



24

2015

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.

2015

Ports and protocols were the name of the game, with swathes of the web switching to HTTPS connections. HTTP2 also started to gain adoption, and in doing so turned all we had learned about performance optimisation on its head. 24 ways saw increasing exploration of animation on the web, as well as renewed interest in accessibility, style guides and progressive enhancement.

Animating Your Brand	5
Being Customer Supportive	25
How to Do a UX Review	37
Get Expressive with Your Typography	54
Universal React	65
Bringing Your Code to the Streets	87
Git Rebasing: An Elfin Workshop Workflow	99
Helping VIPs Care About Performance	115

Animation in Responsive Design.....	122
Putting My Patterns through Their Paces.....	132
Upping Your Web Security Game	144
Be Fluid with Your Design Skills: Build Your Own Sites	155
Designing with Contrast.....	162
What I Learned about Product Design This Year	173
Grid, Flexbox, Box Alignment: Our New System for Layout	182
Beyond the Style Guide	202
The Accessibility Mindset.....	216
Cooking Up Effective Technical Writing	226
Being Responsive to the Small Things.....	246
Make a Comic.....	257
What's Ahead for Your Data in 2016?	268
How Tabs Should Work.....	278
Blow Your Own Trumpet.....	294
Solve the Hard Problems.....	301

1. Animating Your Brand

Donovan Hutchinson

24ways.org/201501

Let's talk about how we add animation to our designs, in a way that's consistent with other aspects of our brand, such as fonts, colours, layouts and everything else.

Animating is fun. Adding animation to our designs can bring them to life and make our designs stand out. Animations can show how the pieces of our designs fit together. They provide context and help people use our products.

All too often animation is something we tack on at the end. We put a transition on a modal window or sliding menu and we often don't think about whether that animation is consistent with our overall design.

STYLE GUIDES TO THE RESCUE

A style guide is a document that **establishes and enforces style to improve communication**. It can cover anything from typography and writing style to ethics and other,

broader goals. It might be a static visual document showing every kind of UI, like in the Codecademy.com redesign shown below.



1-1. UI toolkit from “Reimagining Codecademy.com” by @mslima

It might be a technical reference with code examples. CodePen’s new design patterns and style guide is a great example of this, showing all the components used throughout the website as live code.



1-2. CodePen's design patterns and style guide

A style guide gives a wide view of your project, it maintains consistency when adding new content, and we can use our style guide to present animations.

LIVING DOCUMENTS

Style guides don't need to be static. We can use them to show movement. We can share CSS keyframe animations or transitions that can then go into production. We can also explain why animation is there in the first place.

Just as a style guide might explain why we chose a certain font or layout, we can use style guides to explain the intent behind animation. This means that if someone else wants to create a new component, they will know why animation applies.

If you haven't yet set up a style guide, you might want to take a look at **Pattern Lab**. It's a great tool for setting up your own style guide and includes loads of design patterns to get started.

There are many style guide articles linked from the excellent, open sourced, **Website Style Guide Resources**. Anna Debenham also has an excellent **pocket book** on the subject.

ADDING ANIMATION

Before you begin throwing animation at all the things, establish the character you want to convey.



1-3. Andrex Puppy (British TV ad from 1994)

List some words that describe the character you're aiming for. If it was the Andrex brand, they might have gone for: fun, playful, soft, comforting.

Perhaps you're aiming for something more serious, credible and authoritative. Or maybe exciting and intense, or relaxing and meditative. For each scenario, the animations that best represent these words will be different.

In the example below, two animations both take the same length of time, but use different timing functions. One eases, and the other bounces around. Either might be good, depending on your needs.



1-4. Timing functions ([CodePen](#))

EXAMPLE: KITMAN LABS

Working with Kitman Labs, we spent a little time working out what words best reflected the brand and came up with the following:

- Scientific
- Precise
- Fast
- Solid
- Dependable
- Helpful
- Consistent
- Clear

With such a list of words in hand, we design animation that fits. We might prefer a tween that moves quickly to its destination over one that drifts slowly or bounces.

We can use the list when justifying our use of animation, such as when it helps our customers understand the context of data on the page. Or we may even choose not to animate, when that might make the message inconsistent.

CREATE GUIDELINES

If you already have a style guide, adding animation could begin with creating an overview section.

One approach is to create a local website and share it within your organisation. We recently set up a local site for this purpose.

« Brand Guidelines

Animation

The goal of animation

Subtle is better
Accessibility

Character

Kitman Labs

Applying animation qualities

Building blocks

Position

Scale

Rotation

Opacity

Hierarchy

Tools

CSS & JavaScript

Native animations

Examples

Animation

Just as we specify fonts, colours and visual layout styles, we can also design the way our interfaces move. Animation is as much part of the character of our brand as anything else, and this document covers how we use it.

The goal of animation

The goal of animation is to help the interfaces we design communicate with our customers. Animation can show where information comes from, highlight when on-screen content changes, and connect with the way people are used to interacting with the real world.

In reality, nothing suddenly appears or disappears. While it's trivial to have modal popups or lists of items suddenly change online, the effect is jarring and not something we've evolved to expect. If content suddenly appears in a subtle location, we might miss it.

We can instead use animation to give context and help people better understand what's happening.

1-5. A recent project's introduction to the topic of animation

This document becomes a reference when adding animation to components. Include links to related resources or examples of animation to help demonstrate the animation style you want.

get the most ideas out of your head fastest. Iterate and refine an animation before it gets anywhere near production.

BUILD UP A COLLECTION

Build up your guide, one animation at a time.

Some people prefer to loosely structure a guide with places to put things as they are discovered or invented; others might build it one page at a time – it doesn't matter. The main thing is that you collect animations like you would trading cards. Or Pokemon. Keep them ready to play and deliver that explosive result.

You could include animated GIFs, or link to videos or even live webpages as examples of animation. The use of animation to help user experience is also covered nicely in Val Head's [UI animation and UX article on A List Apart](#).

What matters is that you create an organised place for them to be found. Here are some ideas to get started.

Logos and landmarks

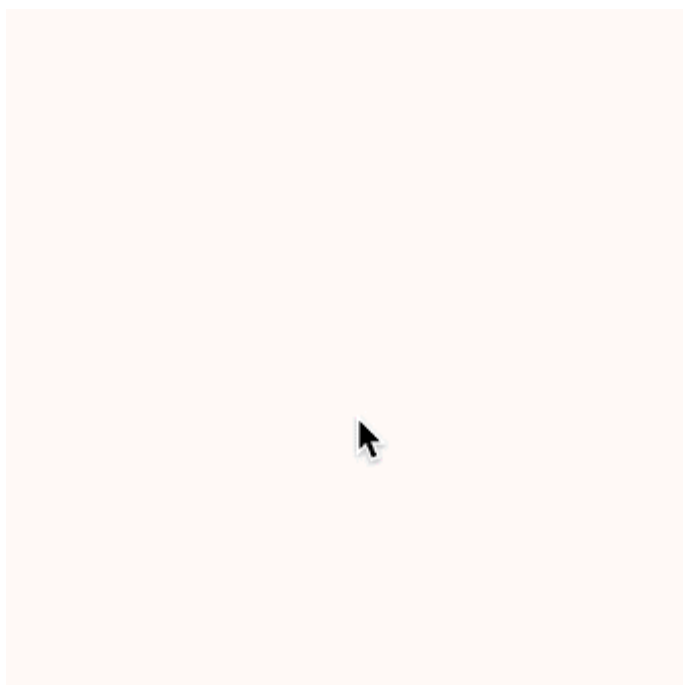
Many sites include some subtle form of animation in their logos. This can draw the eye, add some character, or bring a little liveliness to an otherwise static page. Yahoo and

Google have been experimenting with animation on their logos. Even a simple bouncing animation, such as the logo on Hop.ie, can add character.

1-7. The CSS-animated bouncer from Hop.ie

Content transitions

Adding content, removing content, showing and hiding messages are all opportunities to use animation. Careful and deliberate use of animation helps convey what's changing on screen.



1-8. Animating list items with CSS ([CSSAnimation.rocks](http://cssanimation.rocks))

For more detail on this, I also recommend “[Transitional Interfaces](#)” by Pasquale D’Silva.

Page transitions

On a larger scale than the changes to content, full-page transitions can smooth the flow between sections of a site. Medium’s article transitions are a good example of this.

Queequeg to care what god made him sink; said the savage, agonizingly lifting his hand up and down; "wedder Fejee god or Nantucket god; but de god wat made shark must be one dam Ingin."

It was a Saturday night, and such a Sabbath as followed! Ex officio professors of Sabbath breaking are all whalemen. The ivory Pequod was turned into what seemed a shamble; every sailor a butcher. You would have thought we were offering up ten thousand red oxen to the sea gods.



1-9. Medium-style page transition (Tympanus.net)

Preparing a layout before the content arrives

We can use animation to draw a page before the content is ready, such as when a page calls a server for data before showing it.

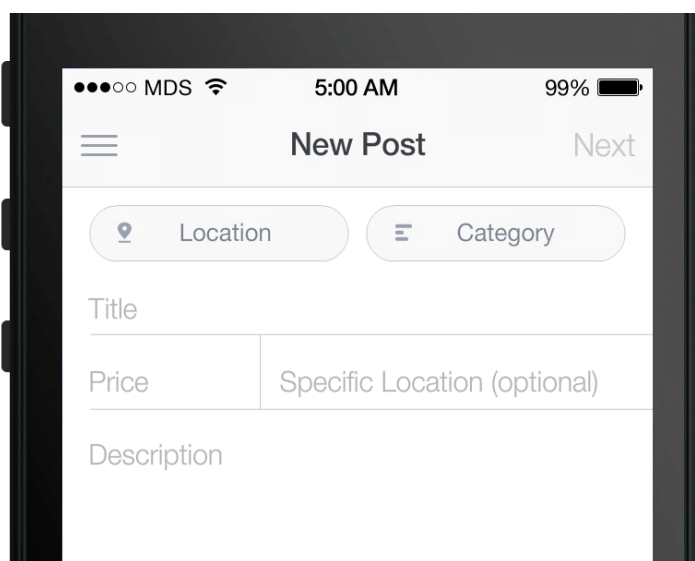


1-10. Optimistic loading grid ([CodePen](#))

Sometimes it's good to show something to let the user know that everything's going well. A short animation could cover just enough time to load the initial content and make the loading transition feel seamless.

Interactions

Hover effects, dropdown menus, slide-in menus and active states on buttons and forms are all opportunities. Look for ways you can remove the sudden changes and help make the experience of using your UI feel smoother.



1-11. Form placeholder animation (Studio MDS)

KEEP ANIMATION VISIBLE

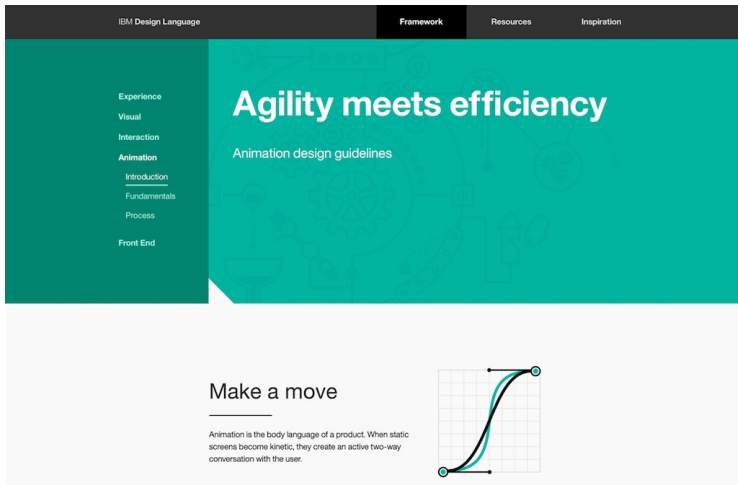
It takes continuous effort to maintain a style guide and keep it up to date, but it's worth it. Make it *easy* to include animation and related design decisions in your documentation and you'll be more likely to do so. If you can make it fun, and be proud of the result, better still.

When updating your style guide, be sure to show the animations at the same time. This might mean animated GIFs, videos or live embedded examples of your components.

By doing this you can make animation integral to your design process and make sure it stays relevant.

INSPIRATION AND RESOURCES

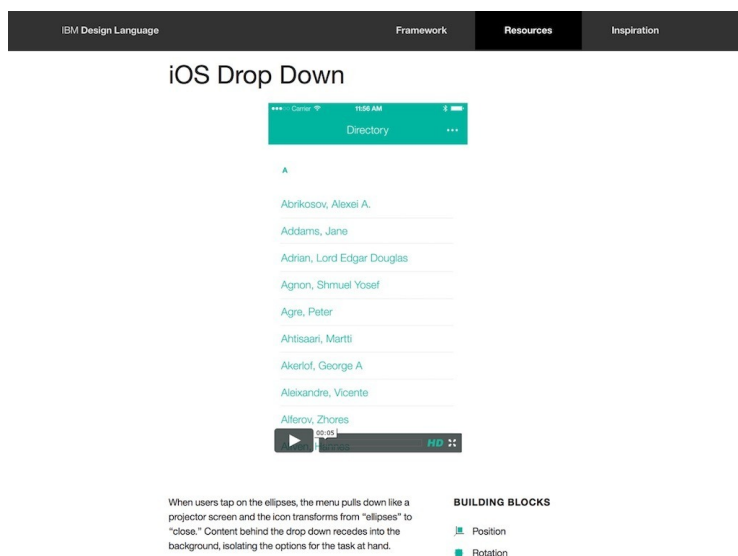
There are loads of great resources online to help you get started. One of my favourites is IBM's design language site.



1-12. IBM's design language: animation design guidelines

IBM describes how animation principles apply to its UI work and components. They break down the animations into five categories of animations and explain how they apply to each example.

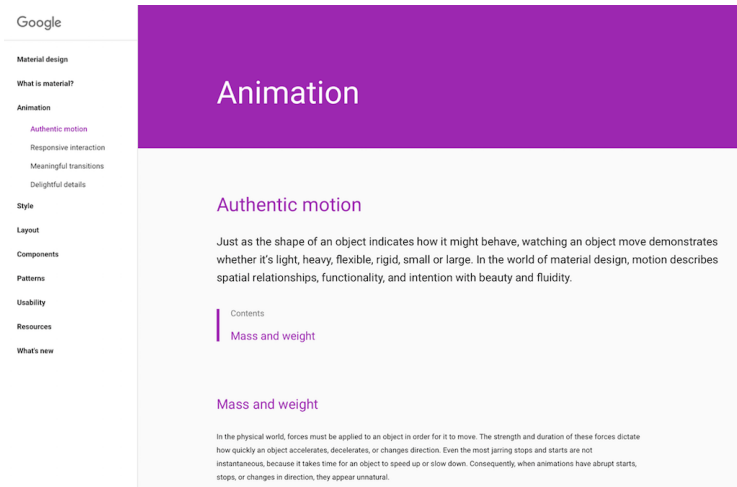
The site also includes an animation library with example videos of animations and links to source code.



1-13. Example component from IBM's component library

The way IBM sets out its aims and methods is helpful not only for their existing designers and developers, but also helps new hires. Furthermore, it's a good way to show the world that IBM cares about these details.

Another popular animation resource is Google's material design.



