position: absolute;
    z-index: 2;
    top: 0;
    right: 0;
  }
  [role="navigation"] {
    width: 75%;
    position: absolute;
    z-index: 1;
    top: 0;
    right: 0;
  }
  .active [role="main"] {
```

```
    transform: translateX(-75%);
  }
}
```

In my JavaScript, I'm going to listen out for any clicks on the `.control` links and toggle the `.active` class on the body accordingly:

```
(function (win, doc) {
  'use strict';
  var linkclass = 'control',
    activeclass = 'active',
    toggleClassName = function (element, toggleClass) {
      var reg = new RegExp('(s|^)' + toggleClass +
'(s|$)');
      if (!element.className.match(reg)) {
        element.className += ' ' + toggleClass;
      } else {
        element.className =
element.className.replace(reg, '');
      }
    },
    navListener = function (ev) {
      ev = ev || win.event;
      var target = ev.target || ev.srcElement;
      if (target.className.indexOf(linkclass) !== -1) {
        ev.preventDefault();
        toggleClassName(doc.body, activeclass);
      }
    };
  doc.addEventListener('click', navListener, false);
}(this, this.document));
```

I'm all set, right? Not so fast!

Cutting the mustard

I've made the assumption that `addEventListener` will be available in my JavaScript. That isn't a safe assumption. That's because JavaScript – unlike HTML or CSS – isn't fault-tolerant. If you use an HTML element or attribute that a browser doesn't understand, or if you use a CSS selector, property or value that a browser doesn't understand, it's no big deal. The browser will just ignore what it doesn't understand: it won't throw an error, and it won't stop parsing the file.

JavaScript is different. If you make an error in your JavaScript, or use a JavaScript method or property that a browser doesn't recognise, that browser *will* throw an error, and it *will* stop parsing the file. That's why it's important to test for features before using them in JavaScript. That's also why it isn't safe to rely on JavaScript for core functionality.

In my case, I need to test for the existence of `addEventListener`:

```
(function (win, doc) {
  if (!win.addEventListener) {
    return;
  }
  ...
}(this, this.document));
```

The good folk over at the BBC call this kind of feature test cutting the mustard. If a browser passes the test, it cuts the mustard, and so it gets the enhancements. If a browser doesn't cut the mustard, it doesn't get the enhancements. And that's fine because, remember, websites don't need to look the same in every browser.

I want to make sure that my off-canvas styles are only going to apply to mustard-cutting browsers. I'm going to use JavaScript to add a class of `.cutsthemustard` to the document:

```
(function (win, doc) {
  if (!win.addEventListener) {
    return;
  }
  ...
  var enhanceclass = 'cutsthemustard';
  doc.documentElement.className += ' ' + enhanceclass;
}(this, this.document));
```

Now I can use the existence of that class name to adjust my CSS:

```
@media all and (max-width: 35em) {
  .cutsthemustard [role="main"] {
    transition: all .25s;
    width: 100%;
    position: absolute;
    z-index: 2;
    top: 0;
    right: 0;
```

```
  }
  .cutsthemustard [role="navigation"] {
    width: 75%;
    position: absolute;
    z-index: 1;
    top: 0;
    right: 0;
  }
  .cutsthemustard .active [role="main"] {
    transform: translateX(-75%);
  }
}
```

See the enhanced mustard-cutting off-canvas navigation. Remember, this only applies to small screens so you might have to squish your browser window.

ENHANCE ALL THE THINGS!

This was a relatively simple example, but it illustrates the thinking behind progressive enhancement: once you're providing the core functionality to everyone, you're free to go crazy with all the latest enhancements for modern browsers.

Progressive enhancement doesn't mean you have to provide *all* the same functionality to everyone – quite the opposite. That's why it's key to figure out early on what the *core* functionality is, and make sure that it can be provided with the most basic technology. But from that point on, you're free to add many more features that

aren't mission-critical. You should reward more capable browsers by giving them more of those features, such as animation in CSS, geolocation in JavaScript, and new input types in HTML.

Like I said, progressive enhancement isn't a technology. It's a way of thinking. Once you start thinking this way, you'll be prepared for whatever the next ten years throws at us.

ABOUT THE AUTHOR

Jeremy Keith is an Irish web developer living in Brighton, England where he works with the web consultancy firm Clearleft. He wrote the books, DOM Scripting, Bulletproof Ajax, and most recently HTML5 For Web Designers.

His latest project is Huffduffer, a service for creating podcasts of found sounds. When he's not making websites, Jeremy plays bouzouki in the band Salter Cane. His loony bun is fine benny lava.

10. Making Sites More Responsive, Responsibly

Sally Jenkinson 24ways.org/201410

With digital projects we're used to shifting our thinking to align with our target audience. We may undertake research, create personas, identify key tasks, or observe usage patterns, with our findings helping to refine our ongoing creations. A product's overall experience can make or break its success, and when it comes to defining these experiences our development choices play a huge role alongside more traditional user-focused activities.

The popularisation of responsive web design is a great example of how we are able to shape the web's direction through using technology to provide better experiences. If we think back to the move from table-based layouts to CSS, initially our clients often didn't know or care about the difference in these approaches, but we did. Responsive design was similar in this respect –

momentum grew through the web industry choosing to use an approach that we felt would give a better experience, and which was more future-friendly.

We tend to think of responsive design as a means of displaying content appropriately across a range of devices, but the technology and our implementation of it can facilitate much more. A responsive layout not only helps your content work when the newest smartphone comes out, but it also ensures your layout suitably adapts if a visually impaired user drastically changes the size of the text.

The 24 ways site at 400% on a Retina MacBook Pro displays a layout more typically used for small screens.

When we think more broadly, we realise that our technical choices and approaches to implementation can have knock-on effects for the greater good, and beyond our initial target audiences. We can make our experiences *more* responsive to people's needs, enhancing their usability and accessibility along the way.

BEING RESPONSIBLY RESPONSIVE

Of course, when we think about being more responsive, there's a fine line between creating useful functionality and becoming intrusive and overly complex. In the excellent ***Responsible Responsive Design***, Scott Jehl states that:

> A responsible responsive design equally considers the following throughout a project:
>
> - Usability: The way a website's user interface is presented to the user, and how that UI responds to browsing conditions and user interactions.
> - Access: The ability for users of all devices, browsers, and assistive technologies to access and understand a site's features and content.
> - Sustainability: The ability for the technology driving a site or application to work for devices that exist today and to continue to be usable and accessible to users, devices, and browsers in the future.
> - Performance: The speed at which a site's features and content are perceived to be delivered to the user and the efficiency with which they operate within the user interface.

Scott's book covers these ideas in a lot more detail than I'll be able to here (put it on your Christmas list if it's not there already), but for now let's think a bit more about our roles as digital creators and the power this gives us.

Our choices around technology and the decisions we have to make can be extremely wide-ranging. Solutions will vary hugely depending on the needs of each project, though we can further explore the concept of making our creations more responsive through the use of humble web technologies.

THE POWER OF THE WEB

We all know that under the HTML5 umbrella are some great new capabilities, including a number of JavaScript APIs such as geolocation, web audio, the file API and many more. We often use these to enhance the functionality of our sites and apps, to add in new features, or to facilitate device-specific interactions.

You'll have seen articles with flashy titles such as "Top 5 JavaScript APIs You've Never Heard Of!", which you'll probably read, think "That's quite cool", yet never use in any real work.

There is great potential for technologies like these to be misused, but there are also great prospects for them to be used well to enhance experiences. Let's have a look at a few examples you may not have considered.

Offline first

When we make websites, many of us follow a process which involves user stories – standardised snippets of context explaining who needs what, and why.

"As a student I want to pay online for my course so I don't have to visit the college in person."

"As a retailer I want to generate unique product codes so I can manage my stock."

We very often focus heavily on **what** needs doing, but may not consider carefully **how** it will be done. As in Scott's list, accessibility is extremely important, not only in terms of providing a great experience to users of assistive technologies, but also to make your creation more accessible in the general sense – including under different conditions.

Offline first is yet another 'first' methodology (my personal favourite being 'tea first'), which encourages us to develop so that connectivity itself is an enhancement – letting users continue with tasks even when they're offline. Despite the rapid growth in public Wi-Fi, if we consider data costs and connectivity in developing countries, our travel habits with planes, underground trains and roaming (or simply if you live in the UK's signal-barren East Anglian wilderness as I do), then you'll realise that connectivity isn't as ubiquitous as our internet-addled brains would make us believe. Take a scenario that I'm sure we're all familiar with – the digital conference. Your venue may be in a city served by high-speed networks, but after overloading capacity with a full house of hashtag-hungry attendees, each carrying several devices, then everyone's likely to be offline after all. Wouldn't it be better if we could do something like this instead?

- Someone visits our conference website.

- On this initial run, some assets may be cached for future use: the conference schedule, the site's CSS, photos of the speakers.
- When the attendee revisits the site on the day, the page shell loads up from the cache.
- If we have cached content (our session timetable, speaker photos or anything else), we can load it directly from the cache. We might then try to update this, or get some new content from the internet, but the conference attendee already has a base experience to use.
- If we don't have something cached already, then we can try grabbing it online.
- If for any reason our requests for new content fail (we're offline), then we can display a pre-cached error message from the initial load, perhaps providing our users with alternative suggestions from what is cached.

There are a number of ways we can make something like this, including using the application cache (AppCache) if you're that way inclined. However, you may want to look into service workers instead. There are also some great resources on Offline First! if you'd like to find out more about this.

Building in offline functionality isn't necessarily about starting offline first, and it's also perfectly possible to retrofit sites and apps to catch offline scenarios, but this kind of graceful degradation can end up being more complex than if we'd considered it from the start. By

treating connectivity as an enhancement, we can improve the experience and provide better performance than we can when waiting to counter failures. Our websites can respond to connectivity and usage scenarios, on top of adapting how we present our content. Thinking in this way can enhance each point in Scott's criteria.

As I mentioned, this isn't necessarily the kind of development choice that our clients will ask us for, but it's one we may decide is simply the right way to build based on our project, enhancing the experience we provide to people, and making it more responsive to their situation.

Even more accessible

We've looked at accessibility in terms of broadening when we can interact with a website, but what about how? Our user stories and personas are often of limited use. We refer in very general terms to students, retailers, and sometimes just users. What if we have a student whose needs are very different from another student? Can we make our sites even more usable and accessible through our development choices?

Again using JavaScript to illustrate this concept, we can do a lot more with the ways people interact with our websites, and with the feedback we provide, than simply accepting keyboard, mouse and touch inputs and displaying output on a screen.

INPUT

Ambient light detection is one of those features that looks great in simple demos, but which we struggle to put to practical use. It's not new – many satnav systems automatically change the contrast for driving at night or in tunnels, and our laptops may alter the screen brightness or keyboard backlighting to better adapt to our surroundings. Using web technologies we can adapt our presentation to be better suited to ambient light levels.

If our device has an appropriate light sensor and runs a browser that supports the API, we can grab the ambient light in units using ambient light events, in JavaScript. We may then change our presentation based on different bandings, perhaps like this:

```
window.addEventListener('devicelight', function(e) {
    var lux = e.value;

    if (lux < 50) {
        //Change things for dim light
    }
    if (lux >= 50 && lux <= 10000) {
        //Change things for normal light
    }
    if (lux > 10000) {
        //Change things for bright light
    }
});
```

Live demo (requires light sensor and supported browser).

Soon we may also be able to do such detection through CSS, with `light-level` being cited in the Media Queries Level 4 specification. If that becomes the case, it'll probably look something like this:

```
@media (light-level: dim) {
  /*Change things for dim light*/
}

@media (light-level: normal) {
  /*Change things for normal light*/
}

@media (light-level: washed) {
  /*Change things for bright light*/
}
```

While we may be quick to dismiss this kind of detection as being a gimmick, it's important to consider that apps such as Light Detector, listed on Apple's accessibility page, provide important context around exactly this functionality.

> "If you are blind, Light Detector helps you to be more independent in many daily activities. At home, point your iPhone towards the ceiling to understand where the light fixtures are and whether they are switched on. In a room, move the device along the wall to check if there is a window and where it is. You can find out whether the shades are drawn by moving the device up and down."
>
> *everywaretechnologies.com/apps/lightdetector*

Input can be about so much more than what we enter through keyboards. Both an ever increasing amount of available sensors and more APIs being supported by the major browsers will allow us to cater for more scenarios and respond to them accordingly. This can be as complex or simple as you need; for instance, while `x-webkit-speech` has been deprecated, the web speech API is available for a number of browsers, and research into sign language detection is also being performed by organisations such as Microsoft.

OUTPUT

Web technologies give us some great enhancements around input, allowing us to adapt our experiences accordingly. They also provide us with some nice ways to provide feedback to users.

When we play video games, many of our modern consoles come with the ability to have rumble effects on our controller pads. These are a great example of an enhancement, as they provide a level of feedback that is entirely optional, but which can give a great deal of extra information to the player in the right circumstances, and broaden the scope of our comprehension beyond what we're seeing and hearing.

Haptic feedback is possible on the web as well. We could use this in any number of responsible applications, such as alerting a user to changes or using different patterns as a communication mechanism. If you find yourself in a pickle, here's how to print out SOS in Morse code through the vibration API. The following code indicates the length of vibration in milliseconds, interspersed by pauses in milliseconds.

```
navigator.vibrate([100, 300, 100, 300, 100, 300, 600,
300, 600, 300, 600, 300, 100, 300, 100, 300, 100]);
```

Live demo (requires supported browser)

WITH GREAT POWER…

What you've no doubt come to realise by now is that these are just more examples of progressive enhancement, whose inclusion will provide a better experience if the capabilities are available, but which we should not rely on. This idea isn't new, but the most

important thing to remember, and what I would like you to take away from this article, is that it is up to us to decide to include these kind of approaches within our projects – if we don't root for them, they probably won't happen. This is where our professional responsibility comes in.

We won't necessarily be asked to implement solutions for the scenarios above, but they illustrate how we can help to push the boundaries of experiences. Maybe we'll have to switch our thinking about how we build, but we can create more usable products for a diverse range of people and usage scenarios through the choices we make around technology. Let's stop thinking simply in terms of features inside a narrow view of our target users, and work out how we can extend these to cater for a wider set of situations.

When you plan your next digital project, consider the power of the web and the enhancements we can use, and try to make your projects even more responsive and responsible.

ABOUT THE AUTHOR

Sally Jenkinson is a freelance technical consultant and strategist, based in Colchester in the UK. Working with a mix of digital agencies and brands directly, Sally has been involved in projects for people like Nokia, Electronic Arts, The Open Data Institute, BBC, and Inghams and aims to get people talking and thinking about technology in a creative way. She's also a speaker, an author, and drinks a lot of tea.

You can find out more about Sally's work at sallyjenkinson.co.uk, and she tweets as @sjenkinson when she's not got her head stuck in a comic book or her hands wrapped around an Xbox controller.

11. Putting Design on the Map

Shane Hudson 24ways.org/201411

The web can leave us feeling quite detached from the real world. Every site we make is really just a set of abstract concepts manifested as tools for communication and expression. At any minute, websites can disappear, overwritten by a newfangled version or simply gone. I think this is why so many of us have desires to create a product, write a book, or play with the internet of things. We need to keep in touch with the physical world and to prove (if only to ourselves) that we do make real things.

I could go on and on about preserving the web, the challenges of writing a book, or thoughts about how we can deal with the need to make real things. Instead, I'm going to explore something that gives us a direct relationship between a website and the physical world – maps.

> A map does not just chart, it unlocks and formulates meaning; it forms bridges between here and there, between disparate ideas that we did not know were previously connected.
> *Reif Larsen, The Selected Works of T.S. Spivet*

The simplest form of map on a website tends to be used for showing where a place is and often directions on how to get to it. That's an incredibly powerful tool. So why is it, then, that so many sites just plonk in a default Google Map and leave it as that? You wouldn't just use dark grey Helvetica on every site, would you? Where's the personality? Where's the tailored experience? Where is the design?

JUMPING INTO DESIGN

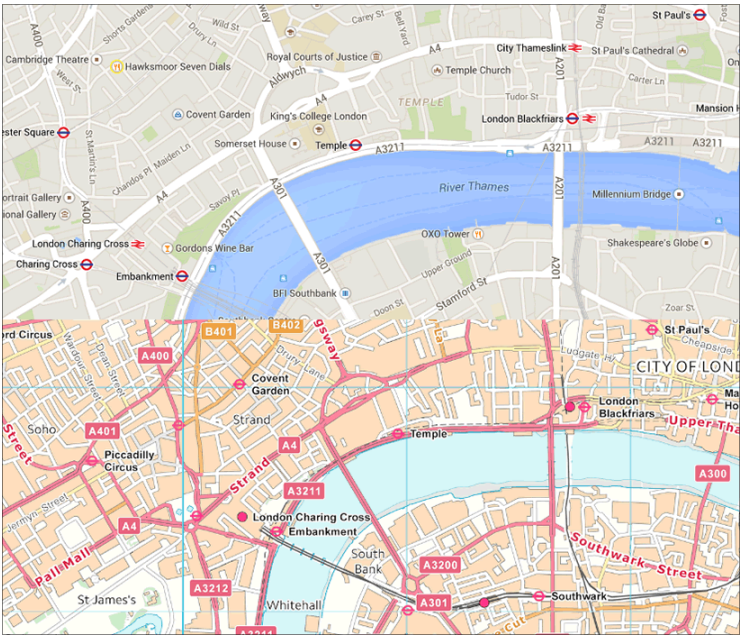
Let's keep this simple – we all want to be better web folk, not cartographers. We don't need to go into the history, mathematics or technology of map making (although all of those areas are really interesting to research). For the sake of our sanity, I'm going to gloss over some of the technical areas and focus on the practical concepts.

Tiles

If you've ever noticed a map loading in sections, it's because it uses tiles that are downloaded individually instead of requiring the user to download everything that

they might need. These tiles come in many styles and can be used for anything that covers large areas, such as base maps and data. You've seen examples of alternative base maps when you use Google Maps as Google provides both satellite imagery and road maps, both of which are forms of base maps. They are used to provide context for the real world, or any other world for that matter. A marker on a blank page is useless.

The tiles are representations of the physical; they do not have to be photographic imagery to provide context. This means you can design the map itself. The easiest way to conceive this is by comparing Google's road maps with Ordnance Survey road maps. Everything about the two maps is different: the colours, the label fonts and the symbols used. Yet they still provide the exact same context (other maps may provide different context such as terrain contours).

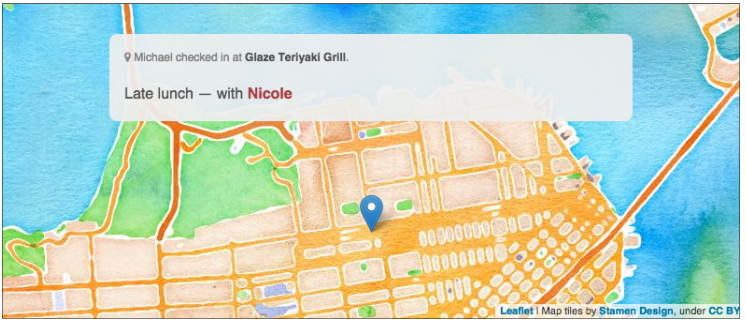


Comparison of Google Maps (top) and the Ordnance Survey (bottom).

Carefully designing the base map tiles is as important as any other part of the website. The most obvious, yet often overlooked, aspect are aesthetics and branding. Maps could fit in with the rest of the site; for example, by matching the colours and line weights, they can enhance the full design rather than inhibiting it. You're also able to define the exact purpose of the map, so instead of showing everything you could specify which symbols or labels to show and hide.

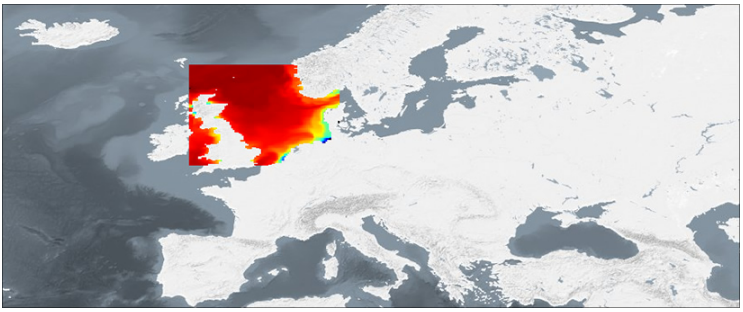
I've not done any real research on the accessibility of base maps but, having looked at some of the available options, I think a focus on the typography of labels and the colour of the various elements is crucial. While you can choose to hide labels, quite often they provide the data required to make sense of the map. Therefore, make sure each zoom level is not too cluttered and shows enough to give context. Also be as careful when choosing the typeface as you are in any other design work. As for colour, you need to pay closer attention to issues like colour-blindness when using colour to convey information. Quite often a spectrum of colour will be used to show data, or to show the topography, so you need to be aware that some people struggle to see colour differences within a spectrum.

A nice example of a customised base map can be found on Michael K Owens' check-in pages:



One of Michael K Owens' check-in pages.

As I've already mentioned, tiles are not just for base maps: they are also for data. In the screenshot below you can see how Plymouth Marine Laboratory uses tiles to show data with a spectrum of colour.



A map from the Marine Operational Ecology data portal, showing data of adult cod in the North Sea.

TECHNICAL

You're probably wondering how to design the base layers. I will briefly explain the concepts here and give you tools to use at the end of the article. If you're worried about the time it takes to design the maps, don't be – you can automate most of it. You don't need to manually draw each tile for the entire world!

We've learned the importance of web standards the hard way, so you'll be glad (and I won't have to explain the advantages) of the standard for web mapping from the Open Geospatial Consortium (OGC) called the Web Map Service (WMS). You can use conventional file formats for

the imagery but you need a way to query for the particular tiles to show for the area and zoom level, that is what WMS does.

Features

Tiles are great for covering large areas but sometimes you need specific smaller areas. We call these **features** and they usually consist of polygons, lines or points. Examples include postcode boundaries and routes between places, or even something more dynamic such as borders of nations changing over time.

Showing features on a map presents interesting design challenges. If the colour or shape conveys some kind of data beyond geographical boundaries then it needs to be made obvious. This is actually really hard, without building complicated user interfaces. For example, in the image below, is it obvious that there is a relationship between the colours? Does it need a way of showing what the colours represent?

Choropleth map showing ranked postcode areas, using ViziCities.

> Features are represented by means of lines or colors; and the effective use of lines or colors requires more than knowledge of the subject – it requires artistic judgement.
> *Erwin Josephus Raisz, cartographer (1893–1968)*

Where lots of boundaries are small and close together (such as a high street or shopping centre) will it be obvious where the boundaries are and what they represent? When designing maps, the hardest challenge is dealing with how the data is represented and how it is understood by the user.

TECHNICAL

As you probably gathered, we use WMS for tiles and another standard called the web feature service (WFS) for specific features. I need to stress that the difference between the two is that WMS is for tiling, whereas WFS is for specific features. Both can use similar file formats but should be used for their particular use cases. You may be wondering why you can't just use a vector format such as KML, GeoJSON (or even SVG) – and you can – but the issue is the same as for WMS: you need a way to query the data to get the correct area and zoom level.

User interface

There is of course never a correct way to design an interface as there are so many different factors to take into consideration for each individual project. Maps can be used in a variety of ways, to provide simple information about directions or for complex visualisations to explain large amounts of data. I would like to just touch on matters that need to be taken into account when working with maps.

As I mentioned at the beginning, there are so many Google Maps on the web that people seem to think that its UI is the only way you can use a map. To some degree we don't want to change that, as people know how to use them; but does every map require a zoom slider or base map toggle? In fact, does the user need to zoom at all? The answer to that one is generally yes, zooming does provide more context to where the map is zoomed in on.

In some cases you will need to let users choose what goes on the map (such as data layers or directions), so how do they show and hide the data? Does a simple drop-down box work, or do you need search? Google's base map toggle is quite nice since it doesn't offer many options yet provides very different contexts and styling.

It isn't until we get to this point that we realise just plonking a quick Google map is really quite ridiculous, especially when compared to the amount of effort we

make in other areas such as colour, typography or how the CSS is written. Each of these is important but we need to make sure the whole site is designed, and that includes the maps as much as any other content.

Putting it into practice

I could ramble on for ages about what we can do to customise maps to fit a site's personality and correctly represent the data. I wanted to focus on concepts and standards because tools constantly change and it is never good to just rely on a tool to do the work. That said, there are a large variety of tools that will help you turn these concepts into reality. This is not a comparison; I just want to show you a few of the many options you have for maps on the web.

GOOGLE

OK, I've been quite critical so far about Google Maps but that is only because there is such a large amount of the default maps across the web. You can style them almost as much as anything else. They may not allow you to use custom WMS layers but Google Maps does have its own version, called styled maps. Using an array of map features (in the sense of roads and lakes and landmarks rather than the kind WFS is used for), you can style the base map with JavaScript. It even lets you toggle visibility, which helps to avoid the issue of too much clutter on the

map. As well as lacking WMS, it doesn't support WFS, but it does support GeoJSON and KML so you can still show the features on the map. You should also check out Google Maps Engine (the new version of My Maps), which provides an interface for creating more advanced maps with a selection of different base maps. A premium version is available, essentially for creating map-based visualisations, and it provides a step up from the main Google Maps offering. A useful feature in some cases is that it gives you access to many datasets.

LEAFLET

You have probably seen Leaflet before. It isn't quite as popular as Google Maps but it is definitely used often and for good reason. Leaflet is a lightweight open source JavaScript library. It is not a service so you don't have to worry about API throttling and longevity. It gives you two options for tiling, the ability to use WMS, or to directly get the file using variables in the filename such as */{z}/{x}/{y}.png*. I would recommend using WMS over dynamic file names because it is a standard, but the ability to use variables in a file name could be useful in some situations. Leaflet has a strong community and a well-documented API.

MAPBOX

As a freemium service, Mapbox may not be perfect for every use case but it's definitely worth looking into. The service offers incredible customisation tools as well as lots of data sources and hosting for the maps. It also provides plenty of libraries for the various platforms, so you don't have to only use the maps on the web.

Mapbox is a service, though its map design tool is open source. Mapbox Studio is a vector-only version of their previous tool called Tilemill. Earlier I wrote about how typography and colour are as important to maps as they are to the rest of a website; if you thought, "Yes, but how on earth can I design those parts of a map?" then this is the tool for you. It is incredibly easy to use. Essentially each map has a stylesheet.

If you do not want to open a paid-for Mapbox account, then you can export the tiles (as PNG, SVG etc.) to use with other map tools.

OPENLAYERS

After a long wait, OpenLayers 3 has been released. It is similar to Leaflet in that it is a library not a service, but it has a much broader scope. During the last year I worked on the GIS portal at Plymouth Marine Laboratory (which I used to show the data tiles earlier), it essentially used

OpenLayers 2 to create a web-based geographic information system, taking a large amount of data and permitting analysis (such as graphs) without downloading entire datasets and complicated software. OpenLayers 3 has improved greatly on the previous version in both performance and accessibility. It is the ideal tool for complex map-based web apps, though it can be used for the simple use cases too.

OPENSTREETMAP

I couldn't write an article about maps on the web without at least mentioning OpenStreetMap. It is the place to go for crowd-sourced data about any location, with complete road maps and a strong API.

VIZICITIES

The newest project on this list is ViziCities by Robin Hawkes and Peter Smart. It is a open source 3-D visualisation tool, currently in the very early stages of development. The basic example shows 3-D buildings around the world using OpenStreetMap data. Robin has used it to create some incredible demos such as real-time London underground trains, and planes landing at an airport. Edward Greer and I are currently working on using ViziCities to show ideal housing areas based on particular personas. We chose it because the 3-D aspect gives us interesting possibilities for the data we are able

to visualise (such as bar charts on the actual map instead of in the UI). Despite not being a completely stable, fully featured system, ViziCities is worth taking a look at for some use cases and is definitely going to go from strength to strength.

So there you have it – a whistle-stop tour of how maps can be customised. Now please stop plonking in maps without thinking about it and design them as you design the rest of your content.

ABOUT THE AUTHOR

Despite being a constant presence on Twitter, Shane Hudson occasionally does some work. He is a developer interested in all things web. Currently focussing on completing a degree in Artificial Intelligence, Shane has previously written a book called JavaScript Creativity, worked on a web-based geographic data portal at Plymouth Marine Laboratory and freelanced as a front-end developer.

12. Is Agile Harder for Agencies?

Charlie Perrins 24ways.org/201412

I once sat in a pitch meeting and watched a new business exec tell a potential client that his agency followed an agile workflow process at all times. The potential client nodded wisely, and they both agreed that agile was indeed the way to go.

The meeting progressed and they signed off on a contract for a massive project, to be delivered in a standard waterfall fashion, with all manner of phases and key deliverables.

Of course both of them left the meeting perfectly happy, because neither of them knew nor cared what an agile workflow process might be.

That was about five years ago. As 2015 heaves into view I think it's fair to say that attitudes have changed. Perhaps the same number of people claim to do Agile™ now as in 2010, but I think more of them are telling the truth.

As a developer in an agency that works primarily with larger organisations, this year I have started to see a shift from agencies pushing agile methodologies with their clients, to clients requesting and even demanding agile practices from their agencies. Only a couple of years ago this would have been unusual behaviour.

SO WHAT'S THE PROBLEM?

We should be happy then, no? Those of us in agencies will get to spend more time delivering great products, and less time arguing over out-of-date functional specs or battling through an adversarial change management procedure because somebody had a good idea during development rather than planning. We get to be a little bit more like our brothers and sisters in vaunted teams like the Government Digital Service, which is using agile approaches to great effect on projects that have a real benefit to their users.

Almost. Unfortunately, it seems to be the case that adhering to an agile framework such as scrum is more difficult within an agency/client structure than it is for an in-house development team.

This is no surprise. The Agile Manifesto was written in 2001 by a group of software developers for their own use. Many of the underlying principles of a framework like Scrum assume the existence of an in-house team, working

on a highly technical project, and working for the business that employs them. The agency/client model must to some extent be retrofitted into agile frameworks. It can be done though, and there are plenty of agencies out there doing it well.

This article isn't meant to be another introduction to agile techniques – there are too many of those online already. This article is for people just dipping their toes into this way of working. I've laid out a few of the key reasons why adopting a more fully agile approach seems difficult, at least initially, for those of us working in agencies.

1. AGILE ASKS MORE OF YOUR CLIENTS

When a team adopts Scrum everyone has to get used to a number of unfamiliar roles and rituals. Few team members have a steeper learning curve than the person designated as the product owner.

The product owner carries a lot of weight on their shoulders. They have to uphold the overall vision for the project. They are also meant to be the primary author of the project's user stories (short atomic descriptions of project features which are testable and relate to a real business need). They should own this list of stories (called a backlog) and should be able to prioritise the order in

which the stories are developed, to ensure that the project is delivering real value to the business early and often.

When a burst of work is completed (bursts of work in Scrum are called sprints), the product owner leads a review or show-and-tell session with the wider project stakeholders. The product owner needs to understand the work that has been completed, and must champion it to the business. Finally, and most importantly, the product owner is responsible for managing the feedback and requests from stakeholders in such a way that they don't derail the project team's agreed workload for any given sprint, without upsetting or offending any of the stakeholders – some of whom may outrank the product owner.

If you follow that spec, this is a job for a **superhuman** in any organisational context. And within the agency/client structure this superhuman needs to be client-side for the process to be at its most effective.

So your client, who in the past might have briefed a project to an agency team and then had the work presented back to them every few weeks, is now asked to be involved with the team on a daily basis; to fight on behalf of the team when new or difficult requests come in from senior figures within their organisation; and to

present the agency's work to their own colleagues after each sprint. It's a big change if all that gets dropped into someone's lap without warning.

There are several ways agencies can mitigate this issue. The ScrumAlliance suggests some alternative ways to structure the product owner role. The approach I have taken in the past is simply to start slow, and gradually move more of the product owner role over to the client side as and when they feel comfortable with it. If you're working together long-term on a project, and you both see tangible improvements in the quality of the work after adopting an agile process, then your client is more likely to be open to further changes as the partnership progresses.

2. MY CLIENT WANTS FIXED COSTS, FIXED DEADLINES AND A FIXED SCOPE

I know. Mine too. Of course they do – it is the way that agencies and clients have agreed to work in digital and other creative service industries for a very long time. On both sides of the fence we're used to thinking about projects in this way.

Of the three, fixing scope is the one that agile purists would rail hardest against. The more time we spend working on digital projects, the less sense it makes. James Archer, CEO of UI/UX design agency Forty puts it like this:

> For me, the Agile approach is really about acknowledging that disturbing truth that every project manager knows, but has trouble admitting. The truth that the project plan is wrong. Scope creep. Change orders. Shifting priorities. New directions. We act shocked and appalled when those things happen during our carefully planned project, even though they happen on every project ever.

Successful relationships require trust and honesty, and we shouldn't be afraid of discussing this aspect of project management. If you do move away from a fixed scope of work, then the other two items (costs and timings) can be fixed – more or less. If you can get your clients to buy into this from a standing start then you are doing well. In fact you probably deserve a promotion. For most of us this is a continual discussion.

Anyway, as soon as you've made headway on the argument that it makes little or no sense to try and fix the scope of a digital project, you usually run into a related concern, which we'll look at next.

3. FEAR OF UNCONTROLLED COSTS

We all know that a dog is for life, not just for Christmas. At this time of year perhaps we should reiterate to everyone that digital products and services also need support and love once we have taken the decision to bring them into the world.

More organisations are realising that their investment in digital platforms should be viewed as an operational expenditure rather than a capital expenditure. But from time to time we will find ourselves working on projects for people who have a finite amount of money to invest in a product at a given point in time. When agencies start talking about these projects as rolling investments those responsible can understandably worry about their costs running out of control.

There's another factor at play here. Agile, on the whole, prefers to derive a cost for services from the hours a team spends working on a project. In other industries this is referred to as charging for time and materials, and there seems to be an ingrained distrust in this approach among people in general. See, for example, the Citizens Advice Bureau's "Top tips for employing a builder":

> "Bear in mind that if you pay a daily rate, this makes it easier for a builder to string the work out and get more money so agree what you will do if the job takes longer than expected."

It's hard not to feel stung if you are in the builder's shoes here, as we are when we're talking about our role as an agency. But if you've ever haggled with a builder over time and materials, and also moaned about your clients misunderstanding agile methods, take a moment to reflect on the similarities from your client's point of view.

Again, there are some things we can do to mitigate this issue. Some agencies put in place a service level agreement around their team's velocity (an agile-related term related to how much work a team delivers in any given sprint) and this can help.

As the industry moves further towards a long-term approach to investment in digital I hope this fear will subside. But that shift in approach leads to the final concern I want to address.

4. AGENCY STRUCTURES NEED SHAKING UP

If you work for a company that has spent many years developing a business model around the waterfall process, you may have to break through many layers of entrenched thinking in order to establish new practices and effect organisational change.

There are consultancies that exist specifically to help agencies through their own agile transformation. One of these companies, AgencyAgile, provides a helpful list of

common pitfalls. They emphasise the need to look at your whole agency's structure, rather than simply encouraging project teams to adopt new workflows.

> Even awesomely run Agile projects can have a limited impact on the overall organization.

If you're serious about changing the way your company approaches projects then try talking to people who sit outside the usual project delivery team. Speak to the finance department if you have one, and try to convince your senior management team if they're not already on board. And definitely speak to your new business people, who go out there and win the projects you get to work on.

It's these people who need to understand the potential business benefits of working in a new way, and also which of their existing habits and behaviours they might need to change to accommodate a new approach.

Otherwise you'll find yourself with a team of designers, developers and project managers who are ready and waiting to deliver work in an iterative and collaborative way, but by the time they get hold of the project a cost has already been agreed, a deadline has been imposed, and a functional requirements document has been painstakingly put together. Nobody wins in this situation.

CONCLUSION

So where should we go from here? I certainly don't have hard and fast answers – I'm not sure that they exist in a one-size-fits-all approach for agencies.

There are plenty of smart people thinking about this problem. It's a hot topic right now. Earlier in the year a London-based meetup was established called Agile for Agencies. If you're in the capital and want to discuss these issues with your peers it's a great opportunity to do so.

I've mentioned James Archer and Forty already. Both James and Paul Boag have written in the last twelve months on this subject. They both come out on the side of the argument that suggests you adopt agile principles, but don't have to worry about the rituals if they don't fit in with your practices.

Personally, I think the rituals and the discipline mandated by an agile framework like Scrum can provide a great deal of value to your team, even it if is hard to implement within an agency culture that has traditionally structured its work and its services in another way.

In whatever way you figure out the details, when your teams collaborate with your clients rather than work for them at arm's length, and when everyone prioritises frequent delivery, reflection and iteration over exhaustive scoping and planning, I believe you'll see a tangible difference in the quality of the work that you create.

ABOUT THE AUTHOR

Charlie Perrins is Technical Director at Dare. He's a front-end developer by trade, and a nut for semantic and readable code. He writes and talks about technologies old and new to anyone who'll listen. Most recently he's spoken at events run by Faber & Faber and at Front End London.

Charlie tweets pretty regularly, but is an unreliable blogger. His crowning achievement in self-publishing came some five years ago and was entitled simply 'The Bacon Project'.

Photo by Steve Whittington

13. The Introvert Owner's Manual

Inayaili de León Persson 24ways.org/201413

> Nobody realizes that some people expend tremendous energy merely to be normal.
> *Albert Camus*

"Whatever you plan, just make sure there are lots of people there," said my husband in the run-up to his birthday last year. A few months later, before my own birthday, I uttered, "Whatever you plan, just make sure it is only me and you."

I am an introvert. It is very likely some of you are too, or that you live, work or fraternise with one. Despite there being quite a few of us out there – some say as many as one third of the population, others as little as ten per cent – I think our professional and social lives are biased towards a definition of normality that is more accepting of the extrovert. I hope that by reading this article you will

gain some insight to what goes on inside the head of the introvert(s) that you know and understand how to relate to them in a way that respects their disposition.

Before we go any further, I should define what exactly being an introvert means, and, equally important, what it does not. Only once this is established will you be able to handle your introvert correctly.

WHAT DEFINES AN INTROVERT

The simplest and most accurate way of describing an introvert is that she uses up energy in social situations and needs to be in solitude to recharge.

To explain what I mean, let us take the example of the The Sims: when you create a Sim, you can choose (among other characteristics) whether it will be outgoing or not. If the Sim is outgoing, when you play the game you need to make sure it interacts as much as possible with other Sims or its mood indicator (the plumbob) will become red and that is a bad thing. Conversely, if your Sim is not outgoing, when you put it in too many social situations its plumbob will become red too.

So your (real life) introvert might think you are great (you might even be her best friend, her spouse or her child), but if her plumbob is red, or nearly, she might just need a little time and space to recharge before she is ready to interact.