1-14. Google's material design documentation

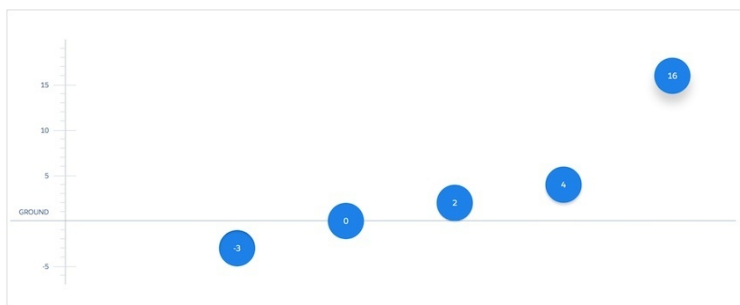
Google's guidelines cover everything from understanding easing through to creating engaging and useful mobile UI.

This approach is visible across many of Google's apps and software, and has influenced design across much of the web. The site is helpful both for learning about animation and as an showcase of how to illustrate examples.

Frameworks

If you don't want to create everything from scratch, there are resources you can use to start using animation in your UI. One such resource is Salesforce's **Lightning design** system.

The system goes further than most guides. It includes a downloadable framework for adding animation to your projects. It has some interesting concepts, such as elevation settings to handle positioning on the z-axis.



1-15. Example of elevation from Salesforce's Lightning design system

You should also check out [Animate.css](#).

Animate.css

Just-add-water CSS animations

bounce

▼

Animate it

[Download Animate.css](#) or [View on GitHub](#)

Another thing from [Daniel Eden](#).

1-16. “Just add water” — Animate.css

Animate.css gives you a set of predesigned animations you can apply to page elements using classes. If you use JavaScript to add or remove classes, you can then trigger complex animations. It also plays well with scroll-triggering, and tools such as **WOW.js**.

LEARN, EVOLVE AND MAKE IT YOUR OWN

There’s a wealth online of information and guides we can use to better understand animation. They can inspire and kick-start our own visual and animation styles. So let’s

think of the design of animations just as we do fonts, colours and layouts. Let's choose animation deliberately, making it part of our style guides.

Many thanks to Val Head for taking the time to proofread and offer great suggestions for this article.

ABOUT THE AUTHOR



Donovan Hutchinson is a front-end designer who loves making fun stuff for the web. He also writes tutorials on CSSAnimation.rocks. Find him on Twitter at [@donovanh](https://twitter.com/donovanh).

2. Being Customer Supportive

Elizabeth Galle

24ways.org/201502

Every day in customer support is an inbox, a Twitter feed, or a software forum full of new questions. Each is brimming with your customers looking for advice, reassurance, or fixes for their software problems. Each one is an opportunity to take a break from wrestling with your own troublesome tasks and assist someone else in solving theirs.

Sometimes the questions are straightforward and can be answered in a few minutes with a short greeting, a link to a help page, or a prewritten bit of text you use regularly: how to print a receipt, reset a password, or even, sadly, close your account.

More often, a support email requires you to spend some time unpacking the question, asking for more information, and writing a detailed personal response, tailored to help that particular user on this particular day.

Here I offer a few of my own guidelines on how to make today's email the best support experience for both me and my customer. And even if you don't consider what you do to be customer support, you might still find the suggestions useful for the next time you need to communicate with a client, to solve a software problem with teammates, or even reach out and ask for help yourself.

(All the examples appearing in this article are fictional. Any resemblance to quotes from real, software-using persons is entirely coincidental. Except for the bit about Star Wars. That happened.)

WHO'S TAHT GIRL

I'll be honest: I briefly tried making these recommendations into a clever mnemonic like FAST (facial drooping, arm weakness, speech difficulties, time) or PAD (pressure, antiseptic, dressing). But instead, you get **TAHT**: *tone, ask, help, thank*. Ah, well.

As I work through each message in my support queue, I

- listen to the *tone* of the email
- *ask* clarifying questions
- bring in extra *help* as needed
- and *thank* the customer when the problem is solved.

Let's open an email and get started!

LEAVE YOUR MESSAGE AT THE SOUND OF THE TONE

With our enthusiasm for emoji, it can be very hard to infer someone's tone from plain text. How much time have you spent pondering why your friend responded with "Thanks." instead of "Thanks!"? I mean, why didn't she :grin: or :wink: too?

Our support customers, however, are often direct about how they're feeling:

I'm working against a deadline. Need this fixed ASAP!!!!

This hasn't worked in a week and I am getting really frustrated.

I've done this ten times before and it's always worked. I must be missing something simple.

They want us to understand the urgency of this from their point of view, just as much as we want to help them in a timely manner. How this information is conveyed gives us an instant sense of whether they are frustrated, angry, or confused—and, just as importantly, *how* frustrated-angry-confused they are.

Listen to this tone before you start writing your reply. Here are two ways I might open an email:

1. "I'm sorry that you ran into trouble with this."

2. “Sorry you ran into trouble with this!”

The content is largely the same, but the tone is markedly different. The first version is a serious, staid reaction to the problem the customer is having; the second version is more relaxed, but no less sincere.

Matching the tone to the sender’s is an important first step. Overusing exclamation points or dropping in too-casual language may further upset someone who is already having a crummy time with your product. But to a cheerful user, a formal reply or an impersonal form response can be off-putting, and damage a good relationship.

When in doubt, I err on the side of being too formal, rather than sending a reply that may be read as flip or insincere. But whichever you choose, matching your correspondent’s tone will make for a more comfortable conversation.

CATCH THE BALL AND THROW IT BACK

Once you’ve got that tone on lock, it’s time to tackle the question at hand. Let’s see what our customer needs help with today:

I tried everything in the troubleshooting page but I can’t get it to work again. I am on a Mac. Please help.

Hmm, not much information here. Now, if I got this short email after helping five other people with the same problem on Mac OS X, I would be sorely tempted to send this customer that common solution in my first reply. I've found it's important to resist the urge to assume this sixth person needs the same answer as the other five, though: there isn't enough to connect this email to the ones that came before hers.

Instead, ask a few questions to start. Invest some time to see if there are other symptoms in common, like so:

I'm sorry that you ran into trouble with this!
I'll need a little more information to see what's happening here.

[questions]

Thank you for your help.

Those questions are customized for the customer's issue as much as possible, and can be fairly wide-ranging. They may include asking for log files, getting some screenshots, or simply checking the browser and operating system version she's using. I'll ask anything that might make a connection to the previous cases I've answered—or, just as importantly, confirm that there isn't a connection. What's more, a few well-placed questions may save us both from pursuing the wrong path and building additional frustration.

(A note on that closing: “Thank you for your help”—I often end an email this way when I’ve asked for a significant amount of follow up information. After all, I’m imposing on my customer’s time to run any number of tests. It’s a necessary step, but I feel that thanking them is a nice acknowledgment we’re in this *together*.)

Having said that, though, let’s bring tone back into the mix:

I tried everything in the troubleshooting but I can’t get it to work again. I am on a Mac. I’m working against a deadline. Need this fixed ASAP!!!!

This customer wants answers *now*. I’ll still ask for more details, but would consider including the solution to the previous problem in my initial reply as well. (But only if doing so can’t make the situation worse!)

I'm sorry that you ran into trouble with this!
I'll need a little more information to see what's
happening here.

[questions]

If you'd like to try something in the meantime,
delete the file named *xyz.txt*. (If this isn't the
cause of the problem, deleting the file won't
hurt anything.) Here's how to find that file on
your computer: [steps]

Let me know how it goes!

In the best case, the suggestion works and the customer is on her way. If it doesn't solve the problem, you will get more information in answer to your questions and can explore other options. And you've given the customer an opportunity to be involved in fixing the issue, and some new tools which might come in handy again in the future.

BRING IN HELP

The support software I use counts how many emails the customer and I have exchanged, and reports it in a summary line in my inbox. It's an easy, passive reminder of how long the customer and I have been working together on a problem, especially first thing in the morning when I'm reacquainting myself with my open support cases.

Three is the smallest number I'll see there: the customer sends the initial question (1 email); I reply with an answer (2 emails); the customer confirms the problem is solved (3 emails). But the most complicated, stickiest tickets climb into double-digit replies, and anything that stretches beyond a dozen is worthy of a cheer in **Slack** when we finally get to the root of the problem and get it fixed.

While an extra round of questions and answers will nudge that number higher, it gives me the chance to feel out the technical comfort level of the person I'm helping. If I ask the customer to send some screenshots or log files and he isn't sure how to do that, I will use that information to adjust my instructions on next steps. I may still ask him to try running a **traceroute** on his computer, but I'll break down the steps into a concise, numbered list, and attach screenshots of each step to illustrate it.

If the issue at hand is getting complicated, take note if the customer starts to feel out of their depth technically—either because they tell you so directly or because you sense a shift in tone. If that happens, propose bringing some outside help into the conversation:

Do you have a network firewall or do you use any antivirus software? One of those might be blocking a connection that the software needs to work properly; here's a list of the required connections [link]. If you have an IT department in-house, they should be able to help confirm that none of those are being blocked.

or:

This error message means you don't have permission to install the software on your own computer. Is there a systems administrator in the office that may be able to help with this?

For email-based support cases, I'll even offer to add someone from their IT department to the thread, so we can discuss the problem together rather than have the customer relay questions and answers back and forth.

Similarly, there are occasionally times when my way of describing things doesn't fit how the customer understands them. Rather than bang our heads against our keyboards, I will ask one of my support colleagues to join the conversation from our side, and see if he can explain things more clearly than I've been able to do.

WE APPRECIATE YOUR BUSINESS. PLEASE CALL AGAIN

And then, o frabjous day, you get your reward: the reply which says the problem has been solved.

That worked!! Thank you so much for saving my day!

I wish I could send you some cookies!

If you were here, I would give you my tickets to Star Wars.

[Reply is an animated gif.]

Sometimes the reply is a bit more understated:

That fixed it. Thanks.

Whether the customer is elated, satisfied, or frankly happy to be done with emailing support, I like to close longer email threads or short, complicated issues with a final thanks and reminder that we're here to help:

Thank you for the update; I'm glad to hear that solved the problem for you! I hope everything goes smoothly for you now, but feel free to email us again if you run into any other questions or problems. Best,

Then mark that support case closed, and move on to the next question. Because even with the most thoughtfully designed software product, there will always be customers with questions for your capable support team to answer.

TONE, ASK, HELP, THANK

So there you have it: **TAHT**. Pay attention to *tone*; *ask* questions; bring in *help*; *thank* your customer.

(Lack of) catchy mnemonics aside, good customer support is about listening, paying attention, and taking care in your replies. I think it can be summed up beautifully by **this quote** from Pamela Marie (as tweeted by Chris Coyier):

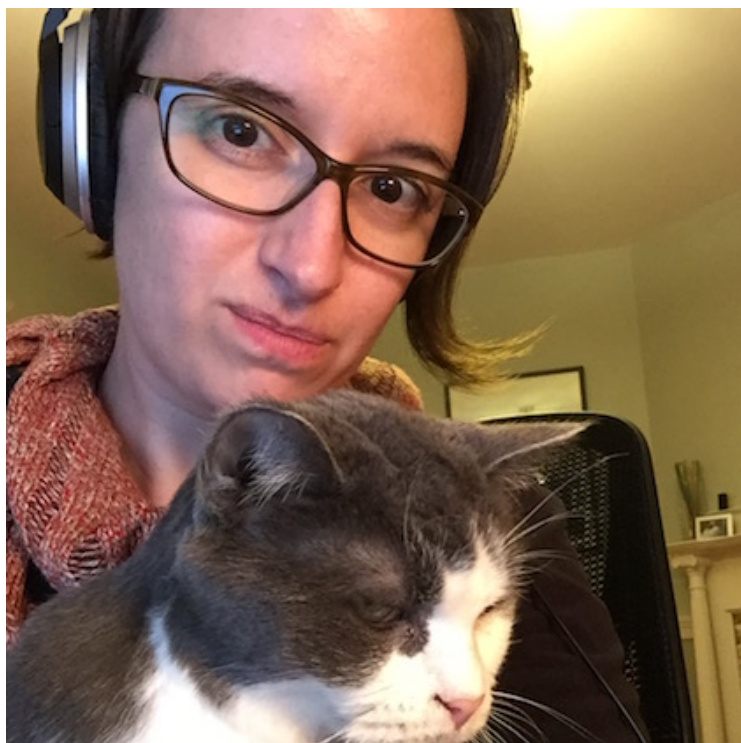
Golden rule asking a question: imagine trying to answer it

Golden rule in answering: imagine getting your answer

You and your teammates are applying a variation of this **golden rule** in every email you write. You're the software ambassadors to your customers and clients. You get the brunt of the problems and complaints, but you also get to help fix them. You write the apologies, but you also have the chance to make each person's experience with your company or product a little bit better for next time.

I hope that your holidays are merry and bright, and may all your support inboxes be light.

ABOUT THE AUTHOR



Elizabeth Galle got her start in customer support behind the counter of her grandfather's delicatessen. She's since moved from sandwiches to software, but the principles are about the same. Liz talks about her cat & quotes West Wing episodes as @drinkerthinker, and supports Adobe Typekit with the team at @typekit.

3. How to Do a UX Review

Joe Leech

24ways.org/201503

A UX review is where an expert goes through a website looking for usability and experience problems and makes recommendations on how to fix them.

I've completed a number of UX reviews over my twelve years working as a user experience consultant and I thought I'd share my approach.

I'll be talking about reviewing websites here; you can adapt the approach for web apps, or mobile or desktop apps.

WHY CONDUCT A REVIEW

Typically, a client asks for a review to be undertaken by a trusted and, ideally, detached third party who either works for an agency or is a freelancer. Often they may ask a new member of the UX team to complete one, or even

set it as a task for a job interview. This indicates the client is looking for an objective view, seen from the outside as a user would see the website.

I always suggest conducting some user research rather than a review. Users know their goals and watching them make (what you might think of as) mistakes on the website is invaluable. Conducting research with six users can give you six hours' worth of review material from six viewpoints. In short, user research can identify more problems and show how common those problems might be.

There are three reasons, though, why a review might better suit client needs than user research:

1. Quick results: user research and analysis takes at least three weeks.
2. Limited budget: the £6–10,000 cost to run user research is about twice the cost of a UX review.
3. Users are hard to reach: in the business-to-business world, reaching users is difficult, especially if your users hold senior positions in their organisations. Working with consumers is much easier as there are often more of them.

There is some debate about the benefits of user research over UX review. In my experience you learn far more from research, but opinions differ.

BE OBJECTIVE

The number one mistake many UX reviewers make is reporting back the issues they identify as their opinion. This can cause credibility problems because you have to keep justifying why your opinion is correct.

I've had the most success when giving bad news in a UX review and then finally getting things fixed when I have been as objective as possible, offering evidence for why something may be a problem.

To be objective we need two sources of data: numbers from analytics to appeal to reason; and stories from users in the form of personas to speak to emotions. Highlighting issues with dispassionate numerical data helps show the extent of the problem. Making the problems more human using personas can make the problem feel more real.

NUMBERS FROM ANALYTICS

The majority of clients I work with use Google Analytics, but if you use a different analytics package the same concepts apply. I use analytics to find two sets of things.