This is not the same thing as being shy or in a bad mood all the time. We are not necessarily awkward in social situations, but, if we have not had the time to recharge, being social might be almost impossible. This explains why your introvert will likely ask who will be at the gathering you have planned, for how long she will have to stay there, and what she will be doing before and after the event. It is the equivalent of you wanting to know if there will be power sockets in the restaurant to charge your iPhone – asking this does not mean you don't want to attend.

The explanation above might be a simplistic way of looking at things, but I would say it is one that introverts can relate to; call it a minimalist approach to socialisation.

CARING FOR YOUR INTROVERT

Articles and conversations about introversion usually focus on how to fix the condition and how to make introverts more outgoing: a clear example of our society's bias towards the normality of extroversion. Avoid this. You will not be able to convert your introvert into an extrovert. Believe it or not, there is nothing wrong with her.

In her 2012 TED talk, "The power of introverts", Susan Cain pointed to the fact that places like school and work are designed for extroverts: students and workers are required to constantly work in groups and speaking up is

highly valued. Both types are evaluated against the same criteria and more often than not people are expected to excel at being outspoken to be considered well rounded.

Obviously, this is not the right way to appraise your introvert. Comparing your introvert with an extrovert using the same parameters and simply asking her to behave more like an extrovert is a mistake and something that will only perpetuate an introvert's idea that the problem lies with her.

Speaking up

Your introvert is likely to have strong opinions and ideas, and to have been listening to other people speak at meetings and workshops. Help her voice those thoughts by creating an environment where everyone stops and listens when someone speaks instead of one which fosters interruptions. Show her that it is acceptable for someone to take time to think before they speak: silences are OK. Allow her the freedom to be herself instead of pressuring her to change an innate quality.

It is not uncommon to find an introvert who likes to express ideas in writing. The world of web professionals excels in the spread of knowledge that is shared and sought through the written word. Give your introvert the

necessary time and tools to write about the job, if she is that way inclined; this might be a good alternative to asking her to speak out.

Group work

I remember the sinking feeling whenever I heard my teachers say the dreaded words: "And now you're going to break out into groups of…" Being an introvert does not mean you do not like people (or like to be around or work with others). It is just that activities such as group work will invariably drain your introvert's energy rapidly. Your introvert's batteries will need to be fully charged for her to be at her best and afterwards she will most likely need to recharge.

Quiet time

These days, one of the things that I value most at work is the ability to have moments to create and to think in solitude. When I am able to have those moments at the right time I will in turn be happy to participate in group conversations and tasks. Allow your introvert to have those moments: this does not mean she will have to work from home one day a week (but maybe it will); it might simply mean allowing her to take her laptop and her notebook and work from the empty side of the office, or from the coffee shop downstairs for an hour or two. In all

likelihood she will come back fully recharged and ready to engage in more social activities – her plumbob will again be bright green.

Leadership

Do not think that your introvert cannot lead. Cain notes that introverted leaders are more likely to let their proactive employees run with their ideas instead of taking the ideas as their own. I would say that is a positive attribute in a leader. Maybe next time a project starts, talk to your introvert about the possibility of her being in a leadership position or of having more responsibility: you might be surprised at her ability to plan and foresee potential obstacles in the project.

FINAL THOUGHTS

You would not tell someone with dyslexia to get better at spelling without giving her the right tools and enough time to do so. Equally, do not ask your introvert to be more outgoing, or to turn her frown upside down, without giving her the space to do so.

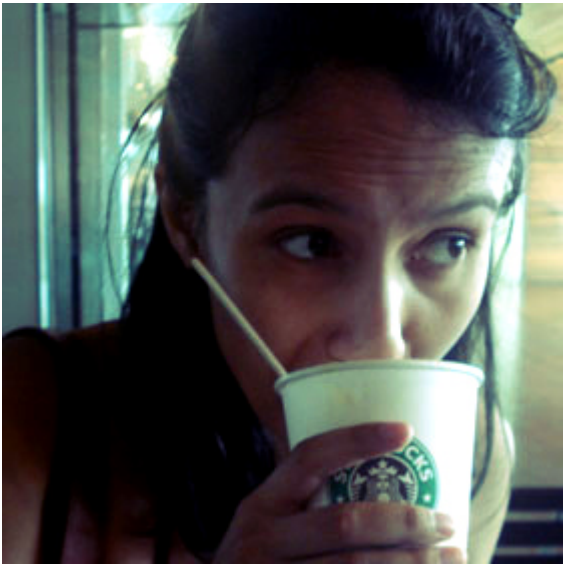
I believe that everyone is an introvert at some point. Everyone needs a moment of solitude now and then, and the work we do requires frequent periods of deep focus and concentration. Striving to create workplaces,

classrooms, homes that allow introverts to shine and be comfortable in their skin has the potential to also make those places more balanced for everyone else.

RESOURCES AND FURTHER READING

- The power of introverts
- 10 myths about introverts
- Susan Cain's 2014 TED Talk | Announcing the Quiet Revolution
- Help Shy Kids — Don't Punish Them
- The Introvert Advantage
- 6 Things You Thought Wrong About Introverts
- Extraversion and introversion

ABOUT THE AUTHOR



Inayaili de León Persson (or just Yaili) is a web designer and author. She's Lead Web Designer at Canonical, the company that delivers Ubuntu. She's Panamanian Portuguese, born in the USSR, and has been living in London since 2008 — her favourite city in the world. She loves cats and naps.

14. Five Ways to Animate Responsibly

Rachel Nabors 24ways.org/201414

It's been two years since I wrote about "Flashless Animation" on this very site. Since then, animation has steadily begun popping up on websites, from sleek app-like user interfaces to interactive magazine-like spreads. It's an exciting time for web animation wonks, interaction developers, UXers, UI designers and a host of other acronyms!

But in our rush to experiment with animation it seems that we're having fewer conversations about whether or not we should use it, and more discussions about what we can do with it. We spend more time fretting over how to animate all the things at 60fps than we do devising ways to avoid incapacitating users with vestibular disorders.

I love web animation. I live it. And I make adorably silly things with it that have no place on a self-respecting production website. I know it can be abused. We've all made fun of *Flash-turbation*. But how quickly we forget the lessons we learned from that period of web design. Parallax scrolling effects may be the skip intro of this generation. Surely we have learned better in the sobering up period between Flash and the web animation API.

So here are five bits of advice we can use to pull back from the edge of animation abuse. With these thoughts in mind, we can make 2015 the year web animation came into its own.

ANIMATE DELIBERATELY

Sadly, animation is considered decorative by the bulk of the web development community. UI designers and interaction developers know better, of course. But when I'm teaching a workshop on animation for interaction, I know that my students face an uphill battle against decision makers who consider it nice to have, and tack it on at the end of a project, if at all.

This stigma is hard to shake. But it starts with us using animation deliberately or not at all. Poorly considered, tacked-on animation will often cause more harm than good. Users may complain that it's too slow or too fast, or that they have no idea what just happened.

When I was at Chrome Dev Summit this year, I had the privilege to speak with Roma Sha, the UX lead behind Polymer's material design (with the wonderful animation documentation). I asked her what advice she'd give to people using animation and transitions in their own designs. She responded simply: **animate deliberately**. If you cannot afford to slow down to think about animation and make well-informed and well-articulated decisions on behalf of the user, it is better that you not attempt it at all. Animation takes energy to perform, and a bad animation is worse than none at all.

IT TAKES MORE THAN TWELVE PRINCIPLES

We always try to draw correlations between disparate things that spark our interest. Recently it feels like more and more people are putting the *The Illusion of Life* on their reading shelf next to *Understanding Comics*. These books give us so many useful insights from other industries. However, we should never mistake a website for a comic book or an animated feature film. Some of these concepts, while they help us see our work in a new light, can be more or less relevant to producing said work.

See: //player.vimeo.com/video/93206523

The illusion of life from cento lodigiani on Vimeo.

I am specifically thinking of the twelve principles of animation put forth by Disney studio veterans in that great tome *The Illusion of Life*. These principles are very useful for making engaging, lifelike animation, like a ball bouncing or a squirrel scampering, or the physics behind how a lightbox should feel transitioning off a page. But they provide no direction at all for when or how something should be animated as part of a greater interactive experience, like how long a drop-down should take to fully extend or if a group of manipulable objects should be animated sequentially or as a whole.

The twelve principles are a great place to start, but we have so much more to learn. I've documented at least six more functions of interactive animation that apply to web and app design. When thinking about animation, we should consider *why* and *how*, not just what, the physics. Beautiful physics mean nothing if the animation is superfluous or confusing.

USEFUL AND NECESSARY, *THEN* BEAUTIFUL

There is a Shaker saying: "Don't make something unless it is both necessary and useful; but if it is both necessary and useful, don't hesitate to make it beautiful." When it comes to animation and the web, currently there is very little documentation about what makes it useful or necessary. We tend to focus more on the beautiful, the

delightful, the aesthetic. And while aesthetics are important, they take a back seat to the user's overall experience.

See: //www.youtube.com/embed/m5MrbCGcaUI

The first time I saw the load screen for Pokemon Yellow on my Game Boy, I was enthralled. By the sixth time, I was mashing the start button as soon as Game Freak's logo hit the screen. What's delightful and meaningful to us while working on a project is not always so for our users. And even when a purely delightful animation is favorably received, as with Pokemon Yellow's adorable opening screen, too many repetitions of the cutest but ultimately useless animation, and users start to resent it as a hindrance.

If an animation doesn't help the user in some way, by showing them where they are or how two elements on a page relate to each other, then it's using up battery juice and processing cycles solely for the purpose of delight. Hardly the best use of resources.

Rather than animating solely for the sake of delight, we should first be able to articulate two things the animation does for the user. As an example, take this menu icon from Finethought.com (found via Use Your Interface). The menu icon does two things when clicked:

1. It gives the user feedback by animating, letting the user know its been clicked.
2. It demonstrates its changed relationship to the page's content by morphing into a close button.

Assuming we have two good reasons to animate something, there is no reason our third cannot be to delight the user.

GO FOUR TIMES FASTER

There is a rule of thumb in the world of traditional animation which is applicable to web animation: however long you think your animation should last, take that time and halve it. Then halve it again! When we work on an animation for hours, our sense of time dilates. What seems fast to us is actually unbearably slow for most users. In fact, the most recent criticism from users of animated interfaces on websites seems to be, "It's so slow!" A good animation is unobtrusive, and that often means running fast.

When getting your animations ready for prime time, reduce those durations to 25% of their original speed: a four-second fade out should be over in one.

INSTALL A KILL SWITCH

No matter how thoughtful and necessary an animation, there will be people who become physically sick from seeing it. For these people, we must add a way to turn off animations on the website.

Fortunately, web designers are already thinking of ways to empower users to make their own decisions about how they experience the web. As an example, this site for the animated film *Little from the Fish Shop* allows users to turn off most of the parallax effects. While it doesn't remove the animation entirely, this website does reduce the most nauseating of the animations.

See: //www.youtube.com/embed/2c3siLM1zlw

Animation is a powerful tool in our web design arsenal. But we must take care: if we abuse animation it might get a bad reputation; if we underestimate it, it won't be prioritized. But if we wield it thoughtfully, use it where it is both necessary and useful, and empower users to turn it off, animation is a tool that will help us build things that are easier to use *and* more delightful for years to come.

Let's make 2015 the year web animation went to work for users.

ABOUT THE AUTHOR

Rachel Nabors is an interaction developer and award-winning cartoonist. She travels the world, speaking and training people in the art of web animation. When not biking around her home city of Portland, she makes interactive comics at her company Tin Magpie. You can catch her as @rachelnabors on Twitter and at rachelnabors.com.

15. SEO in 2015 (and Why You Should Care)

Dave Collins 24ways.org/201415

If your business is healthy, you can always find plenty of reasons to leave SEO on your to-do list in perpetuity. After all, SEO is technical, complicated, time-consuming and potentially dangerous. The SEO industry is full of self-proclaimed gurus whose lack of knowledge can be deadly. There's the terrifying fact that even if you dabble in SEO in the most gentle and innocent way, you might actually end up in a worse state than you were to begin with.

To make matters worse, Google keeps changing the rules. There have been a bewildering number of major updates, which despite their cuddly names have had a horrific impact on website owners worldwide.

Fear aside, there's also the issue of time. It's probably tricky enough to find the time to read this article. Setting up, planning and executing an SEO campaign might well seem like an insurmountable obstacle.

So why should you care enough about SEO to do it anyway?

The main reason is that you probably already see between 30% and 60% of your website traffic come from the search engines. That might make you think that you don't need to bother, because you're already doing so well. But you're almost certainly wrong.

If you have a look through the keyword data in your Google Webmaster Tools account, you'll probably see that around 30–50% of the keywords used to find your website are brand names – the names of your products or companies. These are searches carried out by people who already know about you. But the people who don't know who you are but are searching for what you sell aren't finding you right now. This is your opportunity.

If a person goes looking for a company or product by name, Google will steer them towards what they're looking for. Their intelligence does have limits, however, and even though they know your name they won't be completely clear about what you sell. That's where SEO would come in.

Still need more convincing? How about the fact that the seeming complexities of SEO mean that your competition are almost certainly neglecting it too. They have the same reservations as you about complexity, time and danger,

and hopefully they aren't reading this article and so are none the wiser of the well-kept secret: that 70% of SEO is easy.

I'm going to lead you through what you need to do to tap into that stream of people looking for what you sell right now.

WHAT IS REAL SEO?

Real SEO is all about helping Google understand the content of your website. It's about steering, guiding and assisting Google. Not manipulating it.

It's easy to assume that Google already understands the content and relevance of each and every page on your website, but the fact is that it needs a fair amount of hand-holding. Fortunately, helping Google along really isn't very difficult at all.

Rest assured that real SEO has nothing to do with keyword stuffing, keyword density, hacks, tricks or cunning techniques. If you hear any of these terms from your SEO advisor, run away from them as quickly as you can.

UNDERSTANDING YOUR CURRENT SITUATION – GOOGLE ANALYTICS

Before you can do anything to improve your SEO status, you need to get an idea of how you're already doing. Below is a very quick and easy way of doing so.

1. Open up your Google Analytics account.

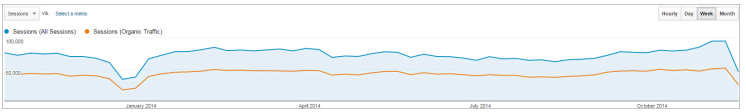
2. Click on the date range selector on the top-right of the interface and change the year of the first date to last year. So 12 Dec 2014 will become 12 Dec 2013. Then click on **Apply**.

3. Click on the **All Sessions** rectangle towards the top-left, click once on **Organic Traffic** and click **Apply**.

4. Click the little black-and-white squares icon that has now appeared under the date selector on the top-right, and drag the slider all the way over to **Higher Precision**.

5. Change the interval buttons on the top-right of the graph to **Week** to make this easier to digest.

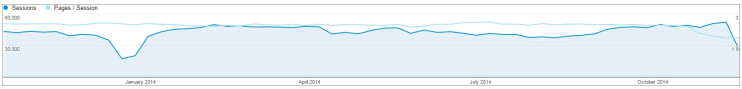
At this point your graph should look something like this:



It's worth noting the approximate proportion of your visitors that currently come from organic sources.

6. Click the little downwards arrow to the right of the **All Sessions** rectangle and choose **Remove**, so that we're only looking at the organic traffic on its own.

7. Click on **Select a metric** next to the **Sessions** button above the graph and select **Pages / Session**. You should then see something like this:



In the example above we can see that the quantity of traffic has been increasing since the middle of August, but the quality of the traffic (as measured by the number of pages per session) has fallen significantly.

How you choose to view this is down to your own graph, recent history and interpretation of events, but this should give you an indication of how things stand at the present time. Trends are often much more revealing than a snapshot of a brief moment in time.

YOUR GOOGLE WEBMASTER TOOLS DATA

If you're not very familiar with your Google Webmaster Tools account, it's really worth taking ten to fifteen minutes to see what's on offer. I can't recommend this enough. From the point of view of an SEO health check, I'd

advise you to look into the **HTML Improvements**, **Crawl Errors** and **Crawl Stats**, and most importantly the **Search Queries**.

From what you see here and the trends shown in your Analytics data, you should now have a good idea of your current status. If you want to explore further, I recommend Screaming Frog as a good diagnostics tool, or Botify if your website is large or unusually complex.

COMBINING THE DATA INTO SOMETHING USEFUL

Your Google Analytics session will have shown you how you're doing from an SEO point of view in terms of the quantity and, to some extent, the quality of your visitors. But it's only showing you what is already working. In other words: the people who are finding you on the search engines, and clicking on your links.

The Google Webmaster Tools search query data, on the other hand, will give you a better idea of what isn't working. It will show you the keyword searches that are getting you listed in the results, but which aren't necessarily getting clicked. And it doesn't take much by the way of expertise to see why.

For example, if you see your targeted keyword, which you feel is extremely relevant, has generated over 2,000 impressions in the last month but produced only two

clicks, you'll probably find a very low average position. Bear in mind that an average position of fourteen will mean being around halfway down the second page of results. Think about how rarely you go beyond the first two or three listings, never mind to the second page of results, and you'll understand why the click-through rate is so low.

So now you have an idea of what you're being found for at the present time. But what about the other terms?

WHAT WOULD YOU LIKE TO BE FOUND FOR?

This is one of the more common SEO mistakes, on a number of different levels.

Many businesses assume that they don't need to worry about keyword research. They think they know what terms people use to find what they sell, and they also assume that Google understands the content on their website. This is incorrect on all counts.

A better starting point is to brainstorm a small number of your most obvious keywords, then run them through Google's Keyword Planner. Ignore the information in the **Ad group ideas** tab, and instead go straight to the **Keyword ideas** tab. Rather than wade through the very unfriendly interface, I recommend downloading the data

as a spreadsheet, in which not only is more detail included, but you can also slice, dice, sort and report the data as required.

From there you can delete all the irrelevant columns, and start working your way through the list, deleting any irrelevant keywords as you go along.

It's around this stage that you may hit a problem in terms of where to focus your efforts. The number of reported searches for a given keyword is of course important, but so is the level of competition. Ideally, you'd like keywords with plenty of searches but not too much competition.

I personally like to factor both together by adding a column that simply divides the number of searches squared by the level of competition:

(*number of searches* × *number of searches*) ÷ *competition*

There are plenty of alternatives to this basic formula, but I like it for ease of use and simplicity. Once I've added this column, I then sort the data by this value (largest to smallest) and I then only usually need ten to fifteen keywords at most to give me plenty of ideas to work with.

This is a slightly involved but effective methodology for keyword research, as what you're left with is a list of keywords that both Google and you consider to be *relevant* to the content of your website. And relevance is an important concept in SEO.

Real SEO keyword research is about making sure that your customers, website and Google are all in agreement and alignment over the content of your website. Other sources of inspiration and ideas include having a look at what terms your competition are targeting, Google Trends and, of course, Google Suggest. If you're not sure where to find these things, you can probably work out where to search for them!

If you want to dive further into understanding your current search engine status, search for some of the better keywords that you just discovered and see where you rank compared to your competition. Note that it's vital to avoid Google serving up personalised results, so either use the privacy, incognito or anonymous mode of your browser for the searches, or use a browser that you don't normally use. I hope this is Internet Explorer. If what you find isn't great, don't despair: everything in SEO is fixable (terms and conditions may apply).

PUTTING IT ALL TOGETHER

You should now have a good idea of where things stand with your current search engine traffic, and a solid list of keywords that you're not getting visitors for but very much want.

All that's left now is to work out how to use these keywords. But before we do, let's take a quick step back.

If you have in any way kept up with what's been happening in SEO over the last couple of years, you'll have probably heard about Google updates with names like Panda, Hummingbird, Phantom, Pirate and more.

I won't go into the technical details of what Google is doing, but it is important to understand why they're trying to do it. At the most basic level, Google understands that there's a very real problem with people who are trying manipulate its index. In response to this, Google is trying to clean up its results. They don't want people getting fed up with bad results and considering other options – have you even tried Bing?

This is extremely important. Remember earlier when I said that 70% of SEO was easy? That rule still applies. So, for example, if you have a list of keywords that you know are relevant to what you sell, then all you need to do is create great content for them. Incredibly, that's all there is to it (terms and conditions apply again, unfortunately – see below).

There is, however, one simple rule to be consistently followed without exception: that the content you create should not only be good quality and completely original, but it should also be written primarily for the human visitor and not the search engine spider.

In other words, if you create some fantastic content for a keyword like "choosing a small business HR service", then the article should not only make perfect sense if read out loud (as opposed to the same phrase being repeated fifteen times), but also provide real value to the person reading it.

So the process is simple:

1. Choose your keywords
2. Create spectacular content

WAIT. IS IT REALLY THAT SIMPLE?

Unfortunately there's a lot more to the other 30% of SEO than just creating great content and waiting for the visitors. There are issues like helping Google understand the content on your pages and website, incoming links, page authority, domain authority, usage patterns, spam factors, canonical issues and much more.

But there's the often overlooked fact about Google: it actually does a reasonable job of working out what's on your website and (to some extent) understanding the gist of it. If you've never done any SEO on your website but still get some traffic from Google, this is why.

Even without dabbling in the other 30% of SEO, by creating the right content for the right visitors using the precise language and terminology that your potential customers are using, you're significantly better off than your competition. And you can only gain from this.

When you've checked this off your to-do list and made it an ingrained part of your content creation process, then you're ready to delve into the other 30% of SEO. The not-so-easy side.

Until then, work on understanding your current situation, exploring the opportunities, creating a list of good keywords, creating the right content for them, and starting 2015 with a little bit of smart, safe and real SEO.

ABOUT THE AUTHOR

Dave Collins is the founder of SoftwarePromotions, and contrary to his youthful charm has been working in online marketing since 1997. Dave oversees the SEO and marketing work for all of their clients, and has a very unhealthy obsession with data and trends. Really. He's also an awful rock climber, amateur photographer and a slave to his wonderful children. He wishes he could eat more and exercise less.

16. An Overview of SVG Sprite Creation Techniques

Sara Soueidan 24ways.org/201416

SVG can be used as an icon system to replace icon fonts. The reasons why SVG makes for a superior icon system are numerous, but we won't be going over them in this article. If you don't use SVG icons and are interested in knowing why you may want to use them, I recommend you check out "Inline SVG vs Icon Fonts" by Chris Coyier – it covers the most important aspects of both systems and compares them with each other to help you make a better decision about which system to choose.

Once you've made the decision to use SVG instead of icon fonts, you'll need to think of the best way to optimise the delivery of your icons, and ways to make the creation and use of icons faster.

Just like bitmaps, we can create image sprites with SVG – they don't look or work exactly alike, but the basic concept is pretty much the same.

There are several ways to create SVG sprites, and this article will give you an overview of three of them. While we're at it, we're going to take a look at some of the available tools used to automate sprite creation and fallback for us.

PREREQUISITES

The content of this article assumes you are familiar with SVG. If you've never worked with SVG before, you may want to look at some of the introductory tutorials covering SVG syntax, structure and embedding techniques. I recommend the following:

- SVG basics: Using SVG.
- Structure: Structuring, Grouping, and Referencing in SVG — The `<g>`, `<use>`, `<defs>` and `<symbol>` Elements. We'll mention `<use>` and `<symbol>` quite a bit in this article.
- Embedding techniques: Styling and Animating SVGs with CSS. The article covers several topics, but the section linked focuses on embedding techniques.
- A compendium of SVG resources compiled by Chris Coyier — contains resources to almost every aspect of SVG you might be interested in.

And if you're completely new to the concept of spriting, Chris Coyier's CSS Sprites explains all about them.

Another important SVG feature is the `viewBox` attribute. For some of the techniques, knowing your way around this attribute is not required, but it's definitely more useful if you understand – even if just vaguely – how it works. The last technique mentioned in the article requires that you do know the attribute's syntax and how to use it. To learn all about `viewBox`, you can refer to my blog post about SVG coordinate systems.

With the prerequisites in place, let's move on to spriting SVGs!

BEFORE YOU SPRITE…

In order to create an SVG sprite with your icons, you'll of course need to have these icons ready for use.

Some spriting tools require that you place your icons in a folder to which a certain spriting process is to be applied. As such, for all of the upcoming sections we'll work on the assumption that our SVG icons are placed in a folder named *SVG*.

Each icon is an individual *.svg* file.

You'll need to make sure each icon is well-prepared and optimised for use – make sure you've cleaned up the code by running it through one of the optimisation tools or processes available (or doing it manually if it's not tedious).

After prepping the icon files and placing them in a folder, we're ready to create our SVG sprite.

HTML INLINE SVG SPRITES

Since SVG is XML code, it can be embedded inline in an HTML document as a *code island* using the <svg> element. Chris Coyier wrote about this technique first on CSS-Tricks.

The embedded SVG will serve as a container for our icons and is going to be the actual sprite we're going to use. So we'll start by including the SVG in our document.