1. Landing pages and search terms

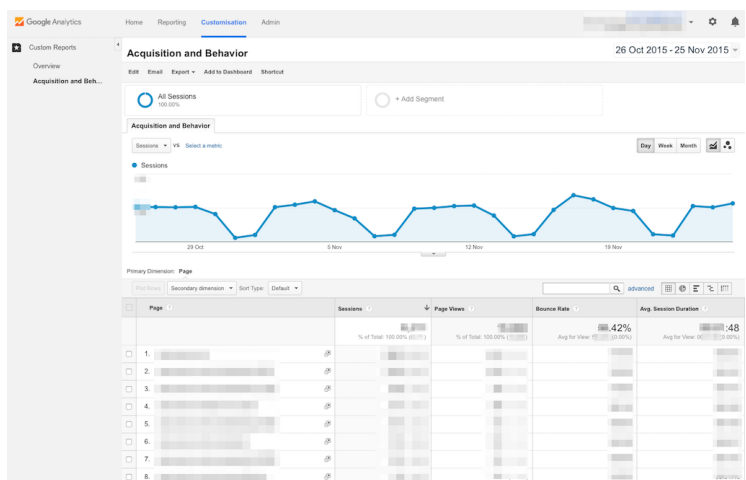
Landing pages are the pages users see first when they visit a website – more often than not via a Google search.

Landing pages reveal user goals. If a user landed on a page called ‘Yellow shoes’ their goal may well be to find out about or buy some yellow shoes.

It would be great to see all the search terms bringing people to the website but in 2011 Google stopped providing search term data to (rightly!) protect users’ privacy. You can get some search term data from Google Webmaster tools, but we must rely on landing pages as a clue to our users’ goals.

The thing to look for is high-traffic landing pages with a high bounce rate. **Bounce rate** is the percentage of visitors to a website who navigate away from the site after viewing only one page. A high bounce rate (over 50%) isn’t good; above 70% is bad.

To get a list of high-traffic landing pages with a high bounce rate install this bespoke report.

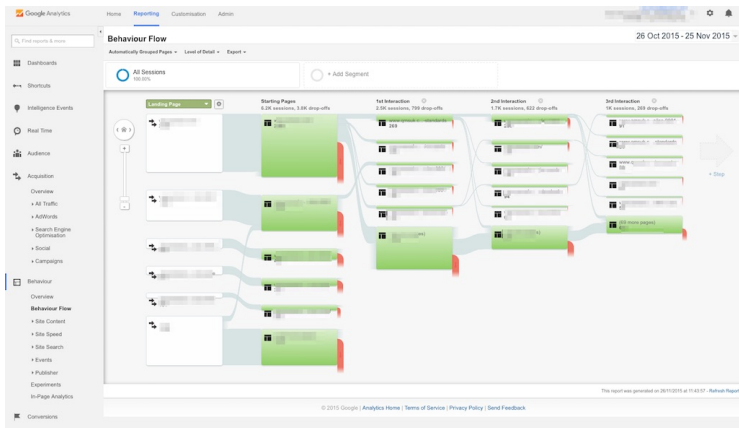


3-1. Google Analytics showing landing pages ordered by popularity and the bounce rate for each.

This is the list of pages with high demand and that have real problems as the bounce rate is high. This is the main focus of the UX review.

2. User flows

We have the beginnings of the user journey: search terms and initial landing pages. Now we can tap into the really useful bit of Google Analytics. Called *behaviour flows*, they show the most common order of pages visited.

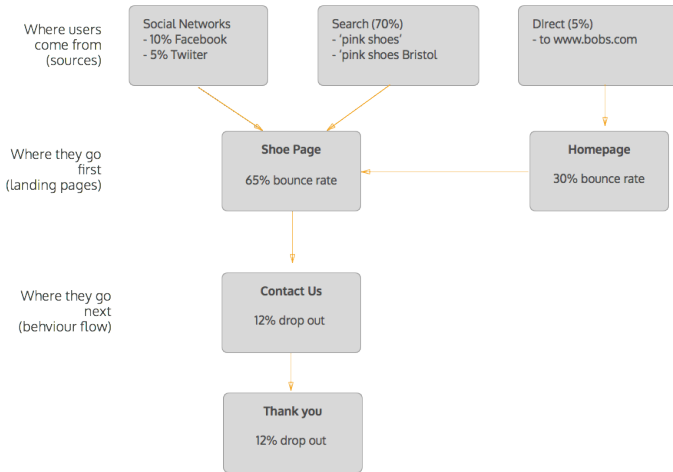


3-2. Behaviour flows from Google Analytics, showing the routes users took through the website.

Here we can see the second and third (and so on) pages users visited. Importantly, we can also see the drop-outs at each step.

If your client has it set up, you can also set goal pages (for example, a post-checkout contact us and thank you page). You can then see a similar view that tracks back from the goal pages. If your client doesn't have this, suggest they set up goal tracking. It's easy to do.

We now have the remainder of the user journey.



3-3. A user journey

Expect the work in analytics to take up to a day.

We may well identify more than one user journey, starting from different landing pages and going to different second- and third-level pages. That's a good thing and shows we have different user types. Talking of user types, we need to define who our users are.

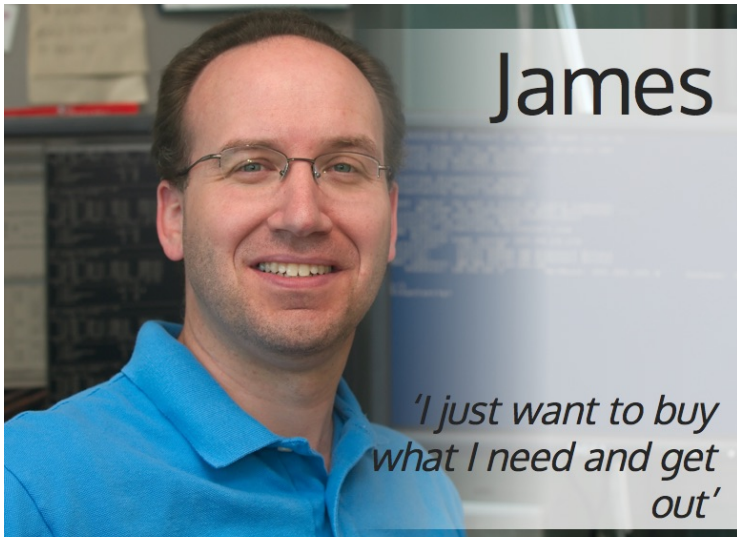
PERSONAS

We have some user journeys and now we need to understand more about our users' motivations and goals.

I have a love-hate relationship with personas, but used properly these portraits of users can help bring a human touch to our UX review.

I suggest using a very cut-down view of a persona. My old friends Steve Cable and Richard Caddick at cxpartners have a great free template for personas from their book *Communicating the User Experience*.

The first thing to do is find a picture that represents that persona. Don't use crappy stock photography – it's sometimes hard to relate to perfect-looking people) – use authentic-looking people. Here's a good collection of persona photos.



🔑 Goals

- Wants to learn about pink shoes
- Avoid spending time browsing (short visit duration, visits just one product page)
- Drops out at contact page as there is no buy it now option

✓ We must

- Clearly show key features of each product
- Provide easy comparison between products
- Allow him to buy online

✗ We must not

- Don't delay him by contacting him the day after he wants to buy
- Don't focus on fluffy descriptions of products
- Don't overwhelm him with choice

Photo source <http://upload.wikimedia.org/>

3-4. An example persona.

The personas have three basic attributes:

1. Goals: we can complete these drawing on the analytics data we have (see example).
2. Musts: things we have to do to meet the persona's needs.
3. Must nots: a list of things we really shouldn't do.

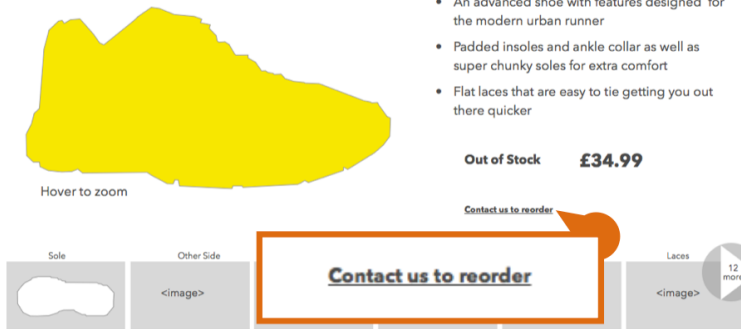
Completing points 2 and 3 can often be done during the writing of the report.

Let's take an example. We know that the search term 'yellow shoes' takes the user to the landing page for yellow shoes. We also know this page has a high bounce rate, meaning it doesn't provide a good experience.

With our expert hat on we can review the page. We will find two types of problem:

1. Usability issues: ineffective button placement or incorrect wording, links not looking like links, and so on.
2. Experience issues: for example, if a product is out of stock we have to contact the business to ask them to restock.

New Balance W980v2 Super Comfy Running Shoes



3-5. That link is very small and hard to see.

We could identify that the contact button isn't easy to find (a usability issue) but that's not the *real* problem here. That the user has to ask the business to restock the item is a bad user experience. We add this to our personas' must nots. The big experience problems with the site form the musts and must nots for our personas.

We now have a story around our user journey that highlights what is going wrong.

If we've identified a number of user journeys, multiple landing pages and differing second and third pages visited, we can create more personas to match. A good rule of thumb is no more than three personas. Any more and they lose impact, watering down your results.

Expect persona creation to take up to a day to complete.

LET'S START THE REVIEW

We take the user journeys and we follow them step by step, working through the website looking for the reasons why users drop out at each step. Using Keynote or PowerPoint, I structure the final report around the user journey with separate sections for each step.

For each step we'll find both usability and experience problems. Split the results into those two groups.

Usability problems are fairly easy to fix as they're often quick design changes. As you go along, note the usability problems in one place: we'll call this 'quick wins'. Simple quick fixes are a reassuring thing for a client to see and mean they can get started on stuff right away. You can mark the severity of usability issues. Use a scale from 1 to 3 (if you use 1 to 5 everything ends up being a 3!) where 1 is minor and 3 is serious.

Review the website on the device you'd expect your persona to use. Are they using the site on a smartphone? Review it on a smartphone.

I allow one page or slide per problem, which allows me to explain what is going wrong. For a usability problem I'll often make a quick wireframe or sketch to explain how to address it.

Product Page » Re-order link is hard to find

When an item is out of stock a link is shown to contact to reorder the item.

Severity: **Serious**

Quick Win

The link is not easy to see.

Recommendation: Make the link a button.



Customer Experience Issue

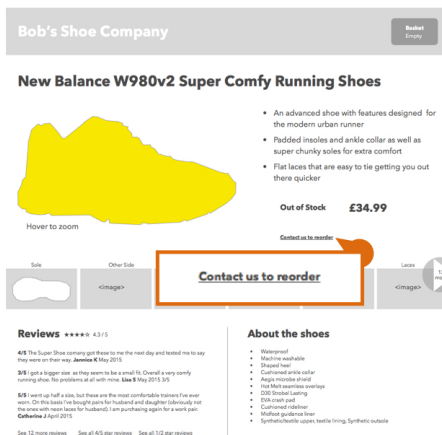
Asking the customer to re-order an out of stock item is not a normal or expected interaction.

James would not expect to have to reorder the item, he would expect it to be automated.



That's not like everyone else does it.

@mrjoe



3-6. A UX review slide displaying all the elements to be addressed. These slides may be viewed from across the room on a screen so zoom into areas of discussion.

(Quick tip: if you use Google Chrome, try Awesome Screenshot to capture screens.)

When it comes to the more severe experience problems – things like an online shop not offering next day delivery, or a business that needs to promise to get back to new customers within a few hours – these will take more than a tweak to the UI to fix.

Call these something different. I use the terms like *business challenges* and *customer experience issues* as they show that it will take changes to the organisation and its

processes to address them. It's often beyond the remit of a humble UX consultant to recommend how to fix organisational issues, so don't try.

Again, create a page within your document to collect all of the business challenges together.

Expect the review to take between one and three days to complete.

The final report should follow this structure:

- The approach
- Overview of usability quick wins
- Overview of experience issues
- Overview of Google Analytics findings
- The user journeys
- The personas
- Detailed page-by-page review (broken down by steps on the user journey)

There are two academic theories to help with the review.

Heuristic evaluation is a set of criteria to organise the issues you find. They're great for categorising the usability issues you identify but in practice they can be quite cumbersome to apply.

I prefer the more scientific and much simpler **cognitive walkthrough** that is focused on goals and actions.

A WORKSHOP TO GO THROUGH THE FINDINGS

The most important part of the UX review process is to talk through the issues with your client and their team.

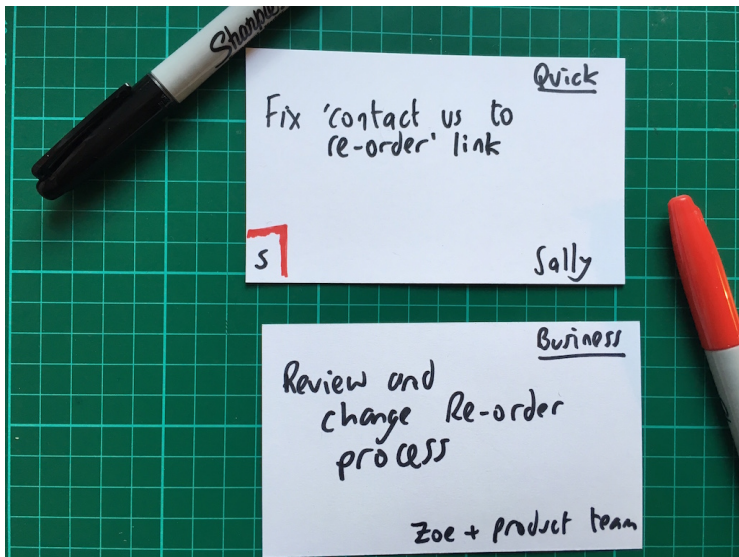
A document can only communicate a certain amount. Conversations about the findings will help the team understand the severity of the issues you've uncovered and give them a chance to discuss what to do about them.

Expect the workshop to last around three hours.

When presenting the report, explain the method you used to conduct the review, the data sources, personas and the reasoning behind the issues you found. Start by going through the usability issues. Often these won't be contentious and you can build trust and improve your credibility by making simple, easy to implement changes.

The most valuable part of the workshop is conversation around each issue, especially the experience problems. The workshop should include time to talk through each experience issue and how the team will address it.

I collect actions on index cards throughout the workshop and make a note of who will take what action with each problem.



3-7. Index cards showing the problem and who is responsible.

When talking through the issues, the person who designed the site is probably in the room – they may well feel threatened. So be nice. When I talk through the report I try to have strong ideas, weakly held.

At the end of the workshop you'll have talked through each of the issues and identified who is responsible for addressing them. To close the workshop I hand out the cards to the relevant people, giving them a physical reminder of the next steps they have to take.

That's my process for conducting a review. I'd love to hear any tips you have in the comments.

ABOUT THE AUTHOR



@MrJoe, Joe to his friends, is the author of the book **Psychology for Designers**.

A recovering neuroscientist, then a spell as a elementary school teacher, Joe started his UX career 12 years ago. He has worked with organisations like Disney, eBay, Glenfiddich and Marriott.