```
<!DOCTYPE html>
<!-- HTML document stuff -->

<svg style="display:none;">
  <!-- icons here -->
</svg>

<!-- other document stuff -->
</html>
```

Next, we're going to place the icons inside the `<svg>`. Each icon will be wrapped in a `<symbol>` element we can then reference and use elsewhere in the page using the SVG `<use>` element. The `<symbol>` element has many benefits, and we're using it because it allows us to define a symbol (which is a convenient markup for an icon) without rendering that symbol on the screen. **The elements defined inside `<symbol>` will only be rendered when they are referenced – or called – by the `<use>` element.**

Moreover, `<symbol>` can have its own `viewBox` attribute, which makes it possible to control the positioning of its content inside its container at any time.

Before we move on, I'd like to shed some light on the `style="display:none;"` part of the snippet above. Without setting the `display` of the SVG to `none`, and even though its contents are not rendered on the page, the SVG will still take up space in the page, resulting in a big empty area. In order to avoid that, we're hiding the SVG entirely with CSS.

Now, suppose we have a Twitter icon in the icons folder. *twitter.svg* might look something like this:

```
<!-- twitter.svg -->
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
```

```
xmlns:xlink="http://www.w3.org/1999/xlink" width="32"
height="32" viewBox="0 0 32 32">
<path d="M32 6.076c-1.177 0.522-2.443 0.875-3.771 1.034
1.355-0.813 2.396-2.099 2.887-3.632-1.269 0.752-2.674
1.299-4.169
1.593-1.198-1.276-2.904-2.073-4.792-2.073-3.626 0-6.565
2.939-6.565 6.565 0 0.515 0.058 1.016 0.17
1.496-5.456-0.274-10.294-2.888-13.532-6.86-0.565
0.97-0.889 2.097-0.889 3.301 0 2.278 1.159 4.287 2.921
5.465-1.076-0.034-2.088-0.329-2.974-0.821-0.001
0.027-0.001 0.055-0.001 0.083 0 3.181 2.263 5.834 5.266
6.437-0.551 0.15-1.131 0.23-1.73 0.23-0.423
0-0.834-0.041-1.235-0.118 0.835 2.608 3.26 4.506 6.133
4.559-2.247 1.761-5.078 2.81-8.154 2.81-0.53
0-1.052-0.031-1.566-0.092 2.905 1.863 6.356 2.95 10.064
2.95 12.076 0 18.679-10.004 18.679-18.68
0-0.285-0.006-0.568-0.019-0.849 1.283-0.926 2.396-2.082
3.276-3.398z" fill="#000000"></path>
</svg>
```

We don't need the root `svg` element, so we'll strip the code and only keep the parts that make up the Twitter icon's shape, which in this example is just the `<path>` element.Let's drop that into the sprite container like so:

```
<svg style="display:none;">
  <symbol id="twitter-icon" viewBox="0 0 32 32">
    <path d="M32 6.076c-1.177 …" fill="#000000"></path>
  </symbol>

 <!-- remaining icons here -->
  <symbol id="instagram-icon" viewBox="0 0 32 32">
   <!-- icon contents -->
```

```
  </symbol>

  <!-- etc. -->
</svg>
```

Repeat for the other icons.

The value of the `<symbol>` element's `viewBox` attribute depends on the size of the SVG. You don't need to know how the `viewBox` works to use it in this case. Its value is made up of four parts: the first two will almost always be "`0 0`"; the second two will be equal to the size of the icon. For example, our Twitter icon is 32px by 32px (see *twitter.svg* above), so the `viewBox` value is "`0 0 32 32`".

That said, it is certainly useful to understand how the `viewBox` works – it can help you troubleshoot SVG sometimes and gives you better control over it, allowing you to scale, position and even crop SVGs manually without having to resort to an editor. My blog post explains all about the `viewBox` attribute and its related attributes.

Once you have your SVG sprite ready, you can display the icons anywhere on the page by referencing them using the SVG `<use>` element:

```
<svg class="twitter-icon">
  <use xlink:href="#twitter-icon"></use>
<svg>
```

And that's all there is to it!

HTML-inline SVG sprites are simple to create and use, but when you have a lot of icons (and the more icon sets you create) it can easily become daunting if you have to manually transfer the icons into the <svg>. Fortunately, you don't have to do that. Fabrice Weinberg created a Grunt plugin called grunt-svgstore which takes the icons in your *SVG* folder and generates the SVG sprites for you; all you have to do is just drop the sprites into your page and use the icons like we did earlier.

This technique works in all browsers supporting SVG. There seems to be a bug in Safari on iOS which causes the icons not to show up when the SVG sprite is defined at the bottom of the document *after* the <use> references to the icons, so it's safest to include the sprite before you use the icons until this bug is fixed.

This technique has one disadvantage: the SVG sprite cannot be cached. We're saving an extra HTTP request here but the browser cannot cache the image, so we aren't speeding up any subsequent page loads by inlining the SVG. There must be a better way – and there is.

Styling the icons is possible, but getting deep into the styles becomes a bit harder owing to the nature of the contents of the <use> element – these contents are cloned into a shadow DOM, and hence selecting elements in CSS the traditional way is not possible. However, some

techniques to work around that do exist, and give us slightly more styling flexibility. Animations work as expected.

Referencing an external SVG sprite in HTML

Instead of including the SVG inline in the document, you can reference the sprite and the icons inside it externally, taking advantage of fragment identifiers to select individual icons in the sprite.

For example, the above reference to the Twitter icon would look something like this instead:

```
<svg class="twitter-icon">
  <use xlink:href="path/to/icons.svg#twitter-icon"></use>
<svg>
```

`icons.svg` is the name of the SVG file that contains all of our icons as `symbols`, and the fragment identifier `#twitter-icon` is the reference to the `<symbol>` wrapping the Twitter icon's contents. Very convenient, isn't it? The browser will request the sprite and then cache it, speeding up subsequent page loads. Win!

This technique also works in all browsers supporting SVG **except Internet Explorer** – not even IE9+ with SVG support permits this technique. No version of IE supports referencing an external SVG in `<use>`.

Fortunately (again), Jonathan Neil has created a plugin called svg4everybody which fills this gap in IE; you can reference an external sprite in <use> *and* also provide fallback for browsers that do not support SVG. However, it requires you to have the fallback images (PNG or JPEG, for example) available to do so. For details, refer to the plugin's Github repository's *readme* file.

CSS INLINE SVG SPRITES

Another way to create an SVG sprite is by inlining the SVG icons in a style sheet using data URIs, and providing fallback for non-supporting browsers – also within the CSS.

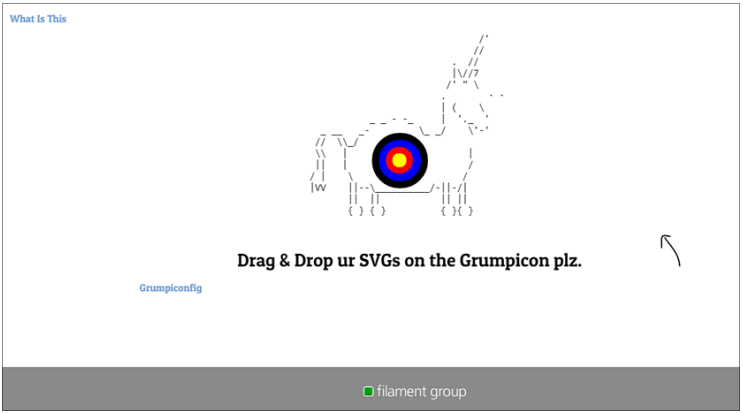
Using this approach, **we're turning the style sheet into the sprite** that includes our icons. The style sheet is normally cached by the browser, so we have that concern out of the way.

This technique is put into practice in Filament Group's icon system approach, which uses their Grunticon plugin – or its sister Grumpicon web app – for generating the necessary CSS for the sprite. As such, we're going to cover this technique by following a workflow that uses one of these tools.

Again, we start with our icon SVG files. To focus on the actual spriting method and not on the tooling, I'll go over the process of sprite creation using the Grumpicon web

app, instead of the Grunticon plugin. Both tools generate the same resources that we're going to use for the icon system. Whether you choose the web app or the Grunt set-up, after processing your *SVG* folder you're going to end up with the same set of resources that we'll be using throughout this section.

The first step is to drop your icons into the Grumpicon web app.



Grumpicon homepage screenshot.

The application will then show you a preview of your icons, and a download button will allow you to download the generated files. These files will contain everything you need for your icon system – all that's left is for you to drop the generated files and code into your project as recommended and you'll have your sprite and icons ready to use anywhere you want in your page.

Grumpicon generates five files and one folder in the downloaded package: a *png* folder containing PNG versions of your icons; three style sheets (that we'll go over briefly); a *loader* script file; and *preview.html* which is a live example showing you the other files in action.

The script in the *loader* goes into the `<head>` of your page. This script handles browser and feature detection, and requests the necessary style sheet depending on browser support for SVG and base64 data URIs. If you view the source code of the preview page, you can see exactly how the script is added.

icons.data.svg.css is the style sheet that contains your icons – the sprite. The icons are embedded inline inside the style sheet using data URIs, and applied to elements of your choice as background images, using class names. For example:

```
.twitter-icon{
    background-image: url('data:image/svg+xml;…'); /*
the ellipsis is where the icon's data would go */
    background-repeat: no-repeat;
    background-position: 50% 50%;
    height: 2em;
    width: 2em;
    /* etc. */
}
```

Then, you only have to apply the `twitter-icon` class name to an element in your HTML to apply the icon as a background to it:

```
<span class="twitter-icon"></span>
```

And that's all you need to do to get an icon on the page.

icons.data.svg.css, along with the other two style sheets **and** the *png* folder should be added to your CSS folder.

icons.data.png.css is the style sheet the script will load in browsers that don't support SVG, such as IE8. Fallback for the inline SVG is provided as a base64-encoded PNG. For instance, the fallback for the Twitter icon from our example would look like so:

```
.twitter-icon{
    background-image: url('data:image/png;base64;…');
    /* etc. */
}
```

icons.fallback.css is the style sheet required for browsers that don't support base64-encoded PNGs – the PNG images are loaded as usual using the image's URL. The script will load this style sheet for IE6 and IE7, for example.

```
.twitter-icon{
    background-image: url(png/twitter-icon.png);
    /* etc. */
}
```

This technique is very different from the previous one. The sprite in this case is literally the style sheet, not an SVG container, and the icon usage is very similar to that of a CSS sprite – the icons are provided as background images.

This technique has advantages and disadvantages. For the sake of brevity, I won't go into further details, but the main limitations worth mentioning are that SVGs embedded as background images cannot be styled with CSS; and animations are restricted to those defined inside the <svg> for each icon. CSS interactions (such as hover effects) don't work either. Thus, to apply an effect for an icon that changes its color on hover, for example, you'll need to export a set of SVGs for each colour in order for Grumpicon to create matching fallback PNG images that can then be used for the animation.

For more details about the Grumpicon workflow, I recommend you check out "A Designer's Guide to Grumpicon" on Filament Group's website.

USING SVG FRAGMENT IDENTIFIERS AND VIEWS

This spriting technique is, again, different from the previous ones, and it is my personal favourite.

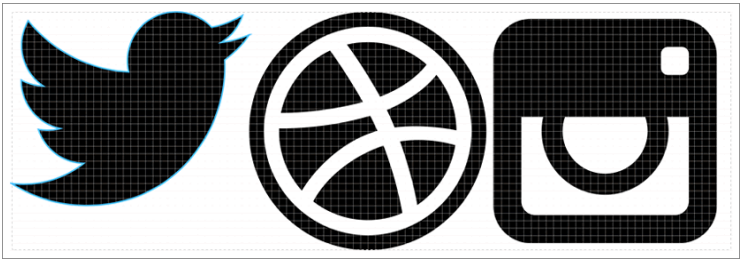
SVG comes with a standard way of cropping to a specific area in a particular SVG image. If you've ever worked with CSS sprites before then this definitely sounds familiar: it's almost exactly what we do with CSS sprites – the image containing all of the icons is cropped, so to speak, to show only the one icon that we want in the background positioning area of the element, using background size and positioning properties.

Instead of using background properties, we'll be using SVG's `viewBox` attribute to crop our SVG to the specific icon we want.

What I like about this technique is that it is more visual than the previous ones. Using this technique, the SVG sprite is treated like an actual image containing other images (the icons), instead of treating it as a piece of code containing other code.

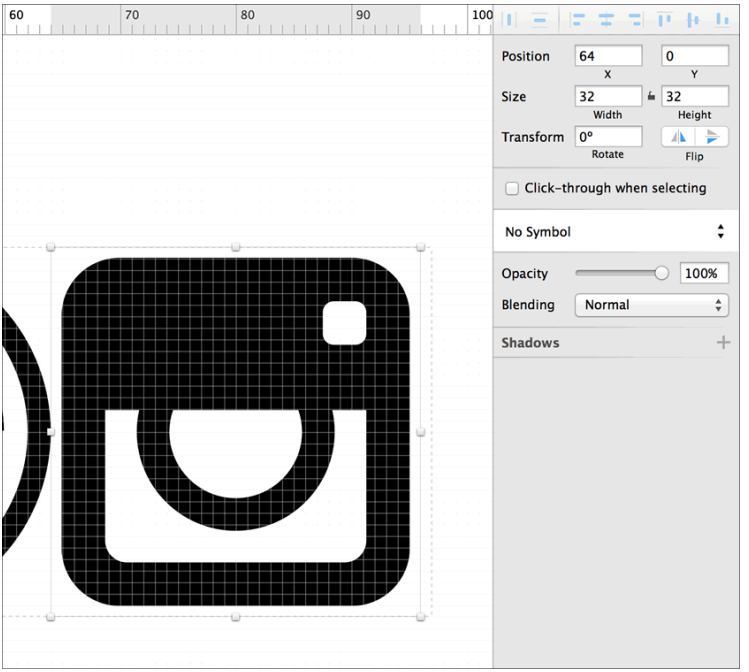
Again, our SVG icons are placed inside a main SVG container that is going to be our SVG sprite. If you're working in a graphics editor, position or arrange your icons inside the canvas any way you want them to be, and then export the graphic as is. Of course, the less empty space there is in your SVG, the better.

In our example, the sprite contains three icons as shown in the following image. The sprite is open in Sketch. Notice how the SVG is just big enough to fit the icons inside it. It doesn't have to be like this, but it's cleaner this way.



Screenshot showing the SVG sprite containing our icons.

Now, suppose you want to display only the Instagram icon. Using the SVG `viewBox` attribute, we can crop the SVG to the icon. The Instagram icon is positioned at 64px along the positive x-axis, and zero pixels along the y-axis. It is also 32px by 32px in size.



Screenshot showing the position (offset) of the Instagram icon inside the SVG sprite, and its size.

Using this information, we can specify the value of the `viewBox` as: `64 0 32 32`. This area of the view box contains *only* the Instagram icon. `64 0` specifies the top-left corner of the view box area, and `32 32` specify its dimensions.

Now, if we were to change the `viewBox` value on the SVG sprite to this value, only the Instagram icon will be visible inside the SVG viewport. Great. But how do we use this information to display the icon in our page using our sprite?

SVG comes with a native way to link to portions or areas of an image using **fragment identifiers**. Fragment identifiers are used to link into a particular view area of an SVG document. Thus, using a fragment identifier and the boundaries of the area that we want (from the `viewBox`), we can link to that area and display it.

For example, if you want to display the icon from the sprite using an `<img>` tag, you can reference the icon in the sprite like so:

```
<img src='uiIcons.svg#svgView(viewBox(64, 0, 32, 32))'
alt="Settings icon"/>
```

The fragment identifier in the snippet above (`#svgView(viewBox(64, 0, 32, 32))`) is the important part. This will result in only the Instagram icon's area of the sprite being displayed.

There is also another way to do this, using the SVG `<view>` element. The `<view>` element can be used to define a view area and then reference that area somewhere else. For example, to define the view box containing the Instagram icon, we can do the following:

```
<view id='instagram-icon' viewBox='64 0 32 32' />
```

Then, we can reference this view in our `<img>` element like this:

```
<img src='sprite.svg#instagram-icon' alt="Instagram
icon" />
```

The best part about this technique – besides the ability to reference an external SVG and hence make use of browser caching – is that it allows us to use practically any SVG embedding technique and does not restrict us to specific tags.

It goes without saying that this feature can be used for more than just icon systems, owing to `viewBox`'s power in controlling an SVG's viewable area.

SVG fragment identifiers have decent browser support, but the technique is buggy in Safari: there is a bug that causes problems when loading a server SVG file and then using fragment identifiers with it. Bear Travis has documented the issue and a workaround.

WHERE TO GO FROM HERE

Pick the technique that works best for your project. Each technique has its own pros and cons, relating to convenience and maintainability, performance, and styling and scripting. Each technique also requires its own fallback mechanism.

The spriting techniques mentioned here are not the only techniques available. Other methods exist, such as SVG stacks, and others may surface in future, but these are the three main ones today.

The third technique using SVG's built-in `viewBox` features is my favourite, and with better browser support and fewer (ideally, no) bugs, I believe it is more likely to become the standard way to create and use SVG sprites. Fallback techniques can be created, of course, in one of many possible ways.

Do you use SVG for your icon system? If so, which is your favourite technique? Do you know or have worked with other ways for creating SVG sprites?

ABOUT THE AUTHOR

Sara is a freelance front-end web developer from Lebanon — focusing on HTML5, SVG, CSS3 and Javascript. She loves teaching and enjoys breaking down complex subjects into simple, easy-to-understand bits. She writes articles and tutorials on front-end web development on her blog, and for various online magazines including Codrops, where she is an author and team member. She also actively tweets on Twitter at @SaraSoueidan.

17. Content Production Planning

Sophie Dennis 24ways.org/201417

While everyone agrees that getting the content of a website right is vital to its success, unless you're lucky enough to have an experienced editor or content strategist on board, planning content production often seems to fall through the cracks. One reason is that, for most of the team, it feels like someone else's problem. Not necessarily a specific person's problem. Just someone else's. It's only when everyone starts urgently asking when the content is going to be ready, that it becomes clear the answer is, "Not as soon as we'd like it".

The good news is that there are some quick and simple things you can do, even if you're not the official content person on a project, to get everyone on the same content planning page.

Content production planning boils down to answering three deceptively simple questions:

1. What content do you need?
2. How much of it do you need?
3. Who's going to make it?

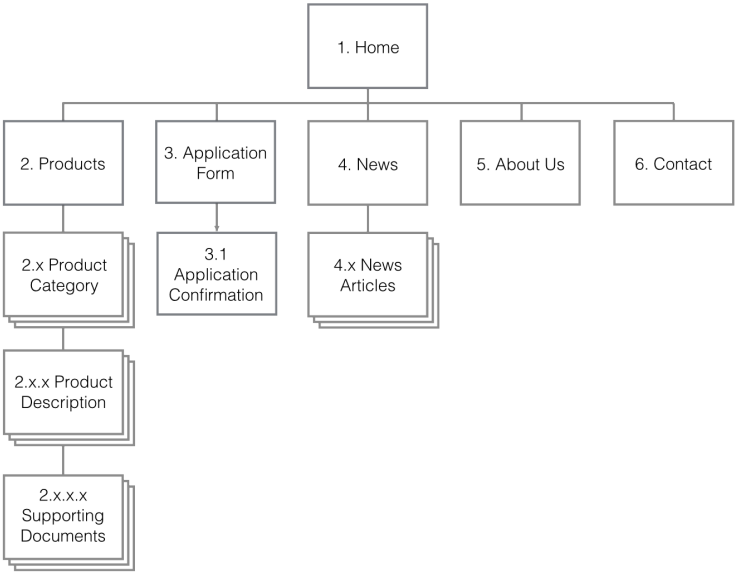
Even if it's not your job to come up with the answers, by asking these questions early enough and agreeing who is going to come up with the answers, you'll be a long way towards avoiding the last-minute content problems which so often plague projects.

HOW MUCH CONTENT DO WE NEED?

People tend to underestimate two crucial things about content: how much content they need, and how long that content takes to produce.

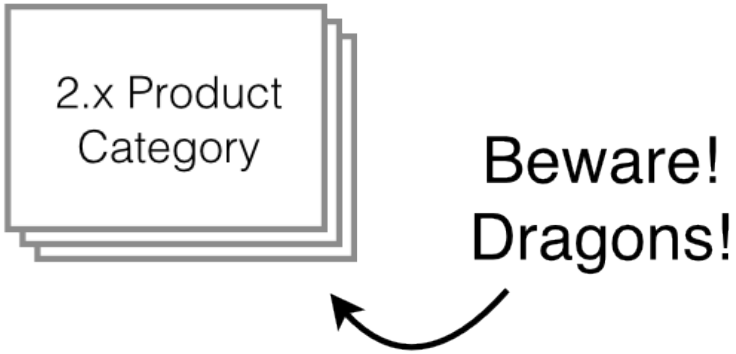
When I ask someone how big their website is – how many pages it contains – I usually double or triple the answer I get. That's because almost everyone's mental model of their website greatly underestimates its true size. You can see the problem for yourself if you look at a site map. Site maps are great at representing a mental model of a website. But because they're a deliberate simplification they naturally lead us to underestimate how much content is involved in populating them.

Several years ago I was asked to help a client create a new microsite (their word) which they wanted ready in two weeks for a conference they were attending. Here's the site map they had in mind. At first glance it looks like a pretty small website. Maybe twenty to thirty pages?



That's what the client thought.

But see those boxes which are multiple boxes stacked on top of one another, for product categories, descriptions and supporting material? They're known as page stacks, and page stacks are the content strategy equivalent of *Here Be Dragons*.



Say we have:

- five product categories
- each with five products
- which all have two or three supporting documents

Those are still fairly small numbers. But small numbers multiplied by other small numbers tend to lead to big numbers.

5 categories = 5 category descriptions

plus

5 categories × 5 products each = 25 product descriptions

plus

25 products × 2.5 (average) supporting documents = 63 supporting documents

equals

93 pages

Suddenly our twenty- or thirty-page website is running towards one hundred.

That's probably enough to get most project teams to sit up and take notice. But there's still the danger of underestimating how long it's going to take to create the content. After all, assuming the supporting documents already exist in some form, there are only about twenty-five to thirty pages of new copy to write.

HOW MUCH WORK IS IT?

Again, we have the problem that small numbers when multiplied by other small numbers tend to lead to big numbers. Let's make a rough guess that it'll take four hours to write each product category and description page we need. That feels a little conservative if we're writing stuff from scratch, but assuming the person doing it already knows the products fairly well it's not unreasonable.

30 pages × 4 hours each = 120 hours

120 hours ÷ 7.5 working hours a day = 16 days

Ouch.

At this point it's pretty clear we're not getting this site launched in two weeks.

THE GOAL IS THE CONVERSATION

By breaking down the site into its content components, and putting some rough estimates on how long each might take to produce, the client instantly realised that there was no way they would be ready to launch it in two weeks. Although we still didn't know exactly when it would be ready, getting to that realisation right at the start of the project was a major win for everybody. Without it, the design agency would have bust a gut to get the design, front-end and CMS all done in double-quick time, only to find it was all for nothing as barely half the content was ready. As it was, an early discussion about content, albeit a brief one, bought everyone time to tackle the project properly, without pulling any long nights or working weekends.

If you haven't been able to get people to discuss content plans for the project, these kinds of rough estimates should give you enough evidence to get everyone to start taking it seriously. Your goal is to get everyone on the project to a place where they are ready to talk in detail about who is going to create this content, and how long it's really going to take them, and to get to those conversations before lack of content becomes a problem.

Be careful though. It's best to talk in ranges and round numbers when your estimates are this uncertain. And watch those multipliers. Given small numbers multiplied by other small numbers lead to big numbers, changing just

one number can greatly change the overall estimate. I like to run a couple of different scenarios to check what things look like if I've under- or overestimated either how many pages we're going to need, or how long they're going to take to create. For example:

Top end: 30 pages × 5 hours = 150 hours, or 20 days

Bottom end: 25 pages × 4 hours = 100 hours, or 13.3 days

So rather than say, "I estimate the content will take around sixteen days to produce", I'm going to say, "I think the content will take about three to four weeks to produce". Even with qualifiers like *estimate* and *around*, sixteen days sounds too precise. Whereas three to four weeks instantly conveys that this is just a rough figure.

WHO'S GOING TO MAKE IT?

So, people tend to underestimate two crucial things about content: how much content they need, and how long content takes to write. At this stage, you're still in danger of the latter, because it's tempting to simply estimate how much time content takes to write (or record, if we're talking audio or visual content), and overlook all the other work that needs to goes on around it.

Take 24 ways as an example. In terms of our three deceptively simple questions: *what* is practical articles about web design; *how many* is twenty-four, one for each day of Advent; and *who* are experts working on the web, one to write each article.

But there's another *who* you might not have considered.

Someone needs to select those authors in the first place, make sure they deliver their articles on time (and find someone to replace them if they don't), review drafts, copy-edit and proofread final versions, upload them to the site, promote them, keep an eye on the comments and make sure there are still presents under the tree on Christmas morning.

Even if each of those tasks only takes an hour or so, it then needs multiplying by twenty-four (except the presents, obviously). And as we've already seen, small numbers multiplied by small numbers quickly turn into much bigger numbers. Just a few hours per article, when multiplied by twenty-four articles, easily multiplies up to days or even weeks of effort.

To get a more accurate estimate of how long the different kinds of content are going to take, you need to break down the content production work into its constituent stages, starting with planning, moving on through the main work of creation, to reviewing, approvals and finally

publishing. You need to think about who needs to be involved at each step, and how much time they'll need to do their bit.

Taken together, these things make up your **content workflow**. The workflow will be different for each organisation, but might look something like this:

1. Eddie the web editor will work out the key messages and objectives for each page, and agree them with Mo the marketing director.
2. Eddie will then get Cal, the copywriter, to write the first draft.
3. As part of that, Cal will interview Sam the subject expert to understand the intricacies of the subject and get all the facts straight.
4. Once Cal's done the first draft, it'll go to Sam to check for accuracy, while Eddie reviews it for style and message.
5. Once Cal has incorporated their feedback it's time to get Mo to have a look at the final draft.
6. If Mo's happy, it'll get a final proofread, be uploaded to the CMS, and Mo will give the final sign-off and release it for publishing.

You can plot this on a table, with the stages of the content production process down the side, and the key roles or personnel along with top. Then the team can estimate how much time they think each of them needs at each stage.

	Mo (marketing director)	Sam (subject expert)	Eddie (web editor)	Cal (copywriter)
Outline: define key messages and objectives			30 min	
Review outline	15 min			
First draft		30 min		3 hours
Review 1st draft		30 min	30 min	
2nd draft				1 hour
Review 2nd draft	15 min	15 min	15 min	
Final amendments				30 min
Proofread			15 min	
Upload				15 min
Sign-off	10 min			
TOTAL	**40 min**	**1 hour 15 min**	**1 hour 30 min**	**4 hours 45 min**

You can then bring out your calculator again, and come up with some more big scary numbers showing how much time it's going to take for the whole team to get all the content needed not just written, but also planned, reviewed, approved and published.

With an experienced team you can run this exercise as a group workshop and get some fairly accurate estimates pretty quickly. If this is all a bit new to you, check out Gather Content's *Content Production Planning for Agencies*

ebook for a useful guide to common content roles, ballpark estimates for how much time each one needs on a typical piece of content, and how to run a process and estimating workshop to dig into them in more detail.

On a small team, one person might play many roles, but you should still sanity-check your estimates by breaking down the process and putting a rough estimate on each stage. With only a couple of people involved, it's even easier to only include the core activity like writing or recording in your estimates, and forget to allow time for the planning, reviewing, proofreading, publishing and promoting you'll still need to do. And even in a team of one, if at all possible you should find at least one other person to act as a second pair of eyes, and give anything you produce a quick once-over and proofread before it's published.

Depending on the kind of content you're making, you should also consider what will happen after it's published. The full content life cycle should include promotion, monitoring and regular reviews to make sure content stays accurate and up to date. Making sure you have the time and resources available to do all those things for each piece of content is essential for creating a sustainable content programme.

THE PROOF OF THE PUDDING

Even after digging into workflow and getting the whole team involved in estimating, you're still largely in the realm of the guesstimate. The good news, though, is that you can quite quickly start finding out if your guesstimates are right or not. As soon as you can, pilot the production process with some real content. This is a double-win: you start finding out how long it really takes to produce all this fab new content, and you get real content to work with in designs and prototypes.

Once you've run a few things through your process, you'll be able to refine your estimates, confirm your workflow, and give everyone involved a clear idea of when it will all be ready, and what you need from them.

KEEPING IT ALL ON TRACK

At this point I like to pull everything together into the content strategist's favourite tool: the spreadsheet.

A simple content production checklist is a bit like a content inventory or audit, but for the content you don't yet have, not the stuff already done. You can grab an example here.

Each piece of content gets its own row, with columns for basic information like page title, ID (which should match the site map), and who's responsible for making it. You can

capture simple details like target audience and key messages here too, though for more complex content, page description tables like those described by Relly Annett-Baker in "Extracting the Content" may be a better tool to use. Just adapt these columns to whatever makes sense for your content.

I then have columns to track where each piece is in the production process. I usually keep this simple, with a column each to mark whether it's draft, final or uploaded. The status column on the left automatically shows the item's status, using a simple traffic light colour scheme for whether the item is still to do (red), in draft (amber), or done (green). Seeing the whole thing slowly turn from red to green is a nice motivator.

If you want to track the workflow in more detail, a kanban board in a tool like Trello is a great way for a team to collaborate on content production, track each item's progress, and keep an eye out for bottlenecks and delays.

GETTING TO THE CONTENT STRATEGY CONVERSATION

It's a relatively simple exercise, then, to decide not just what kinds of pages you need, but also how many of them: put some rough estimates of effort on the tasks needed to create those pages – not just the writing, but all the other stages of planning, reviewing, approving, publishing and

promoting – and then multiply all those things together. This will quickly bring some reality to grand visions and overambitious plans. Do it early enough, and even when the final big scary number is a lot bigger and scarier than everyone thought, you'll still have time to do something about it.

As well as getting everyone on board for some proper content planning activities, that big scary number is your opportunity to get to the real core questions of content strategy: do we really need all this content? Where can existing content be reused and repurposed? How do we prioritise our efforts? What really matters to our readers and users?

Time and again, case studies show that less content delivers more: more leads, more sales, more self-service support and savings in the call centre. Although that argument is primarily one you should make from a good-for-the-users perspective, it doesn't hurt to be able to make it from the cheaper-for-the-business perspective as well, and to have some big scary numbers to back that up.

ABOUT THE AUTHOR

Sophie Dennis consults on content and UX strategy, and leads blended creative / technical teams (mostly agile ones) on a freelance or contract basis. She loves working with other smart, creative people to deliver digital products that make a real difference both to users and clients. Recently she's worked on major digital projects for cxpartners, the Office for National Statistics, Bristol City Council, the National Trust, Jisc, Bloomsbury Publishing and the University of Surrey. She lives in

the rural South West of England, where she runs the Digpen.com grassroots web conference with husband Andy Robinson.

18. A Holiday Wish

Jeffrey Zeldman 24ways.org/201418

A friend and I were talking the other day about why clients spend more on toilet cleaning than design, and how the industry has changed since the mid-1990s, when we got our starts. Early in his career, my friend wrote a fine CSS book, but for years he has called himself a UX designer. And our conversation got me thinking about how I reacted to that title back when I first started hearing it.

"Just what this business needs," I said to myself, "another phony expert."

Okay, so I was wrong about UX, but my touchiness was not altogether unfounded. In the beginning, our industry was divided between freelance jack-of-all-trade punks, who designed and built and coded and hosted and Photoshopped and even wrote the copy when the client couldn't come up with any, and snot-slick dot-com mega-agencies that blew up like Alice and handed out titles like impoverished nobles in the years between the world wars.

I was the former kind of designer, a guy who, having failed or just coasted along at a cluster of other careers, had suddenly, out of nowhere, blossomed into a web designer—an immensely curious designer slash coder slash writer with a near-insatiable lust to shave just one more byte from every image. We had modems back then, and I dreamed in sixteen colors. My source code was as pretty as my layouts (arguably prettier) and I hoovered up facts and opinions from newsgroups and bulletin boards as fast as any loudmouth geek could throw them. It was a beautiful life.

But soon, too soon, the professional digital agencies arose, buying loft buildings downtown, jacking in at T1 speeds, charging a hundred times what I did, and communicating with their clients in person, in large artfully bedecked rooms, wearing hand-tailored Barney's suits and bringing back the big city bullshit I thought I'd left behind when I quit advertising to become a web designer.

Just like the big bad ad agencies of my early career, the new digital agencies stocked every meeting with a totem pole worth of ranks and titles. If the client brought five upper middle managers to the meeting, the agency did likewise. If fifteen stakeholders got to ask for a bigger logo, fifteen agency personnel showed up to take notes on the percentage of enlargement required.

But my biggest gripe was with the titles.

The bigger and more expensive the agency, the lousier it ran with newly invented titles. Nobody was a designer any more. Oh, no. Designer, apparently, wasn't good enough. Designer was not what you called someone you threw that much money at.

Instead of designers, there were user interaction leads and consulting middleware integrators and bilabial experience park rangers and you name it. At an AIGA Miami event where I was asked to speak in the 1990s, I once watched the executive creative director of the biggest dot-com agency of the day make a presentation where he spent half his time bragging that the agency had recently shaved down the number of titles for people who basically did design stuff from forty-six to just twenty-three—he presented this as though it were an Einsteinian coup—and the other half of his time showing a film about the agency's newly opened branch in Oslo. The Oslo footage was shot in December. I kept wondering which designer in the audience who lived in the constant breezy balminess of Miami they hoped to entice to move to dark, wintry Norway. But I digress.

Shortly after I viewed this presentation, the dot-com world imploded, brought about largely by the euphoric excess of the agencies and their clients. But people still needed websites, and my practice flourished—to the point

where, in 1999, I made the terrifying transition from guy in his underwear working freelance out of his apartment to head of a fledgling design studio. (Note: you never stop working on that change.)

I had heard about experience design in the 1990s, but assumed it was a gig for people who only knew one font.

But sometime around 2004 or 2005, among my freelance and small-studio colleagues, like a hobbit in the Shire, I began hearing whispers in the trees of a new evil stirring. The fires of Mordor were burning. Web designers were turning in their HTML editing tools and calling themselves UXers.

I wasn't sure if they pronounced it "uck-sir," or "you-ex-er," but I trusted their claims to authenticity about as far as I trusted the actors in a Doctor Pepper commercial when they claimed to be Peppers. I'm an UXer, you're an UXer, wouldn't you like to be an UXer too? No thanks, said I. I still make things. With my hands.

Such was my thinking. I may have earned an MFA at the end of some long-past period of soul confusion, but I have working-class roots and am profoundly suspicious of, well, everything, but especially of anything that smacks of pretense. I got exporting GIFs. I didn't get how white papers and bullet points helped anybody do anything.

I was wrong. And gradually I came to know I was wrong. And before other members of my tribe embraced UX, and research, and content strategy, and the other airier consultant services, I was on board. It helped that my wife of the time was a librarian from Michigan, so I'd already bought into the cult of information architecture. And if I wasn't exactly the seer who first understood how borderline academic practices related to UX could become as important to our medium and industry as our craft skills, at least I was down a lot faster than Judd Apatow got with feminism. But I digress.

I love the web and all the people in it. Today I understand design as a strategic practice above all. The promise of the web, to make all knowledge accessible to all people, won't be won by HTML5, WCAG 2, and responsive web design alone.

We are all designers. You may call yourself a front-end developer, but if you spend hours shaving half-seconds off an interaction, *that's* user experience and *you*, my friend, are a designer. If the client asks, "Can you migrate all my old content to the new CMS?" and you answer, "Of course we *can*, but *should* we?", you are a designer. Even our users are designers. Think about it.

Once again, as in the dim dumb dot-com past, we seem to be divided by our titles. But, O, my friends, our varied titles are only differing facets of the same bright gem. Sisters, brothers, we are all designers. Love on! Love on!

And may all your web pages, cards, clusters, clumps, asides, articles, and relational databases be bright.

ABOUT THE AUTHOR

Jeffrey Zeldman is the founder and executive creative director of Happy Cog™, an agency of web design specialists, and the co-founder (with Eric Meyer) of An Event Apart.

In 1995, the former art director and copywriter launched one of the first personal sites (Jeffrey Zeldman Presents) and began publishing web design tutorials. In 1998 he co-founded (and for several years led) The Web Standards Project, a grassroots coalition that brought standards to our browsers. That same year, he launched *A List Apart* "for people who make websites."

Jeffrey has written many articles and two books, notably the foundational web standards text *Designing With Web Standards*, now in its third edition.

Photo: John Morrison

19. Why You Should Design for Open Source

Jina Bolton 24ways.org/201419

Let's be honest. Most designers don't like working for nothing. We rally against spec work and make a stand for contracts and getting paid. That's totally what you should do as a professional designer in the industry. It's your job. It's your hard-working skill. It's your bread and butter. *Get paid*.

However, I'm going to make a case for why you could also consider designing for open source. First, I should mention that not all open source work is free work. Some companies hire open source contributors to work on their projects full-time, usually because that project is used by said company. There are other companies that encourage open source contribution and even offer 20%-time for these projects (where you can spend one day a week contributing to open source). These are super rad situations to be in. However, whether you're able to land a

gig doing this type of work, or you've decided to volunteer your time and energy, designing for open source can be rewarding in many other ways.

PORTFOLIO BUILDING

New designers often find themselves in a catch-22 situation: they don't have enough work experience showcased in their portfolio, which leads to them not getting much work because their portfolio is bare. These new designers often turn to unsolicited redesigns to fill their portfolio. An unsolicited redesign is a proof of concept in which a designer attempts to redesign a popular website. You can see many of these concepts on sites like Dribbble and Behance and there are even websites dedicated to showcasing these designs, such as Uninvited Designs. There's even a subreddit for them.

There are quite a few negative opinions on unsolicited redesigns, though some people see things from both sides. If you feel like doing one or two of these to fill your portfolio, that's of course up to you. But here's a better suggestion. Why not contribute design for an open source project instead?

You can easily find many projects in great need of design work, from branding to information design, documentation, and website or application design. The benefits to doing this are far better than an unsolicited

redesign. You get a great portfolio piece that actually has greater potential to get used (especially if the core team is on board with it). It's a win-win situation.