4. Get Expressive with Your Typography

Richard Rutter

24ways.org/201504

In 1955 Beatrice Warde, an American communicator on typography, published a series of essays entitled *The Crystal Goblet* in which she wrote, “People who love ideas must have a love of words. They will take a vivid interest in the clothes that words wear.” And with that proposition Warde introduced the idea that just as we judge someone based on the clothes they are wearing, so we make judgements about text based on the typefaces in which it is set.



4-1. Beatrice Warde. ©1970 Monotype Imaging Inc.

Choosing the same typeface as everyone else, especially if you're trying to make a statement, is like turning up to a party in the same dress; to a meeting in the same suit, shirt and tie; or to a craft ale dispensary in the same plaid shirt and turned-up skinny jeans.

But there's more to your choice of typeface than simply making an impression. In 2012 Jon Tan wrote on 24 ways about a scientific study called "The Aesthetics of Reading"

which concluded that “good quality typography is responsible for greater engagement during reading and thus induces a good mood.”

Furthermore, at this year’s **Ampersand conference** Sarah Hyndman, an expert in multisensory typography, discussed how typefaces can communicate with our subconscious. Sarah showed that different fonts could have an effect on how food tasted. A rounded font placed near a bowl of jellybeans would make them taste sweeter, and a jagged angular font would make them taste more sour.

The quality of your typography can therefore affect the mood of your reader, and your font choice directly affect the senses. This means you can manipulate the way people feel. You can change their emotional state through type alone. Now that’s a real superpower!

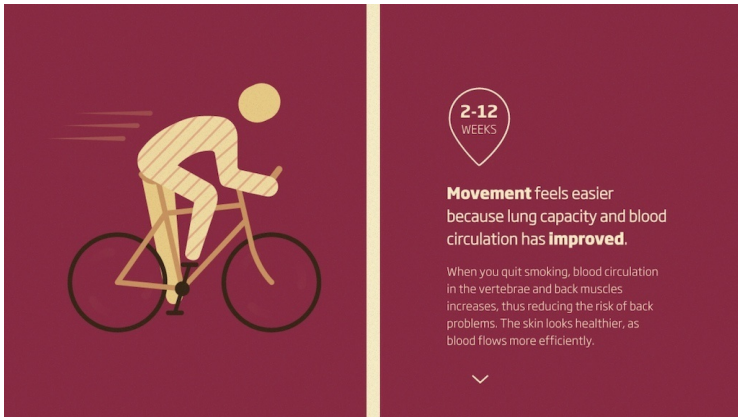
The effects of your body text design choices are measurable but subtle. If you really want to have an impact you need to think big. Literally. Display text and headings are your attention grabbers. They are your chance to interrupt, introduce and seduce.

Display text and headings set the scene and draw people in. Text set large creates an image that visitors *see* before they *read*, and that’s your chance to choose a typeface that immediately expresses what the text, and indeed the entire website, stands for. What expectations of the text

do you want to set up? Youthful enthusiasm?
Businesslike? Cutting-edge? Hipster? Sensible and
secure? Fun and informal? Authoritarian?

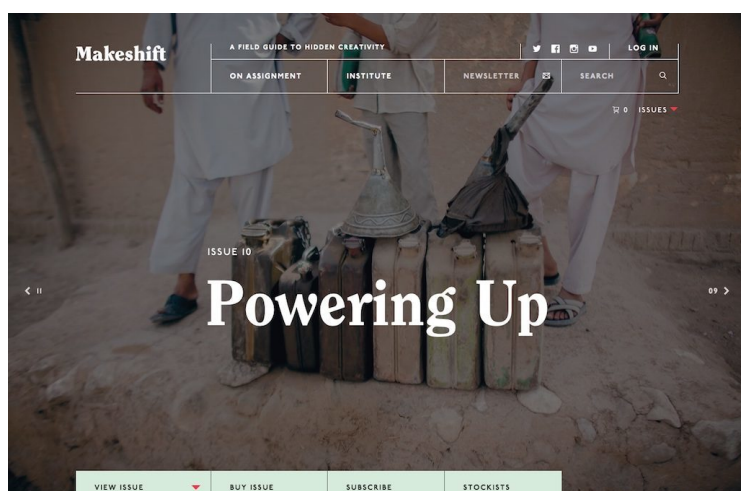
Typography conveys much more than just information. It imparts feeling, emotion and sentiment, and arouses preconceived ideas of trust, tone and content. Think about taking advantage of this by introducing impactful, expressive typography to your designs on the web. You can alter the way your reader feels, so what emotion do you want to provoke?

Maybe you want them to feel inspired like this stop smoking campaign:



4-2. helsenorge.no

Perhaps they should be moved and intrigued, as with
Makeshift magazine:



4-3. mkshft.org

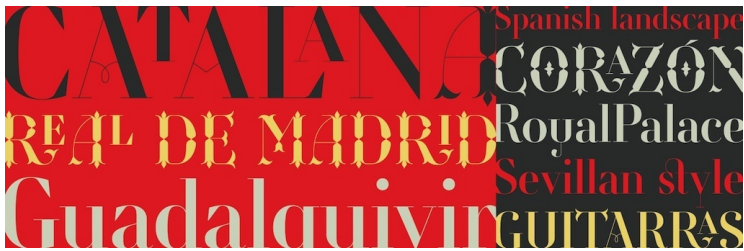
Or calmly reassured:



4-4. www.cleopatra-marina.gr

Fonts also tap into the complex library of associations that we've been accumulating in our brains all of our lives. You build up these associations every time you see a font from the context that you see it in. All of us associate certain letterforms with topics, times and places.

Retiro is obviously Spanish:



4-5. Retiro by Typofonderie

Bodoni and Eurostile used in this menu couldn't be much more Italian:



4-6. Bodoni and Eurostile, both designed in Italy

To me, Clarendon gives a sense of the 1960s and 1970s. I'm not sure if that's what Costa was going for, but that's what it means to me:

The flier is split into two panels. The left panel has a dark red background and features the headline "It begins with the perfect Espresso" in a mix of serif and sans-serif fonts. Below the headline is a paragraph of text and a white cup of espresso with the word "COSTA" on it. The right panel has a lighter red background and features the headline "The rise of the Flat White" in a similar font mix. Below the headline is a paragraph of text and a white cup of flat white with latte art and the word "COSTA" on it.

It begins with the perfect Espresso

It's true. Most of our coffees start with a classic espresso. So even if you're not one for this intense little pick-me-up, it's still thanks to the espresso that your coffee tastes so good.

The rise of the Flat White

A few years ago, no one on this side of the world had heard of the flat white. We've played a part in introducing this antipodean invention to the nation, which has now been wholeheartedly embraced as the true coffee lovers' coffee.

4-7. Costa coffee flier

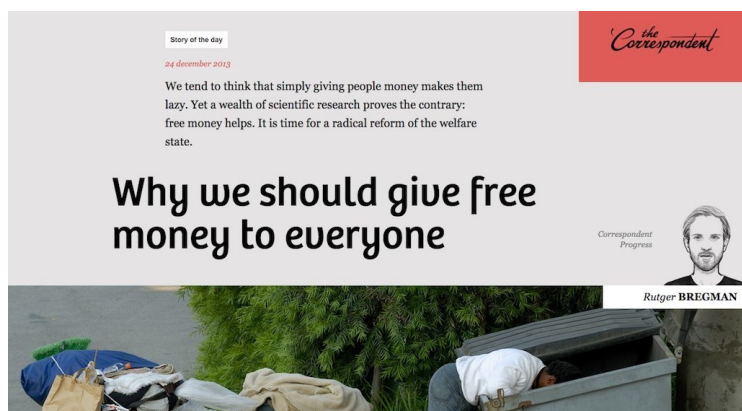
And Knockout and Gotham really couldn't be much more American:

A collage of various USPS shipping service cards. The cards feature different headlines like "U.S. SHIPPING", "INTERNATIONAL SHIPPING", "SELF-SERVICE", and "THANK YOU FOR VISITING". They list various services such as "OVERNIGHT", "1-2 OR 3 DAY SPECIFIC", "FIRST-CLASS", "1-3 BUSINESS DAYS", "3-5 BUSINESS DAYS", "6-10 BUSINESS DAYS", "SHIP PACKAGES", "MAIL LETTERS", "UPGRADE SERVICES", "BUY POSTAGE STAMPS", "RENEW YOUR PO BOX", and "SCAN AND PAY FOR PRODUCTS". Each card includes pricing information and a "TOUCH SCREEN TO BEGIN" button.

4-8. Knockout and Gotham by Hoefler & Co

When it comes to choosing your display typeface, the type designer Christian Schwartz says there are two kinds. First are the workhorse typefaces that will do whatever you want them to do. Helvetica, Proxima Nova and Futura are good examples. These fonts can be shaped in many different ways, but this also means they are found everywhere and take great skill and practice to work with in a unique and striking manner.

The second kind of typeface is one that does most of the work for you. Like finely tailored clothing, it's the detail in the design that adds interest.



4-9. Setting headings in Bree rather than Helvetica makes a big difference to the tone of the article

Such typefaces carry much more inherent character, but are also less malleable and harder to adapt to different contexts. Good examples are Marr Sans, FS Clerkenwell, Strangelove and Bree.

PUSH THE BOAT OUT

Remember, all type can have an effect on the reader. Take advantage of that and allow your type to have its own vernacular and impact. Be expressive with your type. Don't be too reverential, dogmatic – or ordinary. Be brave and push a few boundaries.

Adapted from **Web Typography** a book in progress by Richard Rutter.

ABOUT THE AUTHOR



Richard Rutter is a user experience consultant and director of **Clearleft**. In 2009 he cofounded the webfont service, **Fontdeck**. He runs an ongoing project called **The Elements of Typographic Style Applied to the Web**, where he extols the virtues of good web typography. Richard occasionally blogs at **Clagnut**, where he writes about design, accessibility and web standards issues, as well as his passion for music and mountain biking.

5. Universal React

Jack Franklin

24ways.org/201505

One of the libraries to receive a huge amount of focus in 2015 has been ReactJS, a library created by Facebook for building user interfaces and web applications.

More generally we've seen an even greater rise in the number of applications built primarily on the client side with most of the logic implemented in JavaScript. One of the main issues with building an app in this way is that you immediately forgo any customers who might browse with JavaScript turned off, and you can also miss out on any robots that might visit your site to crawl it (such as Google's search bots). Additionally, we gain a performance improvement by being able to render from the server rather than having to wait for all the JavaScript to be loaded and executed.

The good news is that this problem has been recognised and it is possible to build a fully featured client-side application that can be rendered on the server. The way in which these apps work is as follows:

- The user visits *www.yoursite.com* and the server executes your JavaScript to generate the HTML it needs to render the page.
- In the background, the client-side JavaScript is executed and takes over the duty of rendering the page.
- The next time a user clicks, rather than being sent to the server, the client-side app is in control.
- If the user doesn't have JavaScript enabled, each click on a link goes to the server and they get the server-rendered content again.

This means you can still provide a very quick and snappy experience for JavaScript users without having to abandon your non-JS users. We achieve this by writing JavaScript that can be executed on the server or on the client (you might have heard this referred to as *isomorphic*) and using a JavaScript framework that's clever enough handle server- or client-side execution. Currently, ReactJS is leading the way here, although Ember and Angular are both working on solutions to this problem.

It's worth noting that this tutorial assumes some familiarity with React in general, its syntax and concepts. If you'd like a refresher, the **ReactJS docs** are a good place to start.

GETTING STARTED

We're going to create a tiny ReactJS application that will work on the server and the client. First we'll need to create a new project and install some dependencies. In a new, blank directory, run:

```
npm init -y
npm install --save ejb express react react-router
react-dom
```

That will create a new project and install our dependencies:

- `ejb` is a templating engine that we'll use to render our HTML on the server.
- `express` is a small web framework we'll run our server on.
- `react-router` is a popular routing solution for React so our app can fully support and respect URLs.
- `react-dom` is a small React library used for rendering React components.

We're also going to write all our code in ECMAScript 6, and therefore need to install **BabelJS** and configure that too.

```
npm install --save-dev babel-cli babel-preset-es2015
babel-preset-react
```

Then, create a `.babelrc` file that contains the following:

```
{  
  "presets": ["es2015", "react"]  
}
```

What we've done here is install Babel's command line interface (CLI) tool and configured it to transform our code from ECMAScript 6 (or ES2015) to ECMAScript 5, which is more widely supported. We'll need the React transforms when we start writing **JSX** when working with React.

CREATING A SERVER

For now, our ExpressJS server is pretty straightforward. All we'll do is render a view that says 'Hello World'. Here's our server code:

```
import express from 'express';  
import http from 'http';  
  
const app = express();  
  
app.use(express.static('public'));  
  
app.set('view engine', 'ejs');  
  
app.get('*', (req, res) => {  
  res.render('index');  
});  
  
const server = http.createServer(app);
```

```
server.listen(3003);
server.on('listening', () => {
  console.log('Listening on 3003');
});
```

Here we're using ES6 modules, which I wrote about on [24 ways last year](#), if you'd like a reminder. We tell the app to render the index view on any GET request (that's what `app.get('*')` means, the wildcard matches any route).

We now need to create the index view file, which Express expects to be defined in `views/index.ejs`:

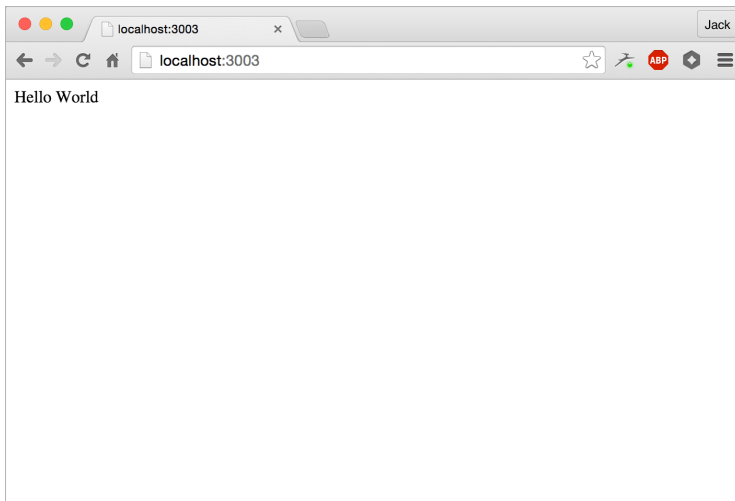
```
<!DOCTYPE html>
<html>
  <head>
    <title>My App</title>
  </head>

  <body>
    Hello World
  </body>
</html>
```

Finally, we're ready to run the server. Because we installed `babel-cli` earlier we have access to the `babel-node` executable, which will transform all your code before running it through node. Run this command:

```
./node_modules/.bin/babel-node server.js
```

And you should now be able to visit <http://localhost:3003> and see 'Hello World' right there:



BUILDING THE REACT APP

Now we'll build the React application entirely on the server, before adding the client-side JavaScript right at the end. Our app will have two routes, `/` and `/about` which will both show a small amount of content. This will demonstrate how to use React Router on the server side to make sure our React app plays nicely with URLs.

Firstly, let's update `views/index.ejs`. Our server will figure out what HTML it needs to render, and pass that into the view. We can pass a value into our view when we render it, and then use EJS syntax to tell it to output that data. Update the template file so the body looks like so:

```
<body>
  <%- markup %>
</body>
```

Next, we'll define the routes we want our app to have using React Router. For now we'll just define the index route, and not worry about the */about* route quite yet. We could define our routes in JSX, but I think for server-side rendering it's clearer to define them as an object. Here's what we're starting with:

```
const routes = {
  path: '',
  component: AppComponent,
  childRoutes: [
    {
      path: '/',
      component: IndexComponent
    }
  ]
}
```

These are just placed at the top of *server.js*, after the import statements. Later we'll move these into a separate file, but for now they are fine where they are.

Notice how I define first that the *AppComponent* should be used at the *''* path, which effectively means it matches every single route and becomes a container for all our other components. Then I give it a child route of */*, which

will match the `IndexComponent`. Before we hook these routes up with our server, let's quickly define *components/app.js* and *components/index.js*. *app.js* looks like so:

```
import React from 'react';

export default class AppComponent extends
React.Component {
  render() {
    return (
      <div>
        <h2>Welcome to my App</h2>
        { this.props.children }
      </div>
    );
  }
}
```

When a React Router route has child components, they are given to us in the props under the `children` key, so we need to include them in the code we want to render for this component. The *index.js* component is pretty bland:

```
import React from 'react';

export default class IndexComponent extends
React.Component {
  render() {
    return (
      <div>
        <p>This is the index page</p>
      </div>
    );
  }
}
```

```
    );
  }
}
```

Server-side routing with React Router

Head back into *server.js*, and firstly we'll need to add some new imports:

```
import React from 'react';
import { renderToString } from 'react-dom/server';
import { match, RoutingContext } from 'react-router';

import AppComponent from './components/app';
import IndexComponent from './components/index';
```

The ReactDOM package provides `react-dom/server` which includes a `renderToString` method that takes a React component and produces the HTML string output of the component. It's this method that we'll use to render the HTML from the server, generated by React. From the React Router package we use `match`, a function used to find a matching route for a URL; and `RoutingContext`, a React component provided by React Router that we'll need to render. This wraps up our components and provides some functionality that ties React Router together with our app. Generally you don't need to concern yourself about how this component works, so don't worry too much.

Now for the good bit: we can update our `app.get('*')` route with the code that matches the URL against the React routes:

```
app.get('*', (req, res) => {
  // routes is our object of React routes defined above
  match({ routes, location: req.url }, (err,
  redirectLocation, props) => {
    if (err) {
      // something went badly wrong, so 500 with a
      message
      res.status(500).send(err.message);
    } else if (redirectLocation) {
      // we matched a ReactRouter redirect, so redirect
      from the server
      res.redirect(302, redirectLocation.pathname +
      redirectLocation.search);
    } else if (props) {
      // if we got props, that means we found a valid
      component to render
      // for the given route
      const markup = renderToString(<RoutingContext
      {...props} />);

      // render `index.ejs`, but pass in the markup we
      want it to display
      res.render('index', { markup })
    } else {
      // no route match, so 404. In a real app you might
      render a custom
      // 404 view here
      res.sendStatus(404);
    }
  })
})
```

```

    }
  });
});

```

We call `match`, giving it the `routes` object we defined earlier and `req.url`, which contains the URL of the request. It calls a callback function we give it, with `err`, `redirectLocation` and `props` as the arguments. The first two conditionals in the callback function just deal with an error occurring or a redirect (React Router has built in redirect support). The most interesting bit is the third conditional, `else if (props)`. If we got given props and we've made it this far it means we found a matching component to render and we can use this code to render it:

```

...
} else if (props) {
  // if we got props, that means we found a valid
  component to render
  // for the given route
  const markup = renderToString(<RoutingContext
{...props} />);

  // render `index.ejs`, but pass in the markup we want
  it to display
  res.render('index', { markup })
} else {
  ...
}

```

The `renderToString` method from `ReactDOM` takes that `RoutingContext` component we mentioned earlier and renders it with the properties required. Again, you need not concern yourself with what this specific component does or what the props are. Most of this is data that `React Router` provides for us on top of our components.

Note the `{...props}`, which is a neat bit of `JSX` syntax that spreads out our object into key value properties. To see this better, note the two pieces of `JSX` code below, both of which are equivalent:

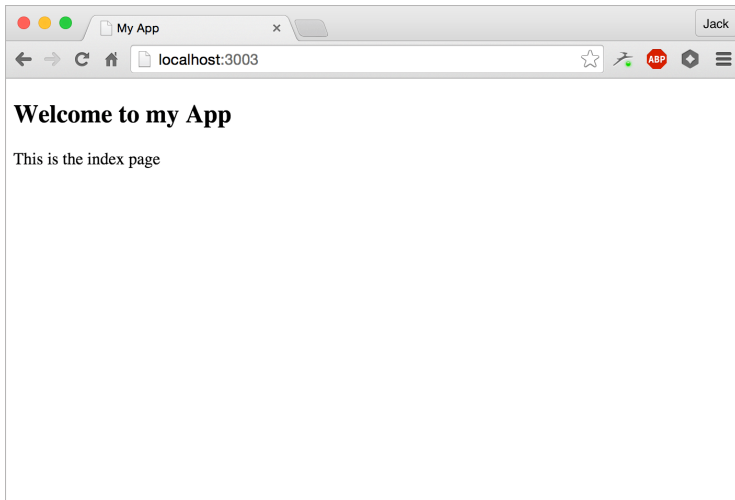
```
<MyComponent a="foo" b="bar" />
```

```
// OR:
```

```
const props = { a: "foo", b: "bar" };  
<MyComponent {...props} />
```

Running the server again

I know that felt like a lot of work, but the good news is that once you've set this up you are free to focus on building your `React` components, safe in the knowledge that your server-side rendering is working. To check, restart the server and head to <http://localhost:3003> once more. You should see it all working!



REFACTORING AND ONE MORE ROUTE

Before we move on to getting this code running on the client, let's add one more route and do some tidying up. First, move our routes object out into *routes.js*:

```
import AppComponent from './components/app';
import IndexComponent from './components/index';

const routes = {
  path: '',
  component: AppComponent,
  childRoutes: [
    {
      path: '/',
      component: IndexComponent
    }
  ]
}
```

```
}
```

```
export { routes };
```

And then update *server.js*. You can remove the two component imports and replace them with:

```
import { routes } from './routes';
```

Finally, let's add one more route for *./about* and links between them. Create *components/about.js*:

```
import React from 'react';
```

```
export default class AboutComponent extends
React.Component {
  render() {
    return (
      <div>
        <p>A little bit about me.</p>
      </div>
    );
  }
}
```

And then you can add it to *routes.js* too:

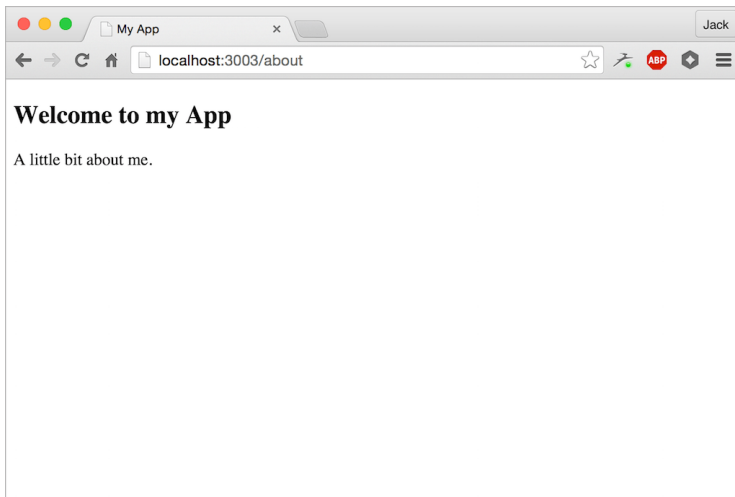
```
import AppComponent from './components/app';
import IndexComponent from './components/index';
import AboutComponent from './components/about';

const routes = {
  path: '',
  component: AppComponent,
```

```
childRoutes: [  
  {  
    path: '/',  
    component: IndexComponent  
  },  
  {  
    path: '/about',  
    component: AboutComponent  
  }  
]  
}
```

```
export { routes };
```

If you now restart the server and head to *<http://localhost:3003/about>* you'll see the about page!



For the finishing touch we'll use the React Router link component to add some links between the pages. Edit *components/app.js* to look like so:

```
import React from 'react';
import { Link } from 'react-router';

export default class AppComponent extends
React.Component {
  render() {
    return (
      <div>
        <h2>Welcome to my App</h2>
        <ul>
          <li><Link to='/'>Home</Link></li>
          <li><Link to='/about'>About</Link></li>
        </ul>
        { this.props.children }
      </div>
    );
  }
}
```

You can now click between the pages to navigate. However, everytime we do so the requests hit the server. Now we're going to make our final change, such that after the app has been rendered on the server once, it gets rendered and managed in the client, providing that snappy client-side app experience.

CLIENT-SIDE RENDERING

First, we're going to make a small change to *views/index.ejs*. React doesn't like rendering directly into the body and will give a warning when you do so. To prevent this we'll wrap our app in a div:

```
<body>
  <div id="app"><%- markup %></div>
  <script src="build.js"></script>
</body>
```

I've also added in a script tag to *build.js*, which is the file we'll generate containing all our client-side code.

Next, create *client-render.js*. This is going to be the only bit of JavaScript that's exclusive to the client side. In it we need to pull in our routes and render them to the DOM.

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Router } from 'react-router';

import { routes } from './routes';

import createBrowserHistory from 'history/lib/
createBrowserHistory';

ReactDOM.render(
  <Router routes={routes}
  history={createBrowserHistory()} />,
  document.getElementById('app')
)
```

The first thing you might notice is the mention of `createBrowserHistory`. React Router is built on top of the history module, a module that listens to the browser's address bar and parses the new location. It has many modes of operation: it can keep track using a hashbang, such as `http://localhost/#!/about` (this is the default), or you can tell it to use the HTML5 history API by calling `createBrowserHistory`, which is what we've done. This will keep the URLs nice and neat and make sure the client and the server are using the same URL structure. You can read more about **React Router and histories** in the React Router documentation.

Finally we use `ReactDOM.render` and give it the Router component, telling it about all our routes, and also tell ReactDOM where to render, the `#app` element.

Generating build.js

We're actually almost there! The final thing we need to do is generate our client side bundle. For this we're going to use **webpack**, a module bundler that can take our application, follow all the imports and generate one large bundle from them. We'll install it and `babel-loader`, a webpack plugin for transforming code through Babel.

```
npm install --save-dev webpack babel-loader
```

To run webpack we just need to create a configuration file, called *webpack.config.js*. Create the file in the root of our application and add the following code:

```
var path = require('path');
module.exports = {
  entry: path.join(process.cwd(), 'client-render.js'),
  output: {
    path: './public/',
    filename: 'build.js'
  },
  module: {
    loaders: [
      {
        test: /\.js$/,
        loader: 'babel'
      }
    ]
  }
}
```

Note first that this file can't be written in ES6 as it doesn't get transformed. The first thing we do is tell webpack the main entry point for our application, which is *client-render.js*. We use `process.cwd()` because webpack expects an exact location – if we just gave it the string 'client-render.js', webpack wouldn't be able to find it.

Next, we tell webpack where to output our file, and here I'm telling it to place the file in *public/build.js*. Finally we tell webpack that every time it hits a file that ends in *.js*, it should use the *babel-loader* plugin to transform the code first.

Now we're ready to generate the bundle!

```
./node_modules/.bin/webpack
```

This will take a fair few seconds to run (on my machine it's about seven or eight), but once it has it will have created *public/build.js*, a client-side bundle of our application. If you restart your server once more you'll see that we can now navigate around our application without hitting the server, because React on the client takes over. Perfect!

The first bundle that webpack generates is pretty slow, but if you run `webpack -w` it will go into watch mode, where it watches files for changes and regenerates the bundle. The key thing is that it only regenerates the small pieces of the bundle it needs, so while the first bundle is very slow, the rest are lightning fast. I recommend leaving webpack constantly running in watch mode when you're developing.

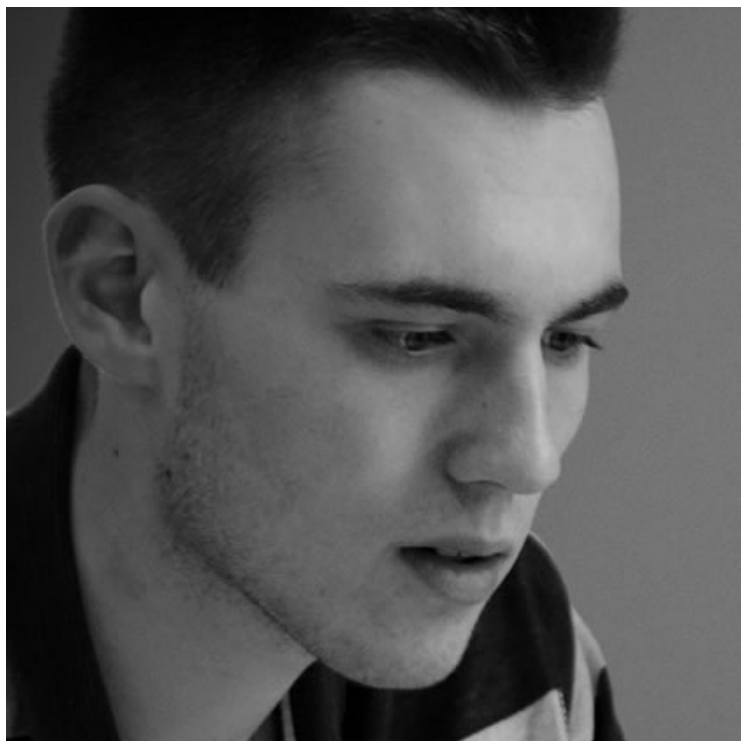
CONCLUSIONS

First, if you'd like to look through this code yourself **you can find it all on GitHub**. Feel free to raise an issue there or **tweet me** if you have any problems or would like to ask further questions.

Next, I want to stress that you shouldn't use this as an excuse to build all your apps in this way. Some of you might be wondering whether a static site like the one we built today is worth its complexity, and you'd be right. I used it as it's an easy example to work with but in the future you should carefully consider your reasons for wanting to build a universal React application and make sure it's a suitable infrastructure for you.

With that, all that's left for me to do is wish you a very merry Christmas and best of luck with your React applications!

ABOUT THE AUTHOR



Jack Franklin is a developer, speaker and author who blogs at javascriptplayground.com and has authored “Beginning jQuery”. Jack works as a developer evangelist for Pusher and is also a Google Developer Expert for the Chrome HTML 5 platform. He tweets as [@jack_franklin](https://twitter.com/jack_franklin) and spends far too much time thinking about JavaScript.

6. Bringing Your Code to the Streets

Ruth John

— or

24ways.org/201506

How to Be a Street VJ

Our amazing world of web code is escaping out of the browser at an alarming rate and appearing in every aspect of the environment around us. Over the past few years we've already seen JavaScript used server-side, hardware coded with JavaScript, a rise of native style and desktop apps created with HTML, CSS and JavaScript, and even virtual reality (VR) is getting its fair share of front-end goodness.

You can go ahead and play with JavaScript-powered hardware such as the Tessel or the Espruino to name a couple. Just check out the [Tessel project page](#) to see JavaScript in the world of coffee roasting or sleep tracking your pet. With the rise of the internet of things, JavaScript can be seen collecting information on flooding

among other things. And if that's not enough 'outside the browser' implementations, Node.js servers can even be found in aircraft!

I previously mentioned VR and with `three.js`'s extra `StereoEffect.js` module it's relatively simple to get browser 3D goodness to be Google Cardboard-ready, and thus set the stage for all things JavaScript and VR. It's been pretty popular in the art world too, with interactive works such as Seb Lee-Delisle's Lunar Trails installation, featuring the old arcade game Lunar Lander, which you can now play in your browser while others watch (it is the web after all). The Science Museum in London held **Chrome Web Lab**, an interactive exhibition featuring five *experiments*, showcasing the magic of the web. And it's not even the connectivity of the web that's being showcased; we can even take things offline and use web code for amazing things, such as fighting Ebola.

One thing is for sure, JavaScript is awesome. Hell, if you believe those telly programs (as we all do), JavaScript can even take down the stock market, purely through the witchcraft of canvas! Go JavaScript!

NOW IT'S OUR TURN

So I wanted to create a little project influenced by this theme, and as it's Christmas, take it to the streets for a little bit of party fun! Something that could take code

anywhere. Here's how I made a portable visual projection pack, a piece of video mixing software and created some web-coded street art.

Step one: The equipment

You will need:

- **One laptop:** with HDMI output and a modern browser installed, such as Google Chrome.
- **One battery-powered mini projector:** I've used a Texas Instruments DLP; for its 120 lumens it was the best cost-to-lumens ratio I could find.
- **One MIDI controller (optional):** mine is an ICON iDJ as it suits mixing visuals. However, there is more affordable hardware on the market such as an Akai LPD8 or a Korg nanoPAD2. As you'll see in the article, this is optional as it can be emulated within the software.
- A case to carry it all around in.



Step two: The software

The projected visuals, I imagined, could be anything you can create within a browser, whether that be simple HTML and CSS, images, videos, SVG or canvas. The only requirement I have is that they move or change with sound and that I can mix any one visual into another.

You may remember a couple of years ago I created a **demo on this very site**, allowing audio-triggered visuals from the ambient sounds your device mic was picking up. That was a great starting point – I used that exact method to pick up the audio and thus the first requirement was complete. If you want to see some more examples of visuals I've put together for this, there's a **showcase on CodePen**.

The second requirement took a little more thought. I needed two *screens*, which could at any point show any of the visuals I had coded, but could be mixed from one into the other and back again. So let's start with two `div`s, both absolutely positioned so they're on top of each other, but at the start the second screen's `opacity` is set to zero.

Now all we need is a slider, which when moved from one side to the other slowly sets the second screen's `opacity` to 1, thereby fading it in.

See the [Pen Mixing Screens \(Software Version\)](#) by Rummyra (@Rummyra) on CodePen.

6-1. Mixing Screens (CodePen)

As you saw above, I have a MIDI controller and although the software method works great, I'd quite like to make use of this nifty piece of kit. That's easily done with the Web MIDI API. All I need to do is call it, and when I move one of the sliders on the controller (I've allocated the big cross fader in the middle for this), pick up on the change of value and use that to control the `opacity` instead.