Not all designers are in need of portfolio filler, but there are other benefits to contributing design.

GIVING BACK TO THE COMMUNITY

My first experience with voluntary work was when I collaborated with my friend, Vineet Thapar, on a pro bono project for the W3C's Web Accessibility Initiative redesign project back in 2004. I was very excited to contribute CSS to a website that would get used by the W3C! Unfortunately, it decided to go a different direction and my work did not get used. However, it was still pretty exciting to have the opportunity, and I don't regret a moment of that work. I learned a lot about accessibility from this experience and it helped me land some of the jobs I've had since.

Almost a decade later, I got super into Sass. One of the core maintainers, Chris Eppstein, lamented on Twitter one day that the Sass website and brand was in dire need of design help. That led to the creation of an open source task force, Team Sass Design, and we revived the brand and the website, which launched at SassConf in 2013.

It helped me in my current job. I showed it during my portfolio review when I interviewed for the role. Then I was able to use inspiration from a technique I'd tried on the Sass website to help create the more feature-rich design system that my team at work is building. But most importantly, I soon learned that it is exhilarating to be a part of the Sass community. This is the biggest benefit of all. It feels really good to give back to the technology I love and use for getting my work done.

Ben Werdmuller writes about the need for design in open source. It's great to see designers contributing to open source in awesome ways. When A List Apart's website went open source, Anna Debenham contributed by helping build its pattern library. Bevan Stephens worked with FontForge on the design of its website. There are also designers who have created their own open source projects. There's Dan Cederholm's Pears, which shares common patterns in markup and style. There's also Brad Frost's Pattern Lab, which shares his famous method of atomic design and applies it to a design system. These systems and patterns have been used in real-world projects, such as RetailMeNot, so designers have contributed to the web in an even larger way simply by putting their work out there for others to use. That's kind of fun to think about.

HOW TO GET STARTED

So are you stoked about getting into the open source community? That's great!

Initially, you might get worried or uncomfortable in getting involved. That's okay. But first consider that the project is open source for a reason. Your contribution (no matter how large or small) can help in a big way.

If you find a project you're interested in helping, make sure you do your research. Sometimes project team members will be attached to their current design. Is there already a designer on the core team? Reach out to that designer first. Don't be too aggressive with why you think your design is better than theirs. Rather, offer some constructive feedback and a proposal of what would make the design better. Chances are, if the designer cares about the project, and you make a strong case, they'll be up for it.

Are there contribution guidelines? It's proper etiquette to read these and follow the community's rules. You'll have a better chance of getting your work accepted, and it shows that you take the time to care and add to the overall quality of the project. Does the project lack guidelines? Consider starting a draft for that before getting started in the design.

When contributing to open source, use your initiative to solve problems in a manageable way. Huge pull requests are hard to review and will often either get neglected or rejected. Work in small, modular, and iterative contributions.

So this is my personal take on what I've learned from my experience and why I love open source. I'd love to hear from you if you have your own experience in doing this and what you've learned along the way as well. Please share in the comments!

Thanks Drew McLellan, Eric Suzanne, Kyle Neath for sharing their thoughts with me on this!

ABOUT THE AUTHOR

Jina Bolton is a Senior Product Designer at Salesforce UX, where she helps design and develop systems for enterprise software. She also loves Sass; she leads Team Sass Design, an open source task force that redesigned the Sass brand and website. Jina also organizes the San Francisco Sass Meet Up, The Mixin. She coauthored two books, Fancy Form Design and The Art & Science of CSS. Previously, she has worked with rad companies including Apple, Engine Yard, and Crush + Lovely.

Photo: Nick Howland

20. Meet for Learning

Leslie Jensen-Inman 24ways.org/201420

"I've never worked in a place like this," said one of my direct reports during our daily stand-up meeting.

And with that statement, my mind raced to the most important thing about lawyering that I've learned from decades of watching lawyers lawyer on TV: *don't ask a question you don't know the answer to*.

But I couldn't stop myself. I wanted to learn more. The thought developed in my mind. The words formed in my mouth. And the vocalization occurred: "A place like this?"

"I've never worked where people are so honest and transparent about things."

DESIGNING A LEARNING-CENTERED CULTURE

Before we started Center Centre, Jared Spool and I discussed both the larger goals and the smaller details of this new UX design school. We talked about things like user experience, curriculum, and structure.

We discussed the pattern we saw in our research. Hiring managers told us time and again that great designers have excellent technical and interpersonal skills. But, more importantly, the *best* designers are lifelong learners—they are willing and able to learn how to do new things. Learning this led us to ask a critical question: how would we intentionally design a learning-centered experience?

To craft the experience we were aiming for, we knew we had to create a learning-centered culture for our students and our employees. We knew that our staff would need to model the behaviors our students needed to learn. We knew the best way to shape the culture was to work with our direct reports—our directs—to develop the behaviors we wanted them to exemplify.

To craft the experience we were aiming for, we knew we had to create a learning-centered culture for our students *and* our employees. We knew that our staff would need to model the behaviors our students needed to learn.

BUILDING A LEARNING TEAM

Our learning-centered culture starts with our staff. We believe in transparency. Transparency builds trust. Effective organizations have effective teams who trust each other as individuals.

One huge way we build that trust and provide opportunities for transparency is in our meetings. (I know, I know—meetings! Yuck!) But seriously, running and participating in effective meetings is a great opportunity to build a learning-centered culture.

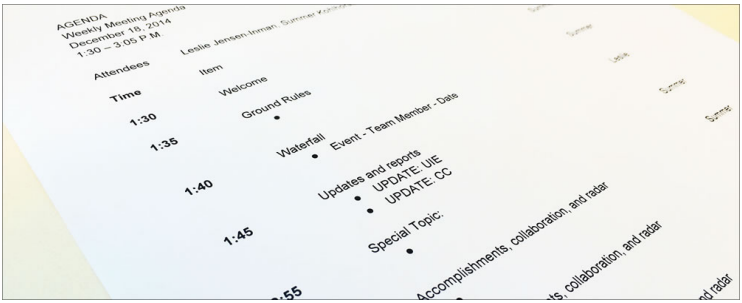
Meetings—when done well—allow individuals time to come together, to share, and to listen. These behaviors, executed on a consistent and regular basis, build honest and trusting relationships.

An effective meeting is one that achieves the desired outcomes of that meeting. While different meetings aim for different results, at Center Centre all meetings have a secondary goal: meet for learning.

A framework for learning-centered meetings

We've developed a framework for our meetings. We use it for all our meetings, which means attendees know what to expect. It also saves us from reinventing the wheel in each meeting.

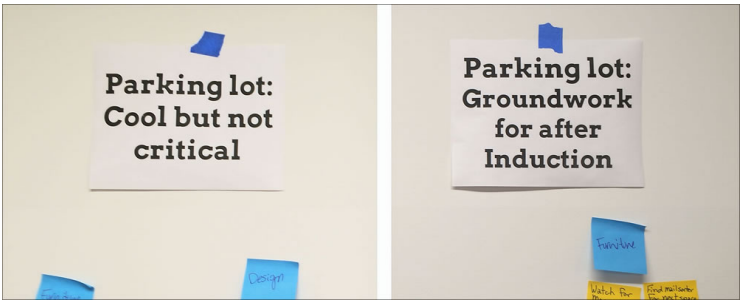
These basic steps help our meetings focus on the valuable face-to-face interaction we're having, and help us truly begin to learn from one another.



An agenda for a staff meeting.

USE EFFECTIVE MEETING BASICS

- Prepare for the meeting *before* the meeting.
- If you're running the meeting, prepare a typed agenda and share it *before* the meeting. Agendas have start times for each item.
- Start the meeting on time. Don't wait for stragglers.
- Define ground rules. Get input from attendees. Recurring meetings don't have to do this every time.
- Keep to the meeting agenda. Put off-topic questions and ideas in a parking lot, a visual document that everyone can see, so you can address the questions and ideas later.
- Finish on time. And if you've reached the meeting's goals, finish early.



Parking lots where ideas on sticky notes can be posted for later consideration.

FOCUS TO LEARN

- Have tech-free meetings: no laptops, no phones, no things with notifications.
- Bring a notebook and a pen.
- Take notes by hand. You're not taking minutes, you're writing to learn.

COME WITH A LEARNING MINDSET

- Ask: what are our goals for this meeting? (Hopefully answered by the meeting agenda.)
- Ask: what can I learn overall?
- Ask: what can I learn from each of my colleagues?
- Ask: what can I share that will help the team learn overall?
- Ask: what can I share that will help each of my colleagues learn?

INVESTING IN REGULARLY SCHEDULED LEARNING-CENTERED MEETINGS

At Center Centre, we have two types of recurring all-staff meetings: daily stand-ups and weekly staff meetings. (We are a small organization, so it makes sense to meet as an entire group.)

Yes, that means we spend thirty minutes each day in stand-up, for a total of two and a half hours of stand-up meeting time each week. And, yes, we also have a weekly ninety-minute sit-down staff meeting on top of that. This investment in time is an investment in learning.

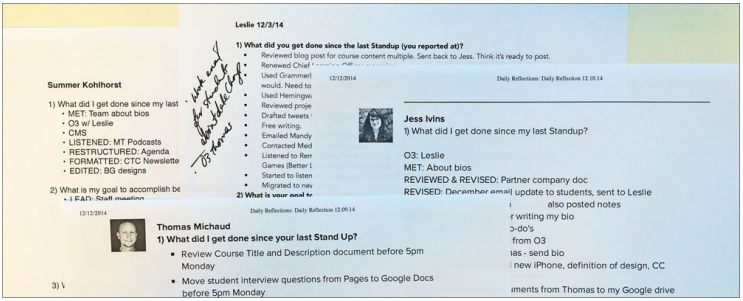
We use these meetings to build our transparency, and, therefore, our trust. The regularity of these meetings helps us maintain ongoing, open sharing about our responsibilities, our successes, and our learning.

For instance, we answer five questions in our stand-up:

1. What did I get done since the last stand-up (I reported at)?
2. What is my goal to accomplish before the next stand-up?
3. What's preventing me from getting these things done, if anything?
4. What's the highest risk or most unknown thing right now about what I'm trying to get done?

5. What is the most important thing I learned since the last time we met *and* how will what I learned change the way I approach things in the future?

Each person writes out their answers to these questions *before* the meeting. Each person brings their answers printed on paper to the meeting. And each person brings a pen to jot down notes.



Notes compiled for a stand-up meeting.

During the stand-up, each person shares their answers to the five questions. To sustain a learning-centered culture, the fifth question is the most important question to answer. It allows individual reflection focused on learning. Sometimes this isn't an easy question to answer. It makes us stretch. It makes us think.

By sharing our individual answers to the fifth question, we open ourselves up to the group. When we honestly share what we've learned, we openly admit that we didn't know something. Sharing like this would be scary (and even risky) if we didn't have a learning-centered culture.

We often share the actual process of how we learned something. By listening, each of us is invited to learn more about the topic at hand, consider what more there is to learn about that topic, and even gain insights into other methods of learning—which can be applied to other topics.

Sharing the answers to the fifth question also allows opportunities for further conversations. We often take what someone has individually learned and find ways to apply it for our entire team in support of our organization. We are, after all, learning together.

BUILDING INDIVIDUAL LEARNERS

We strive to grow together as a team at Center Centre, but we don't lose sight of the importance of the individuals who form our team. As individuals, we bring our goals, dreams, abilities, and prior knowledge to the team.

To build learning teams, we must build individual learners. A team made up of lifelong learners, who share their learning and learn from each other, is a team that will continually produce better results.

As a manager, I need to meet each direct where they are with their current abilities and knowledge. Then, I can help them take their skills and knowledge base to the next levels. This process requires each individual direct to engage in professional development.

We believe effective managers help their directs engage in behaviors that support growth and development. Effective managers encourage and support learning.

Our weekly one-on-ones

One way we encourage learning is through weekly one-on-ones. Each of my directs meets with me, individually, for thirty minutes each week. The meeting is their meeting. It is not my meeting.

My direct sets the agenda. They talk about what they want to talk about. They can talk about work. They can talk about things outside of work. They can talk about their health, their kids, and even their cat. Whatever is important to them is important to me. I listen. I take notes.

Although the direct sets the specific agenda, the meeting has three main parts. Approximately ten minutes for them (the direct), ten minutes for me (the manager), and ten minutes for us to talk about their future within—and beyond—our organization.

COACHING FOR FUTURE PERFORMANCE

The final third of our one-on-one is when I coach my directs. Coaching looks to the direct's future performance. It focuses on developing the direct's skills.

Coaching isn't hard. It doesn't take much time. For me, it usually takes less than five minutes a week during a one-on-one.

The first time I coach one of my directs, I ask them to brainstorm about the skills they want to improve. They usually already have an idea about this. It's often something they've wanted to work on for some time, but didn't think they had the time or the knowhow to improve.

If a direct doesn't know what they want to improve, we discuss their job responsibilities—specifically the aspects of the job that concern them.

Coaching provides an opportunity for me to ask, "In your job, what are the required skills that you feel like you don't have (or know well enough, or perform effectively, or use with ease)?"

Sometimes I have to remind a direct that it's okay not to know how to do something (even if it's a required part of their job). After all, our organization is a learning organization. In a learning organization, no one knows everything but everyone is willing to learn anything.

After we review the job responsibilities together, I ask my direct what skill they'd like to work to improve. Whatever they choose, we focus on that skill for coaching—I've found my directs work better when they're internally motivated.

Sometimes the first time I talk with a direct about coaching, they get a bit anxious. If this happens, I share a personal story about my professional learning journey. I say something like:

I didn't know how to make a school before we started to make Center Centre.

I didn't know how to manage an entire team of people—day in and day out—until I started managing a team of people every day.

When I realized that I was the boss—and that the success of the school would hinge, at least in part, on my skills as a manager—I was a bit terrified. I was missing an important skill set that I needed to know (and I needed to know well).

When I first understood this, I felt bad—like I should have already known how to be a great manager. But then I realized, I'd never faced this situation. I'd never needed to know how to use this skill set in this way.

I worked through my anxiety about feeling inadequate. I decided I'd better learn how to be an effective manager because the school needed me to be one. You needed me to be one.

Every day, I work to improve my management skills. You've probably noticed that some days I'm better at it than others. I try not to beat myself up about this, although it's hard—I'd like to be perfect at it. But I'm not.

> I know that if I make a conscious, daily effort to learn how to be a better manager, I'll continue to improve. So that's what I do.
>
> Every day I learn. I learn by doing. I learn how to be better than I was the day before. That's what I ask of you.

Once we determine the skill the direct wants to learn, we figure out how they can go about learning it. I ask: "How could you learn this skill?"

We brainstorm for two or three minutes about this. We write down every idea that comes to mind, and we write it so both of us can easily see the options (both whiteboards and sticky notes on the wall work well for this exercise).

> Read a book. Research online. Watch a virtual seminar. Listen to a podcast. Talk to a mentor. Reach out to an expert. Attend a conference. Shadow someone else while they do the skill. Join a professional organization.

The goal is to get the direct on a path of self-development. I'm coaching their development, but I'm *not* the main way my direct will learn this new skill.

I ask my direct which path seems like the best place to start. I let them choose whatever option they want (as long as it works with our budget). They are more likely to follow through if they are in control of this process.

Next, we work to break down the selected path into tasks. We only plan one week's worth of tasks. The tasks are small, and the deadlines are short. My direct reports when each task is completed.

At our next one-on-one, I ask my direct about their experience learning this new skill.

Rinse. Repeat.

That's it. I spend five minutes a week talking with each direct about their individual learning. They develop their professional skills, and together we're creating a learning-centered culture.

ASKING QUESTIONS I DON'T KNOW THE ANSWER TO

When my direct said, "I've never worked where people are so honest and transparent about things," it led me to believe that all this is working. We *are* building a learning-centered culture.

This week I was reminded that creating a learning-centered culture starts not just with the staff, but with me. When I challenge myself to learn and then share what I'm currently learning, my directs want to learn more about what I'm learning about.

For example, I decided I needed to improve my writing skills. A few weeks ago, I realized that I was sorely out of practice and I felt like I had lost my voice. So I started to write. I put words on paper. I felt overwhelmed. I felt like I didn't know how to write anymore (at least not well or effectively).

I bought some books on writing (mostly Peter Elbow's books like *Writing with Power*, *Writing Without Teachers*, and *Vernacular Eloquence*), and I read them. I read them all. Reading these books was part of my personal coaching. I used the same steps to coach myself as I use with my directs when I coach them.

In stand-ups, I started sharing what I accomplished (like I completed one of the books) and what I learned by doing—specific things, like engaging in freewriting and an open-ended writing process.

This week, I went to lunch with one of my directs. She said, "You've been talking about freewriting a lot. You're really excited about it. Freewriting seems like it's helping your writing process. Would you tell me more about it?"

So I shared the details with her. I shared the reasons why I think freewriting is helping. I'm not focused on perfection. Instead, each day I'm focused on spending ten, uninterrupted minutes writing down whatever comes to

my mind. It's opening my writing mind. It's allowing my words to flow more freely. And it's helping me feel less self-conscious about my writing.

She said, "Leslie, when you say you're self-conscious about your writing, I laugh. Not because it's funny. But because when I read what you write, I think, 'What is there to improve?' I think you're a great writer. It's interesting to know that you think you can be a better writer. I like learning about your learning process. I think I could do freewriting. I'm going to give it a try."

There's something magical about all of this. I'm not even sure I can eloquently put it into words. I just know that our working environment is something very different. I've never experienced anything quite like it. Somehow, by sharing that I don't know everything and that I'm always working to learn more, I invite my directs to be really open about what they don't know. And they see it's possible always to learn and grow.

I'm glad I ignore all the lawyering I've learned from watching TV. I'm glad I ask the questions I don't know the answers to. And I'm glad my directs do the same. When we meet for learning, we accelerate and amplify the learning process—building individual learners and learning teams. Embracing the unknown and working toward understanding is what makes our culture a learning-centered culture.

Photos by Summer Kohlhorst.

ABOUT THE AUTHOR

Dr. Leslie Jensen-Inman is co-founder of Centre Centre, a school creating industry-ready user experience designers. Leslie combines her 19 years of design practice and eight years of instructional background to make Center Centre an extraordinary learning environment.

Leslie creative directed and co-authored the book, *InterACT with Web Standards: A holistic approach to web design*. She writes articles for publications such as *A List Apart*, *The Pastry Box*, *Ladies in Tech*, and *.net Magazine*. She speaks at and keynotes conferences including Build, Converge, and SXSW. You can reach Leslie at jenseninman.com and on Twitter @jenseninman.

21. Naming Things

Paul Lloyd 24ways.org/201421

> There are only two hard things in computer science: cache invalidation and naming things.
> *Phil Karlton*

Being a professional web developer means taking responsibility for the code you write and ensuring it is comprehensible to others. Having a documented code style is one means of achieving this, although the size and type of project you're working on will dictate the conventions used and how rigorously they are enforced.

Working in-house may mean working with multiple developers, perhaps in distributed teams, who are all committing changes – possibly to a significant codebase – at the same time. Left unchecked, this codebase can become unwieldy. Coding conventions ensure everyone can contribute, and help build a product that works as a coherent whole.

Even on smaller projects, perhaps working within an agency or by yourself, at some point the resulting product will need to be handed over to a third party. It's sensible, therefore, to ensure that your code can be understood by those who'll eventually take ownership of it.

Put simply, code is read more often than it is written or changed. A consistent and predictable naming scheme can make code easier for other developers to understand, improve and maintain, presumably leaving them free to worry about cache invalidation.

LET'S TALK ABOUT SEMANTICS

Names not only allow us to identify objects, but they can also help us describe the objects being identified.

Semantics (the meaning or interpretation of words) is the cornerstone of standards-based web development. Using appropriate HTML elements allows us to create documents and applications that have implicit structural meaning. Thanks to HTML5, the vocabulary we can choose from has grown even larger.

HTML elements provide one level of meaning: a widely accepted description of a document's underlying structure. It's only with the mutual agreement of browser vendors and developers that <p> indicates a paragraph.

Yet (with the exception of widely accepted microdata and microformat schemas) only HTML elements convey any meaning that can be parsed consistently by user agents. While using semantic values for class names is a noble endeavour, they provide no additional information to the visitor of a website; take them away and a document will have exactly the same semantic value.

I didn't always think this was the case, but the real world has a habit of changing your opinion. Much of my thinking around semantics has been informed by the writing of my peers. In "About HTML semantics and front-end architecture", Nicholas Gallagher wrote:

> The important thing for class name semantics in non-trivial applications is that they be driven by pragmatism and best serve their primary purpose – providing meaningful, flexible, and reusable presentational/behavioural hooks for *developers* to use.

These thoughts are echoed by Harry Roberts in his CSS Guidelines:

> The debate surrounding semantics has raged for years, but it is important that we adopt a more pragmatic, sensible approach to naming things in order to work more efficiently and effectively. Instead of focussing on 'semantics', look more closely at sensibility and longevity – choose names based on ease of maintenance, not for their perceived meaning.

NAMING METHODOLOGIES

Front-end development has undergone a revolution in recent years. As the projects we've worked on have grown larger and more important, our development practices have matured. The pros and cons of object-orientated approaches to CSS can be endlessly debated, yet their introduction has highlighted the usefulness of having documented naming schemes.

Jonathan Snook's SMACSS (Scalable and Modular Architecture for CSS) collects style rules into five categories: base, layout, module, state and theme. This grouping makes it clear what each rule does, and is aided by a naming convention:

> By separating rules into the five categories, naming convention is beneficial for immediately understanding which category a particular style belongs to and its role within the overall scope of the page. On large projects, it is more likely to have styles broken up across multiple files. In these cases, naming convention also makes it easier to find which file a style belongs to.
>
> I like to use a prefix to differentiate between layout, state and module rules. For layout, I use `l-` but `layout-` would work just as well. Using prefixes like `grid-` also provide enough clarity to separate layout styles from other styles. For state rules, I like `is-` as in `is-hidden` or `is-collapsed`. This helps describe things in a very readable way.

SMACSS is more a set of suggestions than a rigid framework, so its ideas can be incorporated into your own practice. Nicholas Gallagher's SUIT CSS project is far more strict in its naming conventions:

> SUIT CSS relies on *structured class names* and *meaningful hyphens* (i.e., not using hyphens merely to separate words). This helps to work around the current limits of applying CSS to the DOM (i.e., the lack of style encapsulation), and to better communicate the relationships between classes.

Over the last year, I've favoured a BEM-inspired approach to CSS. BEM stands for *block*, *element*, *modifier*, which describes the three types of rule that contribute to the style of a single component. This means that, given the following markup:

```
<ul class="sleigh">
    <li class="sleigh__reindeer
sleigh__reindeer--famous">Rudolph</li>
    <li class="sleigh__reindeer">Dasher</li>
    <li class="sleigh__reindeer">Dancer</li>
    <li class="sleigh__reindeer">Prancer</li>
    <li class="sleigh__reindeer">Vixen</li>
    <li class="sleigh__reindeer">Comet</li>
    <li class="sleigh__reindeer">Cupid</li>
    <li class="sleigh__reindeer">Dunder</li>
    <li class="sleigh__reindeer">Blixem</li>
</ul>
```

I know that:

- `.sleigh` is a containing *block* or component.
- `.sleigh__reindeer` is used only as a descendent *element* of `.sleigh`.

- `.sleigh__reindeer--famous` is used only as a *modifier* of `.sleigh__reindeer`.

With this naming scheme in place, I know which styles relate to a particular component, and which are shared. Beyond reducing specificity-related head-scratching, this approach has given me a framework within which I can consistently label items, and has sped up my workflow considerably.

Each of these methodologies shows that any robust CSS naming convention will have clear rules around case (lowercase, camelCase, PascalCase) and the use of special (allowed) characters like hyphens and underscores.

WHAT MAKES FOR A GOOD NAME?

Regardless of higher-level conventions, there's no getting away from the fact that, at some point, we're still going to have to name things. Recognising that classes should be named with other developers in mind, what makes for a good name?

Understandable

The most important aspect is for a name to be understandable. Words used in your project may come from a variety of sources: some may be widely understood, and others only be recognised by people working within a particular environment.

- **Culture**

Most words you'll choose will have common currency outside the world of web development, although they may have a particular interpretation among developers (think *menu*, *list*, *input*). However, words may have a narrower cultural significance; for example, in Germany and other German-speaking countries, *impressum* is the term used for legally mandated statements of ownership.

- **Industry**

Industries often use specific terms to describe common business practices and concepts. Publishing has a number of these (*headline*, *standfirst*, *masthead*, *colophon*…) all have well understood meanings – and not all of them are relevant to online usage.

- **Organisation**

Companies may have internal names (or nicknames) for their products and services. The Guardian is rife with such names: *bisons* (and *buffalos*), *pixies* (and *super-pixies*), *bentos* (and *mini-bentos*)… all of which mean something very different outside the organisation. Although such names can be useful inside smaller teams, in larger organisations

they can become a barrier to entry, a sort of secret code used among employees who have been around long enough to know what they mean.

- **Product**

Your team will undoubtedly have created names for specific features or interface components used in your product. For example, at Clearleft we coined the term *gravigation* for a navigation bar that was pinned to the bottom of the viewport. Elements of a visual design language may have names, too. Transport for London's bar and circle logo is known internally as the ***roundel***, while Nike's logo is called the ***swoosh***. Branding agencies often christen colours within a brand palette, too, either to evoke aspects of the identity or to indicate intended usage.

Once you recognise the origin of the words you use, you'll be better able to judge their appropriateness. Using Latin words for class names may satisfy a need to use semantic-sounding terms but, unless you work in a company whose employees have a basic grasp of Latin, a degree of translation will be required. Military ranks might be a clever way of declaring sizes without implying actual values, but I'd venture most people outside the armed forces don't know how they're ordered.

Obvious

Quite often, the first name that comes into your head will be the best option. Names that obliquely reference the function of a class (e.g. *receptacle* instead of *container*, *kevlar* instead of *no-bullets*) only serve to add an additional layer of abstraction. Don't overthink it!

One way of knowing if the names you use are well understood is to look at what similar concepts are called in existing vocabularies. schema.org, Dublin Core and the BBC's ontologies are all useful sources for object names.

Functional

While we've learned to avoid using presentational classes, there remains a tension between naming things based on their content, and naming them for their intended presentation or behaviour (which may change at different breakpoints). Rather than think about a component's appearance or behaviour, instead look to its function, its purpose. To clarify, ask what a component's function is, and not how the component functions.

For example, the Guardian's internal content system uses the following names for different types of image placement: *supporting*, *showcase* and *thumbnail*, with *inline* being the default. These options make no promise of the

resulting position on a webpage (or smartphone app, or television screen…), but do suggest intended use, and therefore imply the likely presentation.

Consistent

Being consistent in your approach to names will allow for easier naming of successive components, and extending the vocabulary when necessary. For example, a predictably named hierarchy might use names like *primary* and *secondary*. Should another level need to be added, *tertiary* is clearly be preferred over *third*.

Appropriate

Your project will feature a mix of style rules. Some will perform utility functions (clearing floats, removing bullets from a list, reseting margins), while others will perform specific functions used only once or twice in a project. Names should reflect this. For commonly used classes, be generic; for unique components be more specific.

It's also worth remembering that you can use multiple classes on an element, so combining both generic and specific can give you a powerful modular design system:

- Generic: `list`
- Specific: `naughty-children`
- Combined: `naughty-children list`

If following the BEM methodology, you might use the following classes:

- Generic: `list`
- Specific: `list--nice-children`
- Combined: `list list--nice-children`

Extensible

Good naming schemes can be extended. One way of achieving this is to use namespaces, which are basically a way of grouping related names under a higher-level term.

Microformats are a good example of a well-designed naming scheme, with many of its vocabularies taking property names from existing and related specifications (e.g. hCard is a 1:1 representation of vCard). Microformats 2 goes one step further by grouping properties under several namespaces:

- `h-*` for root class names (e.g. `h-card`)
- `p-*` for simple (text) properties (e.g. `p-name`)
- `u-*` for URL properties (e.g. `u-photo`)
- `dt-*` for date/time properties (e.g. `dt-bday`)
- `e-*` for embedded markup properties (e.g. `e-note`)

The inclusion of namespaces is a massive improvement over the earlier specification, but the downside is that microformats now occupy five separate namespaces. This

might be problematic if you are using `u-*` for your utility classes. While nothing will break, your naming system won't be as robust, so plan accordingly.

(Note: Microformats perform a very specific function, separate from any presentational concerns. It's therefore considered best practice to not use microformat classes as styling hooks, but instead use additional classes that relate to the function of the component and adhere to your own naming conventions.)

Short

Names should be as long as required, but no longer. When looking for words to describe a particular function, I try to look for single words where possible. Avoid abbreviations unless they are understood within the contexts described above. *rrp* is fine if labelling a recommended retail price in an online shop, but not very helpful if used to mean ragged-right paragraph, for example.

Fun!

Finally, names can be an opportunity to have some fun! Names can give character to a project, be it by providing an outlet for in-jokes or adding little easter eggs for those inclined to look.

The copyright statement on Apple's website has long been named *sosumi*, a word that has a nice little history inside Apple. Until recently, the hamburger menu icon on the Guardian website was labelled *honest-burger*, after the developer's favourite burger restaurant.

A FEW THOUGHTS ON PREPROCESSORS

CSS preprocessors have solved a lot of problems, but they have an unfortunate downside: they require you to name yet more things! Whereas we needed to worry only about style rules, now we need names for variables, mixins, functions… oh my!

A second article could be written about naming these, so for now I'll offer just a few thoughts. The first is to note that preprocessors make it easier to change things, as they allow for DRYer code. So while the names of variables are important (and the advice in this article still very much applies), you can afford to relax a little.

Looking to name colour variables? If possible, find out if colours have been assigned names in a brand palette. If not, use obvious names (based on appearance or function, depending on your preference) and adapt as the palette grows. If it becomes difficult to name colours that are too similar, I'd venture that the problem lies with the design rather than the naming scheme.

The same is true for responsive breakpoints. Preprocessors allow you to move awkward naming conventions out of the markup and into the CSS. Although terms like *mobile*, *tablet* and *desktop* are not desirable given the need to think about device-agnostic design, if these terms are widely understood within a product team and among stakeholders, using them will ensure everyone is using the same language (they can always be changed later).

It still feels like we're at the very beginning of understanding how preprocessors fit into a development workflow, if at all! I suspect over the next few years, best practices will emerge for all of these considerations. In the meantime, use your brain!

Even with sensible rules and conventions in place, naming things can remain difficult, but hopefully I've made this exercise a little less painful. Christmas is a time of giving, so to the developer reading your code in a year's time, why not make your gift one of clearer class names.

ABOUT THE AUTHOR

Paul Robert Lloyd is interaction designer at the Guardian. Prior to this he was a senior designer at Clearleft, where he worked for clients such as NBCUniversal, Channel 4, Mozilla and UNICEF UK.

When not working on side projects (he is currently digitizing George Bradshaw's railway guide), Paul can be found writing about design, travel and more on his blog or blathering on Twitter.

22. Integrating Contrast Checks in Your Web Workflow

Geri Coady

24ways.org/201422

It's nearly Christmas, which means you'll be sure to find an overload of festive red and green decorating everything in sight—often in the ugliest ways possible.

While I'm not here to battle holiday tackiness in today's 24 ways, it might just be the perfect reminder to step back and consider how we can implement colour schemes in our websites and apps that are not only attractive, but also legible and accessible for folks with various types of visual disabilities.

This simulated photo demonstrates how red and green Christmas baubles could appear to a person affected by protanopia-type colour blindness—not as festive as you might think. Source: Derek Bruff

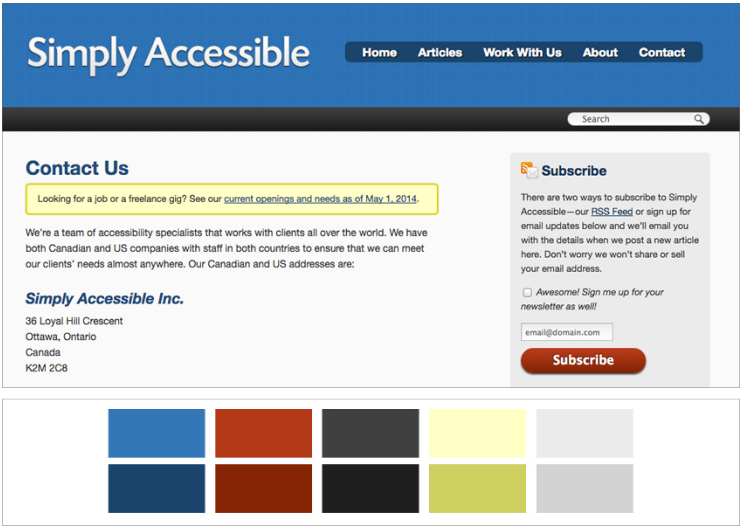
I've been fortunate to work with Simply Accessible to redesign not just their website, but their entire brand. Although the new site won't be launching until the new year, we're excited to let you peek under the tree and share a few treats as a case study into how we tackled colour accessibility in our project workflow. Don't worry—we won't tell Santa!

CREATE A COLOUR GAME PLAN

A common misconception about accessibility is that meeting compliance requirements hinders creativity and beautiful design—but we beg to differ. Unfortunately, like many company websites and internal projects, Simply Accessible has spent so much time helping others that they had not spent enough time helping themselves to show the world who they really are. This was the perfect opportunity for them to practise what they preached.

After plenty of research and brainstorming, we decided to evolve the existing Simply Accessible brand. Or, rather, salvage what we could. There was no established logo to carry into the new design (it was a stretch to even call it a wordmark), and the Helvetica typography across the site

lacked any character. The only recognizable feature left to work with was colour. It was a challenge, for sure: the oranges looked murky and brown, and the blues looked way too corporate for a company like Simply Accessible. We knew we needed to inject a *lot* of personality.



The old Simply Accessible website and colour palette.

After an audit to round up every colour used throughout the site, we dug in deep and played around with some ideas to bring some new life to this palette.

CHOOSE EFFECTIVE COLOURS

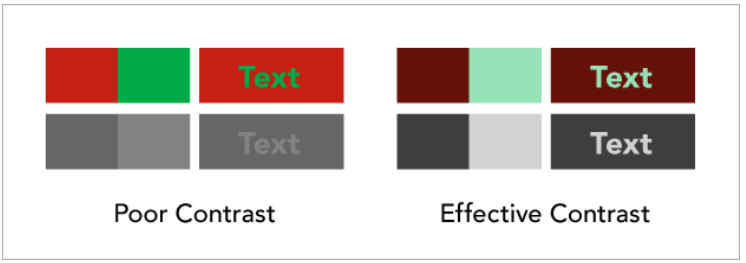
Whether you're starting from scratch or evolving an existing brand, the first step to having an effective and legible palette begins with your colour choices. While we aren't going to cover colour message and meaning in this article, it's important to understand how to choose colours that can be used to create strong contrast—one of the most important ways to create hierarchy, focus, and legibility in your design.

There are a few methods of creating effective contrast.

Light and dark colours

The contrast that exists between light and dark colours is the most important attribute when creating effective contrast.

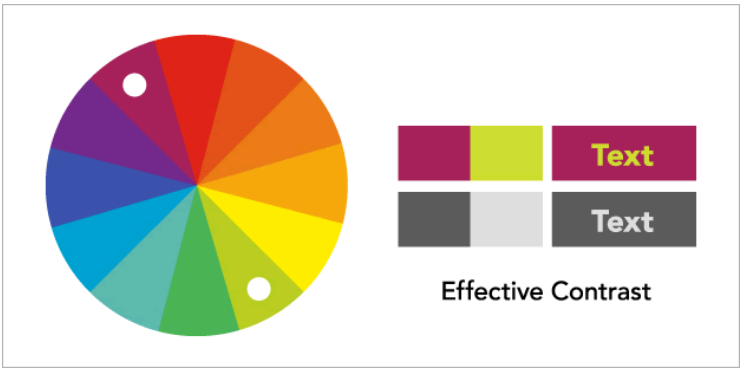
Try not to use colours that have a similar lightness next to each other in a design.



The red and green colours on the left share a similar lightness and don't provide enough contrast on their own without making some adjustments. Removing colour and showing the relationship in greyscale reveals that the version on the right is much more effective.

It's important to remember that red and green colour pairs cause difficulty for the majority of colour-blind people, so they should be avoided wherever possible, especially when placed next to each other.

Complementary contrast



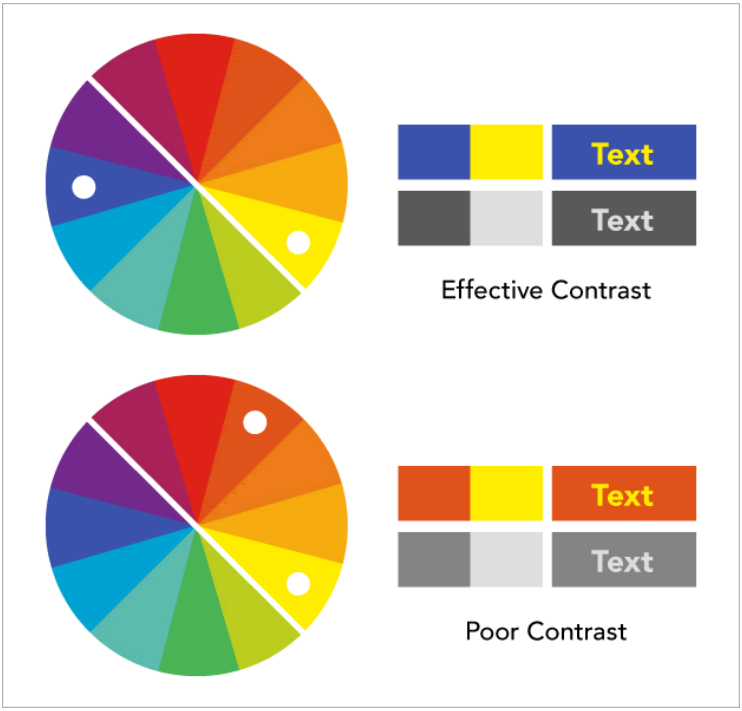
Effective contrast can also be achieved by choosing complementary colours (other than red and green), that are opposite each other on a colour wheel.

These colour pairs generally work better than choosing adjacent hues on the wheel.

Cool and warm contrast

Contrast also exists between cool and warm colours on the colour wheel.

Imagine a colour wheel divided into cool colours like blues, purples, and greens, and compare them to warm colours like reds, oranges and yellows.



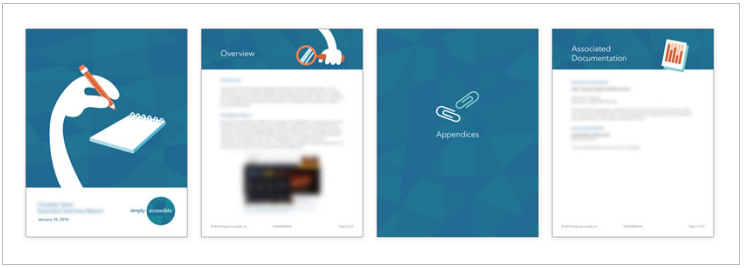
Choosing a dark shade of a cool colour, paired with a light tint of a warm colour will provide better contrast than two warm colours or two cool colours.

DEVELOP COLOUR CONCEPTS

After much experimentation, we settled on a simple, two-colour palette of blue and orange, a cool-warm contrast colour scheme. We added swatches for call-to-action messaging in green, error messaging in red, and body copy and form fields in black and grey. Shades and tints of blue and orange were added to illustrations and other design elements for extra detail and interest.

First stab at a new palette.

We introduced the new palette for the first time on an internal project to test the waters before going full steam ahead with the website. It gave us plenty of time to get a feel for the new design before sharing it with the public.



Putting the test palette into practice with an internal report

It's important to be open to changes in your palette as it might need to evolve throughout the design process. Don't tell your client up front that this palette is set in stone. If you need to tweak the colour of a button later because of legibility issues, the last thing you want is your client pushing back because it's different from what you promised.

As it happened, we did tweak the colours after the test run, and we even adjusted the logo—what looked great printed on paper looked a little too light on screens.

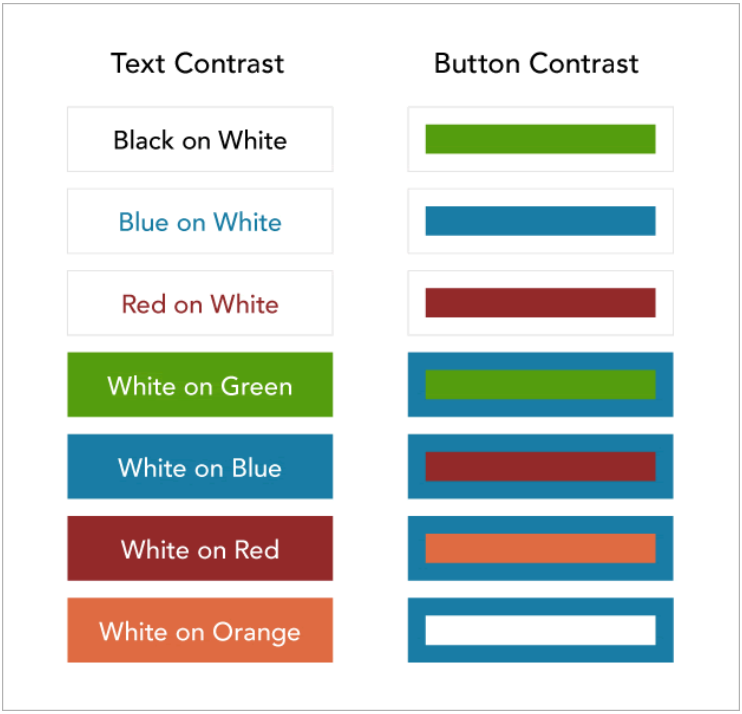
CONSIDER HOW COLOURS MIGHT BE USED

Don't worry if you haven't had the opportunity to test your palette in advance. As long as you have some well-considered options, you'll be ready to think about how the colour might be used on the site or app.

Obviously, in such early stages it's unlikely that you're going to know every element or feature that will appear on the site at launch time, or even which design elements could be introduced to the site later down the road. There are, of course, plenty of safe places to start.

For Simply Accessible, I quickly mocked up these examples in Illustrator to get a handle on the elements of a website where contrast and legibility matter the most: text colours and background colours. While it's less important to consider the contrast of decorative elements

that don't convey essential information, it's important for a reader to be able to discern elements like button shapes and empty form fields.



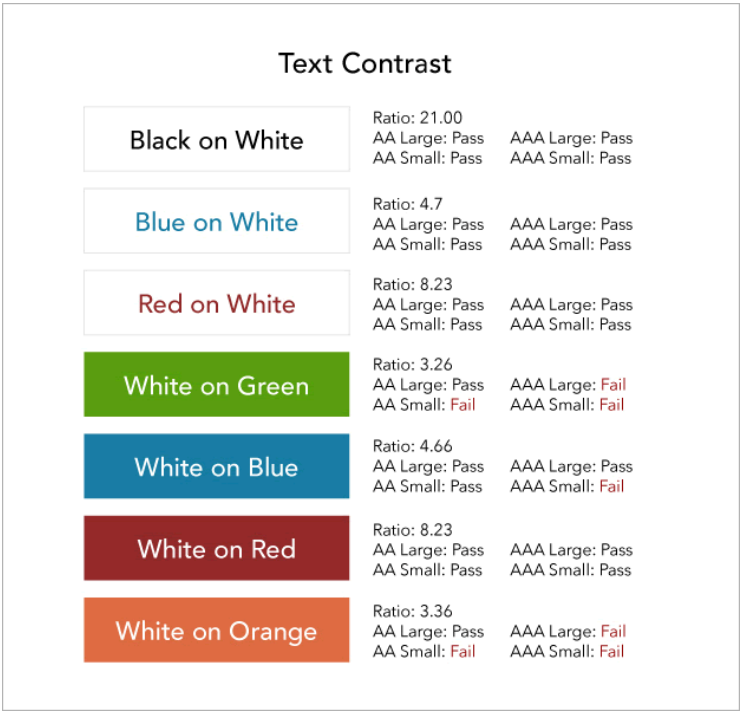
A basic list of possible colour combinations that I had in mind for the Simply Accessible website

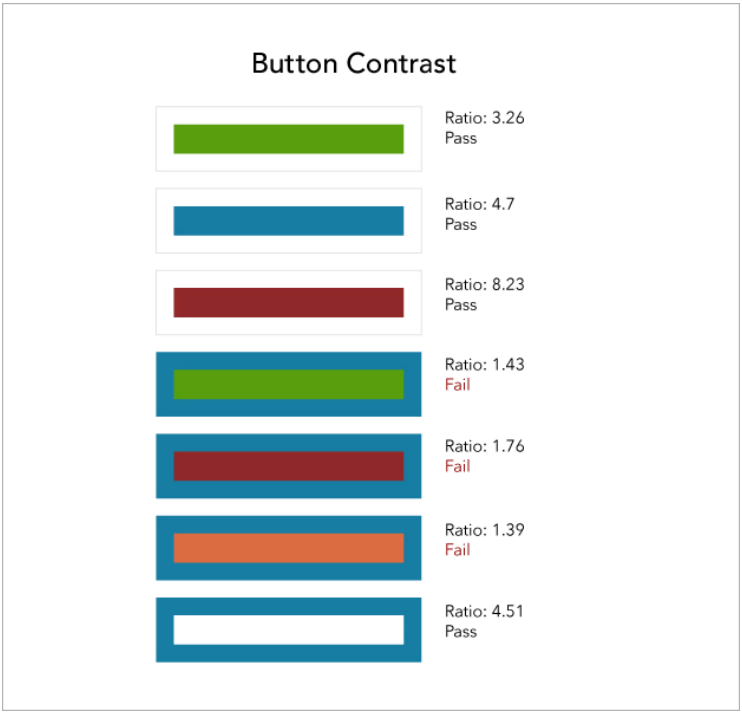
RUN INITIAL TESTS

Once these elements were laid out, I manually plugged in the HTML colour code of each foreground colour and background colour on Lea Verou's Contrast Checker. I

added the results from each colour pair test to my document so we could see at a glance which colours needed adjustment or which colours wouldn't work at all.

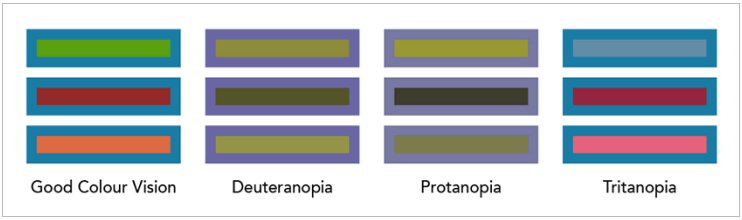
Note: Read more about colour accessibility and contrast requirements





As you can see, a few problems were revealed in this test. To meet the minimum AA compliance, we needed to slightly darken the green, blue, and orange background colours for text—an easy fix. A more complicated problem was apparent with the button colours. I had envisioned some buttons appearing over a blue background, but the contrast ratios were well under 3:1. Although there isn't a guide in WCAG for contrast requirements of two non-text elements, the ISO and ANSI standard for visible contrast is 3:1, which is what we decided to aim for.

We also checked our colour combinations in Color Oracle, an app that simulates the most extreme forms of colour blindness. It confirmed that coloured buttons over blue backgrounds was simply not going to work. The contrast was much too low, especially for the more common deuteranopia and protanopia-type deficiencies.



How our proposed colour pairs could look to people with three types of colour blindness

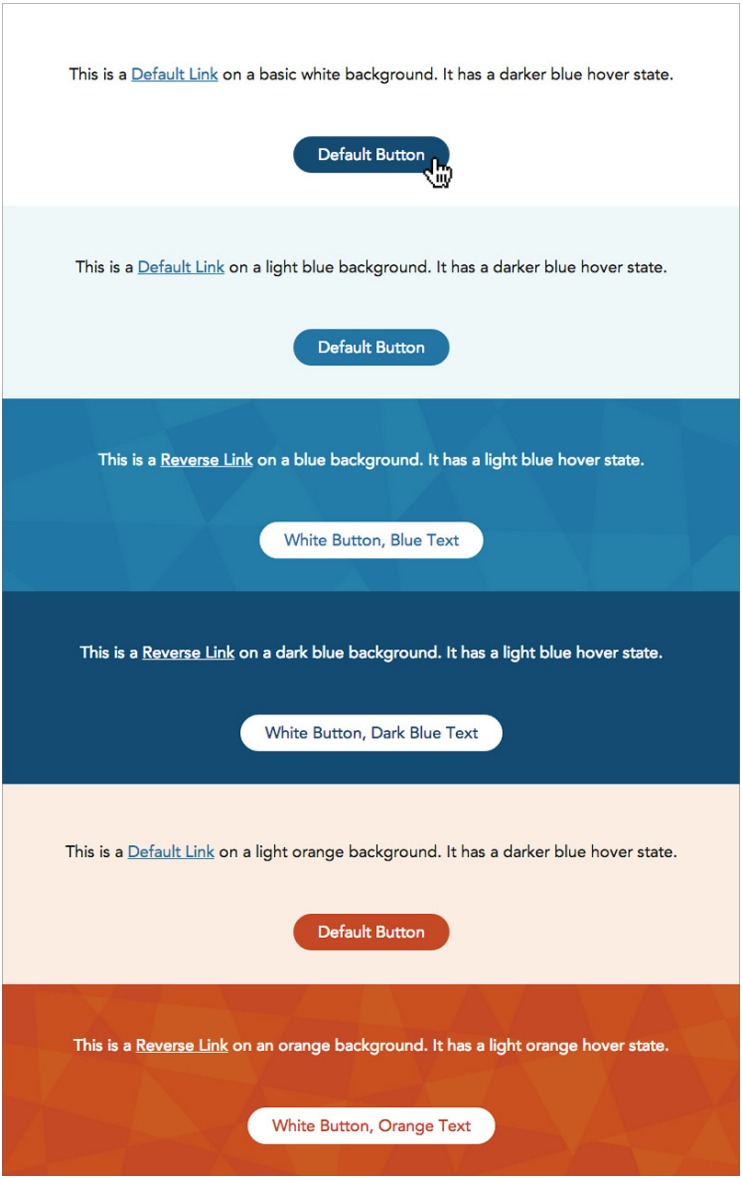
MAKE ADJUSTMENTS IF NECESSARY

As a solution, we opted to change all buttons to white when used over dark coloured backgrounds. In addition to increasing contrast, it also gave more consistency to the button design across the site instead of introducing a lot of unnecessary colour variants.

Putting more work into getting compliant contrast ratios at this stage will make the rest of implementation and testing a breeze. When you've got those ratios looking good, it's time to move on to implementation.