```
var midi, data;
// start talking to MIDI controller
if (navigator.requestMIDIAccess) {
  navigator.requestMIDIAccess({
    sysex: false
  }).then(onMIDISuccess, onMIDIFailure);
```

```

} else {
    alert("No MIDI support in your browser.");
}

// on success
function onMIDISuccess(midiData) {
    // this is all our MIDI data
    midi = midiData;

    var allInputs = midi.allInputs.values();
    // loop over all available inputs and listen for any
    MIDI input
    for (var input = allInputs.next(); input &&
    !input.done; input = allInputs.next()) {
        // when a MIDI value is received call the
        onMIDIMessage function
        input.value.onmidimessage = onMIDIMessage;
    }
}

function onMIDIMessage(message) {
    // data comes in the form [command/channel, note,
    velocity]
    data = message.data;

    // Opacity change for screen. The cross fader values
    are [176, 8, {0-127}]
    if ( (data[0] === 176) && (data[1] === 8) ) {
        // this value will change as the fader is moved
        var opacity = data[2]/127;
        screenTwo.style.opacity = opacity;
    }
}

```

The final code was slightly more complicated than this, as I decided to switch the two screens based on the frequencies of the sound that was playing, and use the cross fader to depict the frequency threshold value. This meant they flickered in and out of each other, rather than just faded. There's a very rough-and-ready first version of the software on [GitHub](#).

Phew, Great! Now we need to get all this to the streets!

Step three: Portable kit

Did you notice how I mentioned a case to carry it all around in? I wanted the case to be morphable, so I could use the equipment from it too, a sort of bag-to-usherette-tray-type affair. Well, I had an unused laptop bag...



I strengthened it with some MDF, so when I opened the bag it would hold like a tray where the laptop and MIDI controller would sit. The projector was Velcroed to the external pocket of the bag, so when it was a tray it would project from underneath. I added two durable straps, one for my shoulders and one round my waist, both attached to the bag itself. There was a lot of cutting and trimming. As it was a laptop bag it was pretty thick to start and sewing was tricky. However, I only broke one sewing machine needle; I've been known to break more working with leather, so I figured I was doing well. By the way, you can actually *buy* usherette trays, but I just couldn't resist hacking my own :)

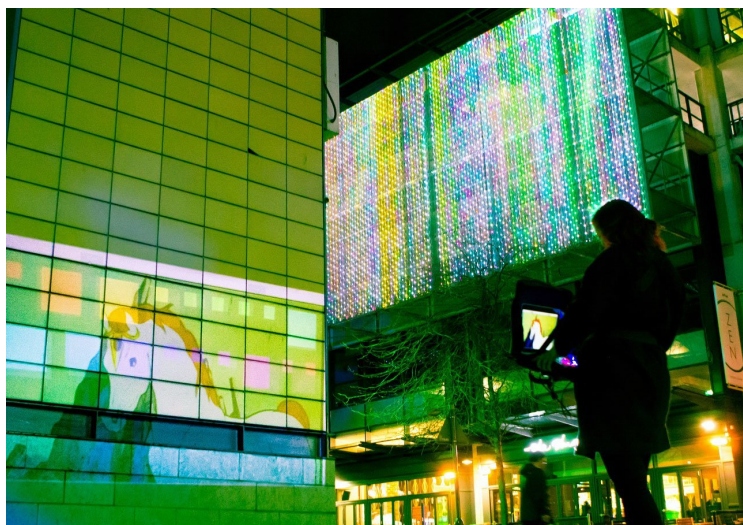
Step four: Take to the streets

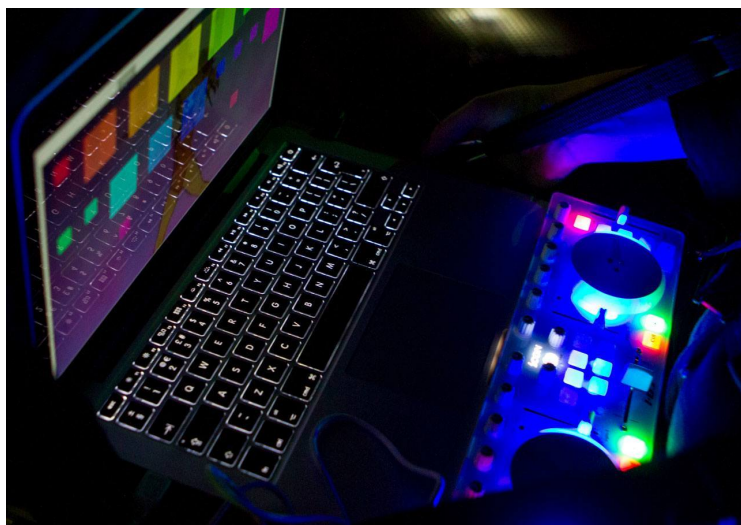
First, make sure everything is charged – everything – a lot! The laptop has to power both the MIDI controller and the projector, and although I have a mobile phone battery booster pack, that'll only charge the projector should it run out. I estimated I could get a good hour of visual artistry before I needed to worry, though.

I had a couple of ideas about time of day and location. Here in the UK at this time of year, it gets dark around half past four, so I could easily head out in a city around 5pm and it would be dark enough for the projections to be seen pretty well. I chose Bristol, around the waterfront, as

there were some interesting locations to try it out in. The best was Millennium Square: busy but not crowded and plenty of surfaces to try projecting on to.

My first time out with the portable audio/visual pack (PAVP as it will now be named) was brilliant. I played music *and* projected visuals, like a one-woman band of A/V!





You might be thinking what the point of this was, besides, of course, it being a bit of fun. Well, this project got me to look at canvas and SVG more closely. The Web MIDI API was really interesting; MIDI as a data format has some great practical uses. I think without our side projects we may not have all these wonderful uses for our everyday code. Not only do they remind us coding can, and should, be fun, they also help us learn and grow as makers.

My favourite part? When I was projecting into a water feature in Millennium Square. For those who are familiar, you'll know it's like a wall of water so it produced a superb effect. I drew quite a crowd and a kid came to stand next to me and all I could hear him say with enthusiasm was, 'Oh wow! That's so cool!'

Yes... yes, kid, it was cool. Making things with code is cool.

Massive thanks to the lovely Drew McLellan for his incredibly well-directed photography, and also Simon Johnson who took a great hand in perfecting the kit while it was attached.

ABOUT THE AUTHOR



Ruth John wireframes, designs and codes for **The Lab** at O2 (Telefonica). She also **tweets** and **blogs** a bit too. You can often find her chatting about web development, building apps and how an extra div is not the answer to your styling problems. Either that or the lesser known Thundercats characters.

7. Git Rebasing: An Elfin Workshop Workflow

Emma Jane Westby

24ways.org/201507

This year Santa's helpers have been tasked with making a garland. It's a pretty simple task: string beads onto yarn in a specific order. When the garland reaches a specific length, add it to the main workshop garland. Each elf has a specific sequence they're supposed to chain, which is given to them via a work order. (This is starting to sound like one of those horrible calculus problems. I promise it isn't. It's worse; it's about Git.)

For the most part, the system works really well. The elves are able to quickly build up a shared chain because each elf specialises on their own bit of garland, and then links the garland together. Because of this they're able to work independently, but towards the common goal of making a beautiful garland.

At first the elves are really careful with each bead they put onto the garland. They check with one another before merging their work, and review each new link carefully. As time crunches on, the elves pour a little more cheer into the eggnog cooler, and the quality of work starts to degrade. Tensions rise as mistakes are made and unkind words are said. The elves quickly realise they're going to need a system to change the beads out when mistakes are made in the chain.

The first common mistake is not looking to see what the latest chain is that's been added to the main garland. The garland is huge, and it sits on a roll in one of the corners of the workshop. It's a big workshop, so it is incredibly impractical to walk all the way to the roll to check what the last link is on the chain. The elves, being magical, have set up a monitoring system that allows them to keep a local copy of the main garland at their workstation. It's an imperfect system though, so the elves have to request a manual refresh to see the latest copy. They can request a new copy by running the command

```
git pull --rebase=preserve
```

(They found that if they ran `git pull` on its own, they ended up with weird loops of extra beads off the main garland, so they've opted to use this method.) This keeps the shared garland up to date, which makes things a lot easier. A visualisation of the rebase process is available.

The next thing the elves noticed is that if they worked on the main workshop garland, they were always running into problems when they tried to share their work back with the rest of the workshop. It was fine if they were working late at night by themselves, but in the middle of the day, it was horrible. (I've been asked not to talk about that time the fight broke out.) Instead of trying to share everything on their local copy of the main garland, the elves have realised it's a lot easier to work on a new string and then knot this onto the main garland when their pattern repeat is finished. They generate a new string by issuing the following commands:

```
git checkout master
git checkout -b 1234_pattern-name
```

1234 represents the work order number and pattern-name describes the pattern they're adding. Each bead is then added to the new link (`git add bead.txt`) and locked into place (`git commit`). Each elf repeats this process until the sequence of beads described in the work order has been added to their mini garland.

To combine their work with the main garland, the elves need to make a few decisions. If they're making a single strand, they issue the following commands:

```
git checkout master
git merge --ff-only 1234_pattern-name
```

To share their work they publish the new version of the main garland to the workshop spool with the command `git push origin master`.

Sometimes this fails. Sharing work fails because the workshop spool has gotten new links added since the elf last updated their copy of the main workshop spool. This makes the elves both happy and sad. It makes them happy because it means the other elves have been working too, but it makes them sad because they now need to do a bit of extra work to close their work order.

To update the local copy of the workshop spool, the elf first unlinks the chain they just linked by running the command:

```
git reset --merge ORIG_HEAD
```

This works because the garland magic notices when the elves are doing a particularly dangerous thing and places a temporary, invisible bookmark to the last safe bead in the chain before the dangerous thing happened. The garland no longer has the elf's work, and can be updated safely. The elf runs the command `git pull --rebase=preserve` and the changes all the other elves have made are applied locally.

With these new beads in place, the elf now has to restring their own chain so that it starts at the right place. To do this, the elf turns back to their own chain (`git checkout`

1234_pattern-name) and runs the command `git rebase master`. Assuming their bead pattern is completely unique, the process will run and the elf's beads will be restrung on the tip of the main workshop garland.

Sometimes the magic fails and the elf has to deal with merge conflicts. These are kind of annoying, so the elf uses a special inspector tool to figure things out. The elf opens the inspector by running the command `git mergetool` to work through places where their beads have been added at the same points as another elf's beads. Once all the conflicts are resolved, the elf saves their work, and quits the inspector. They might need to do this a few times if there are a lot of new beads, so the elf has learned to follow this update process regularly instead of just waiting until they're ready to close out their work order.

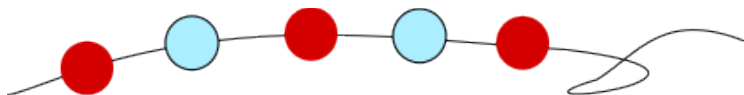
Once their link is up to date, the elf can now reapply their chain as before, publish their work to the main workshop garland, and close their work order:

```
git checkout master
git merge --ff-only 1234_pattern-name
git push origin master
```

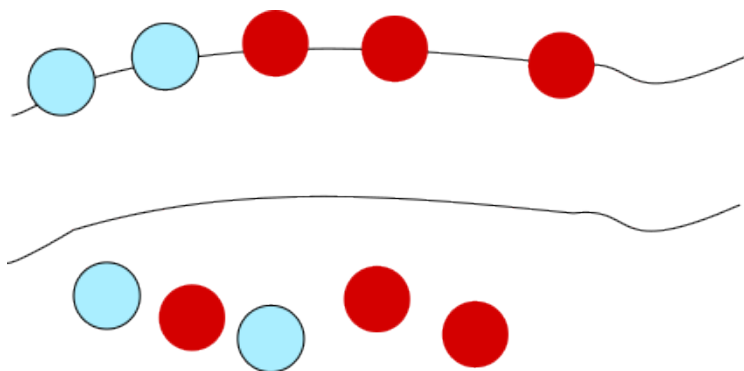
Generally this process works well for the elves. Sometimes, though, when they're tired or bored or a little drunk on festive cheer, they realise there's a mistake in their chain of beads. Fortunately they can fix the beads

without anyone else knowing. These tools can be applied to the whole workshop chain as well, but it causes problems because the magic assumes that elves are only ever adding to the main chain, not removing or reordering beads on the fly. Depending on where the mistake is, the elf has a few different options.

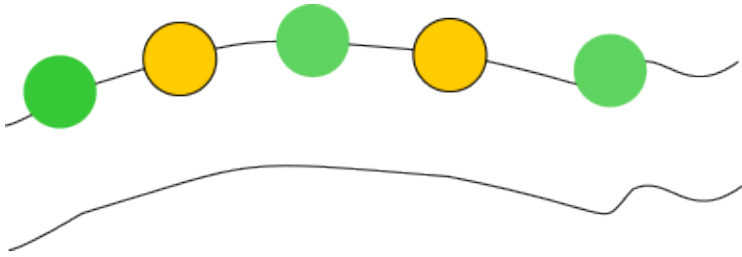
Let's pretend the elf has a sequence of five beads she's been working on. The work order says the pattern should be red-blue-red-blue-red.



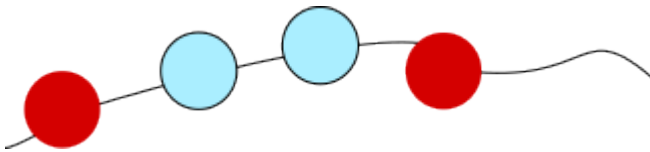
If the sequence of beads is wrong (for example, blue-blue-red-red-red), the elf can remove the beads from the chain, but keep the beads in her workstation using the command `git reset --soft HEAD~5`.



If she's been using the wrong colours and the wrong pattern (for example, green-green-yellow-green), she can remove the beads from her chain and discard them from her workstation using the command `git reset --hard HEAD~5`.



If one of the beads is missing (for example, red-blue-blue-red), she can restring the beads using the first method, or she can use a bit of magic to add the missing bead into the sequence.



Using a tool that's a bit like orthoscopic surgery, she first selects a sequence of beads which contains the problem. A visualisation of this process is available.

Start the garland surgery process with the command:

```
git rebase --interactive HEAD~4
```

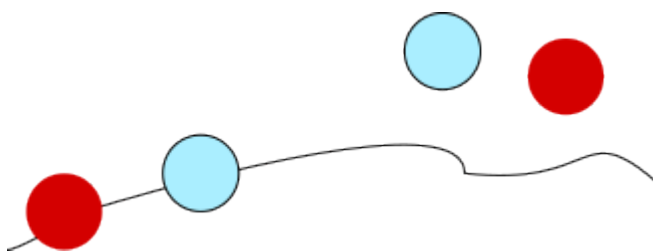
A new screen comes up with the following information
(the oldest bead is on top):

```
pick c2e4877 Red bead
pick 9b5555e Blue bead
pick 7afd66b Blue bead
pick elf2537 Red bead
```

The elf adjusts the list, changing “pick” to “edit” next to
the first blue bead:

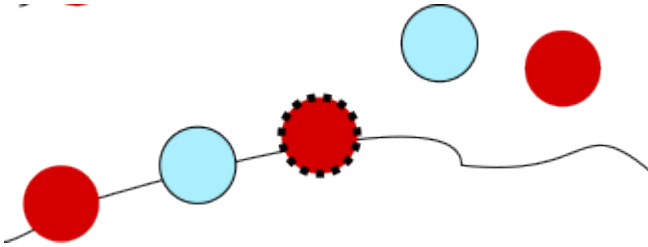
```
pick c2e4877 Red bead
edit 9b5555e Blue bead
pick 7afd66b Blue bead
pick elf2537 Red bead
```

She then saves her work and quits the editor. The garland
magic has placed her back in time at the moment just after
she added the first blue bead.



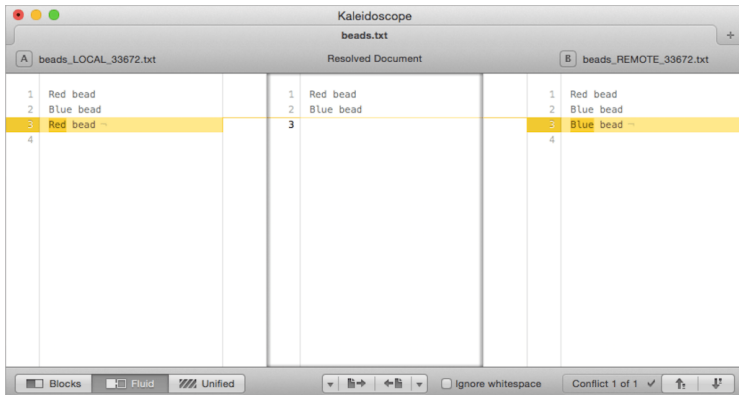
She needs to manually fix up her garland to add the new
red bead. If the beads were files, she might run commands
like `vim beads.txt` and edit the file to make the necessary
changes.

Once she's finished her changes, she needs to add her new bead to the garland (`git add --all`) and lock it into place (`git commit`). This time she assigns the commit message "Red bead – added" so she can easily find it.

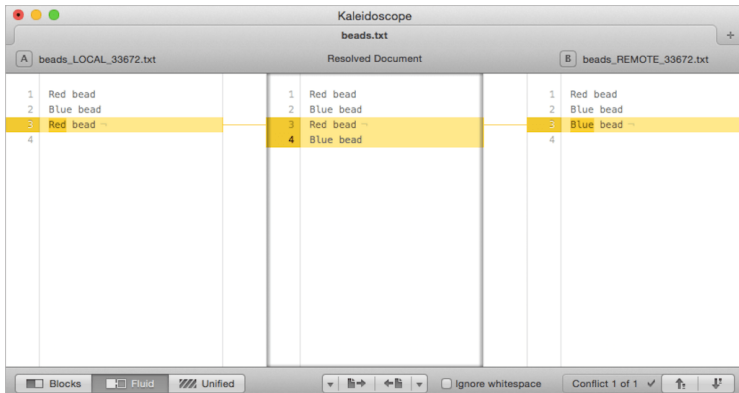


The garland magic has replaced the bead, but she still needs to verify the remaining beads on the garland. This is a mostly automatic process which is started by running the command `git rebase --continue`.

The new red bead has been assigned a position formerly held by the blue bead, and so the elf must deal with a merge conflict. She opens up a new program to help resolve the conflict by running `git mergetool`.



She knows she wants both of these beads in place, so the elf edits the file to include both the red and blue beads.



With the conflict resolved, the elf saves her changes and quits the mergetool.

Back at the command line, the elf checks the status of her work using the command `git status`.

Git Rebasing: An Elfin Workshop Workflow

rebase in progress; onto 4a9cb9d

You are currently rebasing branch '2_RBRBR' on '4a9cb9d'.

(all conflicts fixed: run "git rebase --continue")

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: beads.txt

Untracked files:

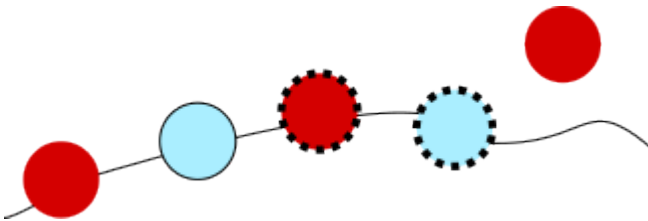
(use "git add <file>..." to include in what will be committed)

beads.txt.orig

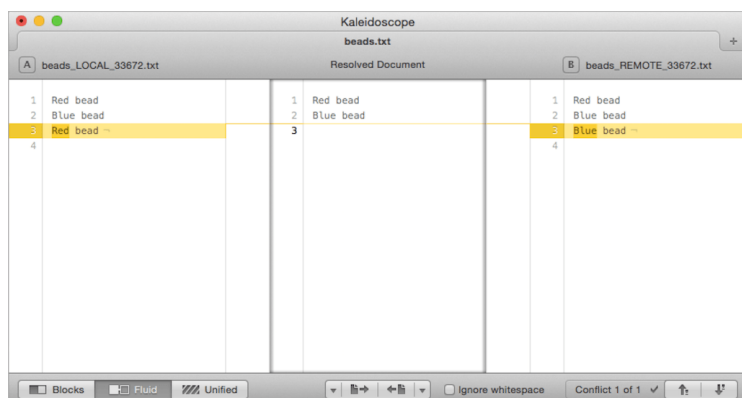
She removes the file added by the mergetool with the command `rm beads.txt.orig` and commits the edits she just made to the bead file using the commands:

```
git add beads.txt
```

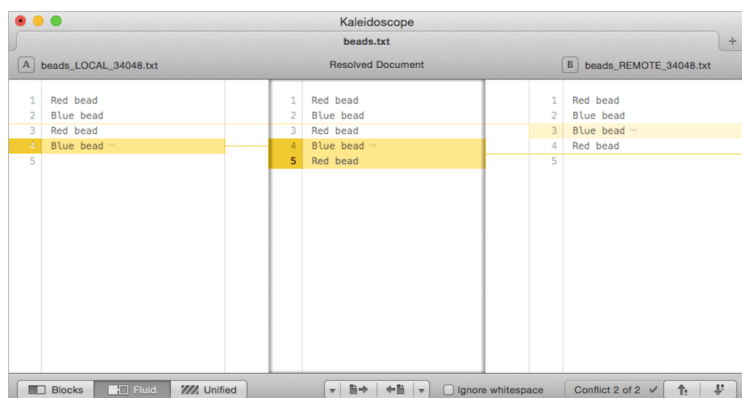
```
git commit --message "Blue bead -- resolved conflict"
```



With the conflict resolved, the elf is able to continue with the rebasing process using the command `git rebase --continue`. There is one final conflict the elf needs to resolve. Once again, she opens up the visualisation tool and takes a look at the two conflicting files.



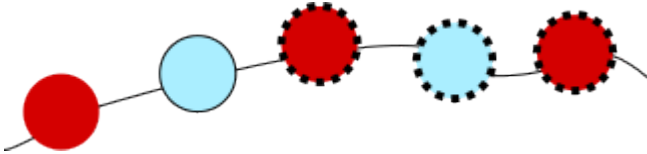
She incorporates the changes from the left and right column to ensure her bead sequence is correct.



Once the merge conflict is resolved, the elf saves the file and quits the mergetool. Once again, she cleans out the backup file added by the mergetool (`rm beads.txt.orig`) and commits her changes to the garland:

```
git add beads.txt
git commit --message "Red bead -- resolved conflict"
```

and then runs the final verification steps in the rebase process (`git rebase --continue`).



The verification process runs through to the end, and the elf checks her work using the command `git log --oneline`.