IMPLEMENT COLOURS IN STYLE GUIDE AND PROTOTYPE

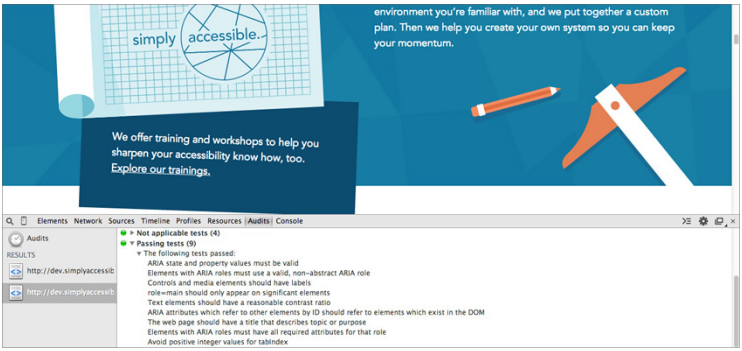
Once I was happy with my contrast checks, I created a basic style guide and added all the colour values from my colour exploration files, introduced more tints and shades, and added patterned backgrounds. I created examples of every panel style we were planning to use on the site, with sample text, links, and buttons—all with working hover states. Not only does this make it easier for the developer, it allows you to check in the browser for any further contrast issues.


This is a Default Link on a basic white background. It has a darker blue hover state.
Default Button
This is a Default Link on a light blue background. It has a darker blue hover state.
Default Button
This is a Reverse Link on a blue background. It has a light blue hover state.
White Button, Blue Text
This is a Reverse Link on a dark blue background. It has a light blue hover state.
White Button, Dark Blue Text
This is a Default Link on a light orange background. It has a darker blue hover state.
Default Button
This is a Reverse Link on an orange background. It has a light orange hover state.
White Button, Orange Text

RUN A FINAL CONTRAST CHECK

During the final stages of testing and before launch, it's a good idea to do one more check for colour accessibility to ensure nothing's been lost in translation from design to code. Unless you've introduced massive changes to the design in the prototype, it should be fairly easy to fix any issues that arise, particularly if you've stayed on top of updating any revisions in the style guide.

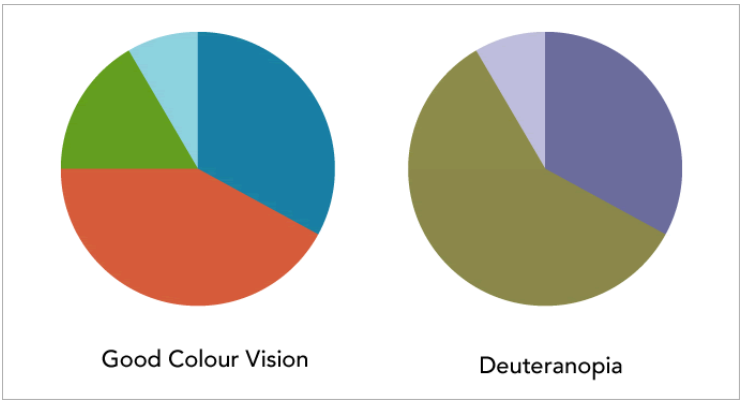
One of the more well-known evaluation tools, WAVE, is web-based and will work in any browser, but I love using Chrome's Accessibility Tools. Not only are they built right in to the Inspector, but they'll work if your site is password-protected or private, too.



Chrome's Accessibility Tools audit feature shows that there are no immediate issues with colour contrast in our prototype

THE HUMAN TOUCH

Finally, nothing beats a good round of user testing. Even evaluation tools have their flaws. Although they're great at catching contrast errors for text and backgrounds, they aren't going to be able to find errors in non-text elements, infographics, or objects placed next to each other where discernible contrast is important.



Our final palette, compared with our initial ideas, was quite different, but we're proud to say it's not just compliant, but shows Simply Accessible's true personality. Who knows, it may not be final at all—there are so many opportunities down the road to explore and expand it further.

Accessibility should never be an afterthought in a project. It's not as simple as adding `alt` text to images, or running your site through a compliance checker at the last minute and assuming that a pass means everything is okay. Considering how colour will be used during every stage of your project will help avoid massive problems before launch, or worse, launching with serious issues.

If you find yourself working on a personal project over the Christmas break, try integrating these checks into your workflow and make colour accessibility a part of your New Year's resolutions.

ABOUT THE AUTHOR

Geri Coady is a colour-obsessed illustrator and designer from Newfoundland, Canada. She is a former Art Director at a Canadian advertising agency and is now pursuing her own clients through her website at hellogeri.com. Geri loves chatting about nerdy things on Twitter and has shared her thoughts in publications such as net magazine, The Pastry Box Project, and Digital Arts. She's the author of the Pocket Guide to Colour Accessibility from Five Simple Steps, a sometimes-illustrator for A List Apart, and was voted Net Magazine's Designer of the Year in 2014.

23. Taglines and Truisms

Andrew Clarke 24ways.org/201423

To bring her good luck, "white rabbits" was the first thing that my grandmother said out loud on the first day of every month. We all need a little luck, but we shouldn't rely on it, especially when it comes to attracting new clients.

The first thing we say to a prospective client when they visit our website for the first time helps them to understand not only what we do but why we do it. We can also help them understand why they should choose to work with us over one of our competitors.

Take a minute or two to look at your competitors' websites. What's the first thing that they say about themselves? Do they say that they "design delightful digital experiences," "craft beautiful experiences" or "create remarkable digital experiences?"

It's easy to find companies who introduce themselves with what they do, their proposition, but what a company does is only part of their story. Their beliefs and values, what they stand for why they do what they do are also important.

When someone visits our websites for the first time, we have only a brief moment to help them understand us. To help us we can learn from the advertising industry, where the job of a tagline is to communicate a concept, deliver a message and sell a product, often using only a few words.

When an advertising campaign is effective, its tagline stays with you, sometimes long after that campaign is over. For example, can you remember which company or brand these taglines help to sell? (Answers at the bottom of the article:)

a. The Ultimate Driving Machine
b. Just Do It
c. Don't Leave Home Without It

A clever tagline isn't just a play on words, although it can include one. A tagline does far more than help make your company memorable. Used well, it brings together notions of what makes your company and what you offer special. Then it expresses those notions in a few words or possibly a short sentence.

I'm sure that everyone can find examples of company slogans written in the type of language that should stay within the walls of a marketing department. We can also find taglines where the meaning is buried so deep that the tag itself becomes effectively meaningless.

A meaningful tagline supports our ideas about who we are and what we offer, and provides a platform for different executions of them, sometimes over a period of time. For a tagline to work well, it must allow for current and future ideas about a brand.

It must also be meaningful to our brand and describe a truism, a truth that need not be a fact or statistic, but something that's true about us, who we are, what we do and why that's distinctive. It can be obvious, funny, serious or specific but above all it must be true. It should also be difficult to argue with, making your messages difficult to argue with too.

I doubt that I need remind you who this tagline belongs to:

> There are some things money can't buy. For everything else there's MasterCard.

That tagline was launched in 1997 by McCann-Erickson along with the "Priceless" campaign and it helped establish MasterCard as a friendlier credit card company, one with a sense of humour.

MasterCard's truism is that the things which really matter in life can't be bought. They are worth more than anything that a monetary value can be applied to. In expressing that truism through the tagline, MasterCard's advertising tells people to use not just any credit card, but their MasterCard, to pay for everything they buy.

"Guinness is good for you" may have been a stretch, but "Good things come to those who wait" builds on the truism that patience is a virtue and therefore a good pint of Guinness takes time to pour (119.5 seconds. I know you were wondering.)

The fact that British Airways flies to more destinations than any other airline is their truism, and led their advertisers to the now famous tagline, "The world's favourite airline."

At my company, Stuff & Nonsense, we've been thinking about taglines as we think about our position within an industry that seems full of companies who "design", "craft", and "create" "delightful", "beautiful", "remarkable digital experiences".

Much of what made us different has changed along with the type of work we're interested in doing. Our work's expanded beyond websites and now includes design for mobile and other media. It's true we can't know how or

where it will be seen. The ways that we make it are flexible too as we're careful not to become tied to particular tools or approaches.

It's also true that we're a small team. One that's flexible enough to travel around the world to work alongside our clients. We join their in-house teams and we collaborate with them in ways that other agencies often find more difficult. We know that our clients appreciate our flexibility and have derived enormous value from it. We know that we've won business because of it and that it's now a big part of our proposition.

Our truism is that we're flexible, "Fabulously flexible" as our tagline now expresses. And although we know that there may be other agencies who can be similarly flexible – after all, being flexible is not a unique selling proposition – only we do it so fabulously.

As the old year rolls into the new, how will your company describe what you do in 2015? More importantly, how will you tell prospective clients why you do it, what matters to you and why they should work with you?

Start by writing a list of truisms about your company. Write as many as you can, but then whittle that list down to just one, the most important truth. Work on that truism to create a tagline that's meaningful, difficult to be argue with and, above all, uniquely yours.

ANSWERS

a. The Ultimate Driving Machine (BMW)
b. Just Do It (Nike)
c. Don't Leave Home Without It (American Express)

ABOUT THE AUTHOR

Andrew Clarke runs Stuff and Nonsense, a tiny web design company where they make fashionably flexible websites. Andrew's the author of Transcending CSS and Hardboiled Web Design and hosts the popular weekly podcast Unfinished Business where he discusses the business side of web, design and creative industries with his guests. He tweets as @malarkey.

24. Cohesive UX

Cameron Moll 24ways.org/201424

With Yosemite, Apple users can answer iPhone calls on their MacBooks. This is weird. And yet it's representative of a greater trend toward cohesion.

Shortly after upgrading to Yosemite, a call came in on my iPhone and my MacBook "rang" in parallel. And I was all, like, "Wut?" This was a new feature in Yosemite, and honestly it was a little bizarre at first.

Apple promotional image showing a phone call ringing simultaneously on multiple devices.

However, I had just spoken at a conference on the very topic you're reading about now, and therefore I appreciated the underlying concept: the cohesion of user experience, the cohesion of screens.

This is just one of many examples I've encountered since beginning to speak about this topic months ago. But before we get ahead of ourselves, let's look back at the past few years, specifically the role of responsive web design.

RWD != COHESIVE EXPERIENCE

I needn't expound on the virtues of responsive web design (RWD). You've likely already encountered more than a career's worth on the topic. This is a good thing. Count me in as one of its biggest fans.

However, if we are to sing the praises of RWD, we must also acknowledge its shortcomings. One of these is that RWD ends where the browser ends. For all its goodness, RWD really has no bearing on native apps or any other experiences that take place outside the browser. This makes it challenging, therefore, to create cohesion for multi-screen users if RWD is the only response to "let's make it work everywhere."

We need something that incorporates the spirit of RWD while unifying all touchpoints for the entire user experience—single device or several devices, in browser or sans browser, native app or otherwise.

I call this *cohesive UX*, and I believe it's the next era of successful user experiences.

TOWARD A UNIFIED WHOLE

Simply put, the goal of cohesive UX is to deliver a consistent, unified user experience regardless of where the experience begins, continues, and ends.

Two facets are vital to cohesive UX:

1. Function *and* form
2. Data symmetry

Let's examine each of these.

Function AND form

Function over form, of course. Right? Not so fast, kiddo.

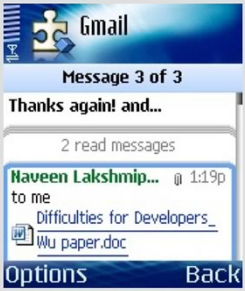
Consider Bruce Lawson's dad. After receiving an Android phone for Christmas and thumbing through his favorite sites, he was puzzled why some looked different from their counterparts on the desktop. "When a site looked radically different," Bruce observed, "he'd check the URL bar to ensure that he'd typed in the right address. In short,

he found RWD to be confusing and it meant he didn't trust the site." A lack of cohesive form led to a jarring experience for Bruce's dad.

Now, if I appear to be suggesting websites must look the same in every browser—you already learned they needn't—know that I recognize the importance of context, especially in regards to mobile. I made a case for this more than seven years ago.

Rather, cohesive UX suggests that *form* deserves the same respect as *function* when crafting user experiences that span multiple screens or devices. And users are increasingly comfortable traversing media. For example, more than 40% of adults in the U.S. owning more than one device start an activity on one screen and finish it on another, according to a study commissioned by Facebook. I suspect that percentage will only increase in 2015, and I suspect the tech-affluent readers of 24 ways are among the 40%.

There are countless examples of cohesive form and function. Consider Gmail, which displays email conversations visually as a stack that can be expanded and collapsed like the bellows of an accordion. This visual metaphor has been consistent in virtually any instance of Gmail—website or app—since at least 2007 when I captured this screenshot on my Nokia 6680:



Screenshot captured while authoring Mobile Web Design (2007). Back then we didn't call this an app, but rather a 'smart client'.

When the holistic experience is cohesive as it is with Gmail, users' mental models and even muscle memory are preserved.[1] Functionality and aesthetics align with the expectations users have for how things should function and what they should look like. In other words, the experience is roughly the same across screens.

But don't be ridiculous, peoples. Note that I said "roughly." It's important to avoid mindless replication of aesthetics and functionality for the sake of cohesion. Again, the goal is a unified whole, not a carbon copy. Affordances and concessions should be made as context and intuition require. For example, while Facebook users are accustomed to top-aligned navigation in the browser, they encounter bottom-aligned navigation in the iOS app as justified by user testing:

> The iOS app model has held up despite many attempts to better it: http://t.co/rSMSAqeh9m pic.twitter.com/mBp36lAEgc
>
> — Luke Wroblewski (@lukew) December 10, 2014

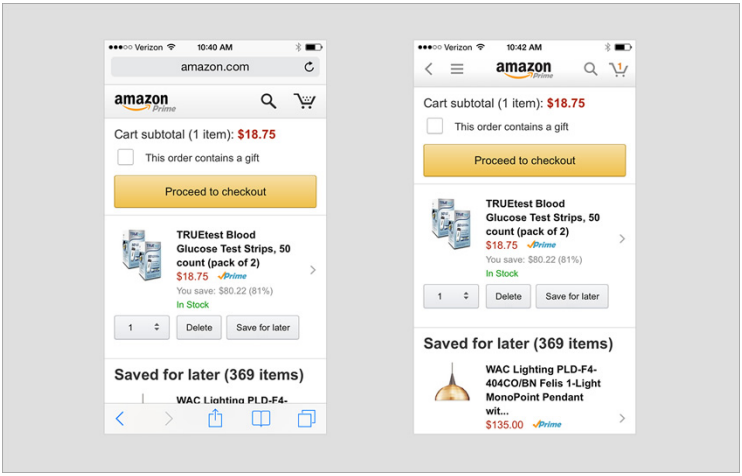
Despite the (rather minor) lack of consistency in navigation placement, other elements such as icons, labels, and color theme work in tandem to produce a unified, holistic whole.

Data symmetry

Data symmetry involves the repetition, continuity, or synchronicity of data across screens, devices, and platforms. As regards cohesive UX, data includes not just the material (such as an article you're writing on Medium) but also the actions that can be performed on or with that material (such as Medium's authoring tools). That is to say, "sync verbs, not just nouns" (Josh Clark).

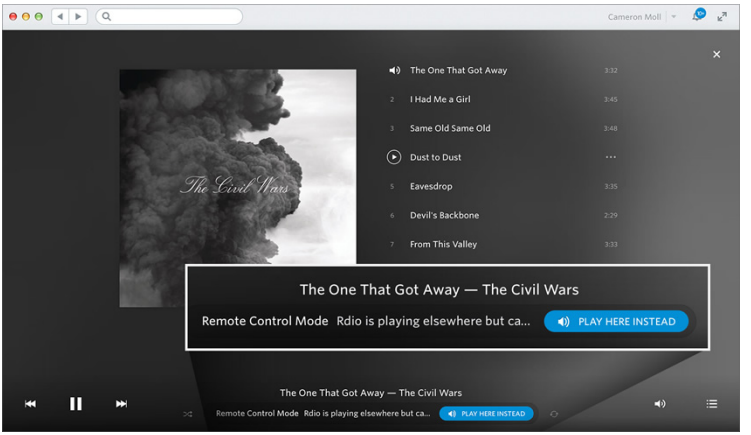
In my estimation, Amazon is an archetype of data symmetry, as is Rdio. When logged in, data is shared across virtually any device of any kind, irrespective of using a browser or native app. Add a product to your Amazon cart from your phone during the morning commute, and finish the transaction at work on your laptop. Easy peasy.

Amazon's aesthetics are crazy cohesive, to boot:



Amazon web (left) and native app (right).

With Rdio, not only are playlists and listening history synced across screens as you would expect, but the cohesion goes even further. Rdio's remote control feature allows you to control music playing on one device using another device, all in real time.



Rdio's remote control feature, as viewed on my MacBook while music plays on my iMac.

At my office I often work from my couch using my MacBook, but my speakers are connected to my iMac. When signed in to Rdio on both devices, my MacBook serves as proxy for controlling Rdio on my iMac, much the same as any Yosemite-enabled device can serve as proxy for an incoming iPhone call.

Me, in my office. Note the iMac and speakers at far right.

This is a brilliant example of cohesive design, and it's executed entirely via the cloud.

THINGS TO CONSIDER

Consider the following when crafting cohesive experiences:

1. **Inventory the elements that comprise your product experience, and *cohesify* them.**[2]
Consider things such as copy, tone, typography, iconography, imagery, flow, placement, brand identification, account data, session data, user

preferences, and so on. Then, create cohesion among these elements to the greatest extent possible, while adapting to context as needed.

2. **Store session data in the cloud rather than locally.** For example, avoid using browser cookies to store shopping cart data, as cookies are specific to a single browser on a single device. Instead, store this data in the cloud so it can be accessed from other devices, as well as beyond the browser.

3. **Consider using web views when developing your native app.** "You're already using web apps in native wrappers without even noticing it," Lukas Mathis contends. "The fact that nobody even notices, the fact that this isn't a story, shows that, when it comes to user experience, web vs. native doesn't matter anymore." Web views essentially allow you to display HTML content inside a native wrapper. This can reduce the time and effort needed to make the overall experience cohesive. So whereas the navigation bar may be rendered by the app, for example, the remaining page display may be rendered via the web. There's readily accessible documentation for using web views in C++, iOS, Android, and so forth.

NATURE IS CALLING

Returning to the example of Yosemite and sychronized phone calls, is it really that bizarre in light of cohesive UX? Perhaps at first. But I suspect that, over time, Yosemite's cohesiveness — and the cohesiveness of other examples like the ones we've discussed here — will become not only more natural but more commonplace, too.

[1] I browse Flipboard on my iPad nearly every morning as part of my breakfast routine. Swiping horizontally advances to the next page. Countless times I've done the same gesture in Flipboard for iPhone only to have it do nothing. This is because the gesture for advancing is vertical on phones. I'm so conditioned to the horizontal swipe that I often fail to make the switch to vertical swipe, and apparently others suffer from the same muscle memory, too.

[2] *Cohesify* isn't a thing. But chances are you understood what I meant. Yay neologism!

ABOUT THE AUTHOR

Cameron Moll is the founder of Authentic Jobs, maker of Structures in Type, and author of *Mobile Web Design* (2007). He resides in Sarasota, Florida with his wife and five sons. Find him on Twitter at @cameronmoll.

Cameron is looking to share the principles of Cohesive UX in 2015. Please get in touch if you'd like to have him speak at your conference.