```
9269914 Red bead -- resolved conflict
4916353 Blue bead -- resolved conflict
aef0d5c Red bead -- added
9b5555e Blue bead
c2e4877 Red bead
```

She knows she needs to read the sequence from bottom to top (the oldest bead is on the bottom). Reviewing the list she sees that the sequence is now correct.

Sometimes, late at night, the elf makes new copies of the workshop garland so she can play around with the bead sequencer just to see what happens. It's made her more confident at restringing beads when she's found real mistakes. And she doesn't mind helping her fellow elves when they run into trouble with their beads. The sugar cookies they leave her as thanks don't hurt either. If you would also like to play with the bead sequencer, you can get a copy of the branches the elf worked.



Our lessons from the workshop:

- By using `rebase` to update your branches, you avoid merge commits and keep a clean commit history.
- If you make a mistake on one of your local branches, you can use `reset` to take commits off your branch. If you want to save the work, but uncommit it, add the parameter `--soft`. If you want to completely discard the work, use the parameter, `--hard`.
- If you have merged working branch changes to the local copy of your master branch and it is preventing you from pushing your work to a remote repository, remove these changes using the command `reset` with the parameter `--merge ORIG_HEAD` before updating your local copy of the remote master branch.

- If you want to make a change to work that was committed a little while ago, you can use the command `rebase` with the parameter `--interactive`. You will need to include how many commits back in time you want to review.

ABOUT THE AUTHOR



Emma Jane Westby is an author, an educator, and a part-time beekeeper. Her latest book, **Git for Teams**, is now available from O'Reilly. You can follow her adventures on Twitter at [@emmajanehw](https://twitter.com/emmajanehw).

8. Helping VIPs Care About Performance

Lara Hogan

24ways.org/201508

Making a site feel super fast is the easy part of performance work. Getting people around you to care about site speed is a much bigger challenge. How do we keep the site fast beyond the initial performance work? Keeping very important people like your upper management or clients invested in performance work is critical to keeping a site fast and empowering other designers and developers to contribute.

The work to get others to care is so meaty that I dedicated a whole chapter to the topic in my book *Designing for Performance*. When I speak at conferences, the majority of questions during Q&A are on this topic. When I speak to developers and designers who care about performance, getting other people at one's organization or agency to care becomes the most pressing question.

My primary response to folks who raise this issue is the question: “**What metric(s) do your VIPs care about?**” This is often met with blank stares and raised eyebrows. But it’s also our biggest clue to what we need to do to help empower others to care about performance and work on it. Every organization and executive is different. This means that three major things vary: the primary metrics VIPs care about; the language they use about measuring success; and how change is enacted. By clueing in to these nuances within your organization, you can get a huge leg up on crafting a successful pitch about performance work.

Let’s start with the metric that we should measure. Sure, (most) everybody cares about money - but is that really the metric that your VIPs are looking at each day to measure the success or efficacy of your site? More likely, dollars are the end game, but the metrics or key performance indicators (KPIs) people focus on might be:

- rate of new accounts created/signups
- cost of acquiring or retaining a customer
- visitor return rate
- visitor bounce rate
- favoriting or another interaction rate

These are just a few examples, but they illustrate how wide-ranging the options are that people care about. I find that developers and designers haven’t necessarily investigated this when trying to get others to care about

performance. We often reach for the obvious – money! – but if we don't use the same kind of language our VIPs are using, we might not get too far. You need to know this before you can make the case for performance work.

To find out these metrics or KPIs, start reading through the emails your VIPs are sending within your company. What does it say on company wikis? Are there major dashboards internally that people are looking at where you could find some good metrics? Listen intently in team meetings or thoroughly read annual reports to see what these metrics could be.

The second key here is to pick up on language you can effectively copy and paste as you make the case for performance work. You need to be able to reflect back the metrics that people already find important in a way they'll be able to hear. Once you know your key metrics, it's time to figure out how to communicate with your VIPs about performance using language that will resonate with them.

Let's start with *visit traffic* as an example metric that a very important person cares about. Start to dig up research that other people and companies have done that correlates performance and your KPI. For example, cite studies:

“When the home page of Google Maps was reduced from 100KB to 70–80KB, traffic went up 10% in the first week, and an additional 25% in the following three weeks.” (source).

Read through websites like **WPOStats**, which collects the spectrum of studies on the impact of performance optimization on user experience and business metrics. Tweet and see if others have done similar research that correlates performance and your site’s main KPI.

Once you have collected some research that touches on the same kind of language your VIPs use about the success of your site, it’s time to present it. You can start with something simple, like a qualitative description of the work you’re actively doing to improve the site that translates to improved metrics that your VIPs care about. It can be helpful to append a **performance budget** to any proposal so you can compare the budget to your site’s reality and how it might positively impact those KPIs folks care about.

Words and graphs are often only half the battle when it comes to getting others to care about performance. Often, videos appeal to folks’ emotions in a way that is missed when glancing through charts and graphs. On **A List Apart** I recently detailed how to create videos of how

fast your site loads. Let's say that your VIPs care about how your site loads on mobile devices; it's time to show them how your site loads on mobile networks.

Open video

You can use these videos to make a number of different statements to your VIPs, depending on what they care about:

- Look at how slow our site loads versus our competitor!
- Look at how slow our site loads for users in another country!
- Look at how slow our site loads on mobile networks!

Again, you really need to know which metrics your VIPs care about and tune into the language they're using. If they don't care about the overall user experience of your site on mobile devices, then showing them how slow your site loads on 3G isn't going to work. This will be your sales pitch; you need to practice and iterate on the language and highlights that will land best with your audience.

To make your sales pitch as solid as possible, gut-check your ideas on how to present it with other co-workers to get their feedback. Read up on how to construct effective arguments and deliver them; do some research and see what others have done at your company when pitching to VIPs. Are slides effective? Memos or emails? Hallway

conversations? Sometimes the best way to change people's minds is by mentioning it in informal chats over coffee. Emulate the other leaders in your organization who are successful at this work.

Every organization and very important person is different. Learn what metrics folks truly care about, study the language that they use, and apply what you've learned in a way that'll land with those individuals. It may take time to craft your pitch for performance work over time, but it's important work to do. If you're able to figure out how to mirror back the language and metrics VIPs care about, and connect the dots to performance for them, you will have a huge leg up on keeping your site fast in the long run.

ABOUT THE AUTHOR



Lara Hogan is a Senior Engineering Manager at Etsy and the author of *Designing for Performance* and *Building a device lab*. She champions performance as a part of the overall user experience, striking a balance between aesthetics and speed, and building performance into company culture. She also believes it's important to celebrate career achievements with donuts.

9. Animation in Responsive Design

Val Head

24ways.org/201509

Animation and responsive design can sometimes feel like they're at odds with each other. Animation often needs space to do its thing, but RWD tells us that the amount of space we'll have available is going to change a lot. Balancing that can lead to some tricky animation situations.

Embracing the squishiness of responsive design doesn't have to mean giving up on your creative animation ideas. There are three general techniques that can help you balance your web animation creativity with your responsive design needs. One or all of these approaches might help you sneak in something just a little extra into your next project.

FOCUSED ART DIRECTION

Smaller viewports mean a smaller stage for your motion to play out on, and this tends to amplify any motion in your animation. Suddenly 100 pixels is *really* far and

multiple moving parts can start looking like they're battling for space. An effect that looked great on big viewports can become muddled and confusing when it's reframed in a smaller space.

Making animated movements smaller will do the trick for simple motion like a basic move across the screen. But for more complex animation on smaller viewports, you'll need to simplify and reduce the number of moving parts. The key to this is determining what the vital parts of the animation are, to zone in on the parts that are most important to its message. Then remove the less necessary bits to distill the motion's message down to the essentials.

For example, **Rally Interactive's navigation** folds down into place with two triangle shapes unfolding each corner on larger viewports. If this exact motion was just scaled down for narrower spaces the two corners would overlap as they unfolded. It would look unnatural and wouldn't make much sense.

Open video

The main purpose of this animation is to show an unfolding action. To simplify the animation, Rally unfolds only one side for narrower viewports, with a slightly different animation. The action is still easily interpreted as unfolding and it's done in a way that is a better fit for

the available space. The message the motion was meant to convey has been preserved while the amount of motion was simplified.

Open video

Si Digital does something similar. The main concept of the design is to portray the studio as a creative lab. On large viewports, this is accomplished primarily through an animated illustration that runs the full length of the site and triggers its animations based on your scroll position. The illustration is there to support the laboratory concept visually, but it doesn't contain critical content.

Open video

At first, it looks like Si Digital just turned off the animation of the illustration for smaller viewports. But they've actually been a little cleverer than that. They've also reduced the complexity of the illustration itself. Both the amount of motion (reduced down to no motion) and the illustration were simplified to create a result that is much easier to glean the concept from.

Open video

The most interesting thing about these two examples is that they're solved more with thoughtful art direction than complex code. Keeping the main concept of the animations at the forefront allowed each to adapt creative design solutions to viewports of varying size without losing the integrity of their design.

RESPONSIVE CHOREOGRAPHY

Static content gets moved around all the time in responsive design. A three-column layout might line up from left to right on wide viewports, then stack top to bottom on narrower viewports. The same approach can be used to arrange animated content for narrower views, but the animation's choreography also needs to be adjusted for the new layout. Even with static content, just scaling it down or zooming out to fit it into the available space is rarely an ideal solution. Rearranging your animations' choreography to change which animation starts when, or even which animations play at all, keeps your animated content readable on smaller viewports.

In a recent project I had three small animations that played one after the other, left to right, on wider viewports but needed to be stacked on narrower viewports to be large enough to see. On wide viewports, all three animations could play one right after the other in

sequence because all three were in the viewable area at the same time. But once these were stacked for the narrower viewport layouts, that sequence had to change.

Open video

What was essentially one animation on wider viewports became three separate animations when stacked on narrower viewports. The layout change meant the choreography had to change as well. Each animation starts independently when it comes into view in the stacked layout instead of playing automatically in sequence. (I've put the animated parts in **this demo** if you want to peek under the hood.)

Open video

I choose to use the **GreenSock** library, with the choreography defined in two different timelines for this particular project. But the same goals could be accomplished with other JavaScript options or even CSS keyframe animations and media queries.

Even more complex responsive choreography can be pulled off with SVG. Media queries can be used to change CSS animations applied to SVG elements at specific breakpoints for starters. For even more responsive power, SVG's `viewBox` property, and the positioning of the objects

within it, can be adjusted at JavaScript-defined breakpoints. This lets you set rules to crop the viewable area and arrange your animating elements to fit any space.

Sarah Drasner has some great examples of how to use this technique with style in this [responsive infographic](#) and this [responsive interactive illustration](#). On the other hand, if smart scalability is what you're after, it's also possible to make all of an SVG's shapes and motion scale with the SVG canvas itself. Sarah covers both these clever responsive SVG techniques in detail. Creative and complex animation can easily become responsive thanks to the power of SVG!

[Open video](#)

BAKE PERFORMANCE INTO YOUR DESIGN DECISIONS

It's hard to get very far into a responsive design discussion before performance comes up. Performance goes hand in hand with responsive design and your animation decisions can have a big impact on the overall performance of your site.

The `translate3D` “hack”, `backface-visibility: hidden`, and the `will-change` property are the heavy hitters of animation performance. But decisions made earlier in

your animation design process can have a big impact on rendering performance and your performance budget too.

Pick a technology that matches your needs

One of the biggest advantages of the current web animation landscape is the range of tools we have available to us. We can use CSS animations and transitions to add just a dash of interface animation to our work, go all out with WebGL to create a 3D experience, or anywhere in between. All within our browsers! Having this huge range of options is amazing and wonderful but it also means you need to be cognizant of what you're using to get the job done.

Loading in the full weight of a robust JavaScript animation library is going to be overkill if you're only animating a few small elements here and there. That extra overhead will have an impact on performance. Performance budgets will not be pleased.

Always match the complexity of the technology you choose to the complexity of your animation needs to avoid unnecessary performance strain. For small amounts of animation, stick to CSS solutions since it's the most lightweight option. As your animations grow in complexity, or start to require more robust logic, move to a JavaScript solution that can accomplish what you need.

Animate the most performant properties

Whether you're animating in CSS or JavaScript, you're affecting specific properties of the animated element. Browsers can animate some properties more efficiently than others based on how many steps need to happen behind the scenes to visually update those properties.

Browsers are particularly efficient at animating opacity, scale, rotation, and position (when the latter three are done with transforms). [This article from Paul Irish and Paul Lewis](#) gives the full scoop on why. Conveniently, those are also the most common properties used in motion design. There aren't many animated effects that can't be pulled off with this list. Stick to these properties to set your animations up for the best performance results from the start. If you find yourself needing to animate a property outside of this list, check [CSS Triggers...](#) to find out how much of an additional impact it might have.

Offset animation start times

Offsets (the concept of having a series of similar movements execute one slightly after the other, creating a wave-like pattern) are a long-held motion graphics trick for creating more interesting and organic looking motion. Employing this trick of the trade can also be smart for performance. Animating a large number of objects all at

the same time can put a strain on the browser's rendering abilities even in the best cases. Adding short delays to offset these animations in time, so they don't all start at once, can improve rendering performance.

GO EXPLORE THE RESPONSIVE ANIMATION POSSIBILITIES FOR YOURSELF!

With smart art direction, responsive choreography, and an eye on performance you can create just about any creative web animation you can think up while still being responsive. Keep these in mind for your next project and you'll pull off your animations with style at any viewport size!

ABOUT THE AUTHOR



Val is a designer and web animation consultant with a talent for getting designers and developers alike excited about the power of animation. She is the author of *The Pocket Guide to CSS Animations* and the upcoming *Designing Interface Animations*.

She curates the UI Animation Newsletter, hosts the All The Right Moves screencast, and co-hosts the Motion and Meaning podcast. Val leads workshops at companies and conferences around the world on motion design for the web and loves every minute of it.

10. Putting My Patterns through Their Paces

Ethan Marcotte

24ways.org/201510

Over the last few years, the conversation around responsive design has shifted subtly, focusing not on designing *pages*, but on *patterns*: understanding the small, reusable elements that comprise a larger design system. And given that many of those patterns are themselves responsive, learning to manage these small layout systems has become a big part of my work.

The thing is, the more pattern-driven work I do, the more I realize my design process has changed in a number of subtle, important ways. I suppose you might even say that pattern-driven design has, in a few ways, redesigned me.

MEET THE TEASER

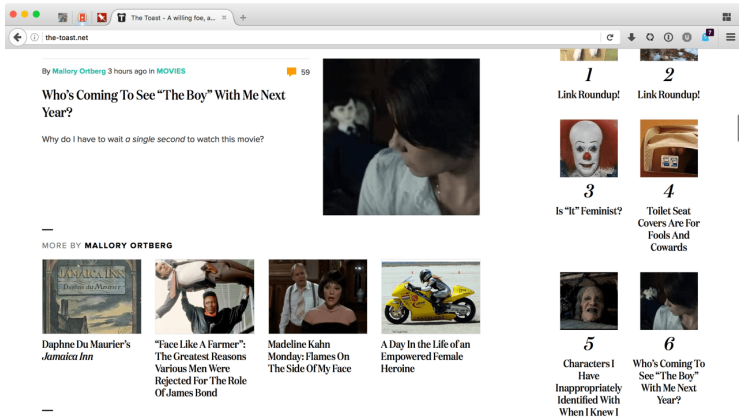
Here's a recent example. A few months ago, some friends and I redesigned The Toast. (It was a *really, really* fun project, and we learned a lot.) Each page of the site is, as you might guess, stitched together from a host of tiny,

reusable patterns. Some of them, like the search form and footer, are fairly unique, and used once per page; others are used more liberally, and built for reuse. The most prevalent example of these more generic patterns is *the teaser*, which is classed as, uh, . teaser. (Look, I never said I was especially clever.)

In its simplest form, a teaser contains a headline, which links to an article:

How To Make Sure You Get Enough To Eat At Holiday Parties

Fairly straightforward, sure. But it's just the foundation: from there, teasers can have a byline, a description, a thumbnail, and a comment count. In other words, we have a basic building block (. teaser) that contains a few discrete content types – some required, some not. In fact, very few of those pieces need to be present; to qualify as a teaser, all we really need is a link and a headline. But by adding more elements, we can build slight *variations* of our teaser, and make it much, much more versatile.



10-1. Nearly every element visible on this page is built out of our generic “teaser” pattern.

But the teaser variation I’d like to call out is the one that appears on The Toast’s homepage, on search results or on section fronts. In the main content area, each teaser in the list features larger images, as well as an interesting visual treatment: the byline and comment count were the most prominent elements within each teaser, appearing above the headline.

By [Author Name](#) 3 hours ago in [RACE](#)

8

Article Title Goes Here

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Alias, ut, necessitatibus error quo magni porro perspiciatis esse hic praesentium earum fuga ex laborum voluptate debitis quibusdam!



10-2. The approved visual design of our teaser, as it appears on lists on the homepage and the section fronts.

And this is, as it happens, the teaser variation that gave me pause. Back in the old days – you know, like six months ago – I probably would’ve marked this module up to match the design. In other words, I would’ve looked at the module’s visual hierarchy (metadata up top, headline and content below) and written the following HTML:

```
<div class="teaser">
  <p class="article-byline">By <a href="#">Author
Name</a></p>
  <a class="comment-count" href="#">126
<i>comments</i></a>
  <h1 class="article-title"><a href="#">Article
Title</a></h1>
  <p class="teaser-excerpt">Lorem ipsum dolor sit amet,
consectetur...</p>
</div>
```

But then I caught myself, and realized this wasn’t the best approach.

MOVING BEYOND LAYOUT

Since I've started working responsively, there's a question I work into every step of my design process. Whether I'm working in Sketch, CSSing a thing, or researching a project, I try to constantly ask myself:

What if someone doesn't browse the web like I do?

...Okay, that doesn't seem especially fancy. (And maybe you came here for fancy.) But as straightforward as that question might seem, it's been invaluable to so many aspects of my practice. If I'm working on a widescreen layout, that question helps me remember the constraints of the small screen; if I'm working on an interface that has some enhancements for touch, it helps me consider other input modes as I work. It's also helpful as a reminder that many **might not see the screen the same way I do**, and that accessibility (**in all its forms**) should be a throughline for our work on the web.

And that last point, thankfully, was what caught me here. While having the byline and comment count at the top was a lovely *visual* treatment, it made for a terrible content hierarchy. For example, it'd be a little weird if the page was being read aloud in a speaking browser: the name of the author and the number of comments would be read aloud *before* the title of the article with which they're associated.

That's why I find it's helpful to begin designing a pattern's hierarchy *before* its layout: to move past the visual presentation in front of me, and focus on the underlying content I'm trying to support. In other words, if someone's encountering my design without the CSS I've written, what should their experience be?

So I took a step back, and came up with a different approach:

```
<div class="teaser">
  <h1 class="article-title"><a href="#">Article
Title</a></h1>
  <h2 class="article-byline">By <a href="#">Author
Name</a></h2>
  <p class="teaser-excerpt">
    Lorem ipsum dolor sit amet, consectetur...
    <a class="comment-count" href="#">126
  <i>comments</i></a>
  </p>
</div>
```

Much, much better. This felt like a better match for the content I was designing: the headline – easily most important element – was at the top, followed by the author's name and an excerpt. And while the comment count is *visually* the most prominent element in the teaser, I decided it was *hierarchically* the least critical: that's why it's at the very end of the excerpt, the last element within our teaser. And with some light styling, we've got a respectable-looking hierarchy in place:

Article Title Goes Here

By [Author Name](#) 3 hours ago in [RACE](#)

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias, ut, necessitatibus error quo magni porro perspiciatis esse hic praesentium earum fuga ex laborum voluptate debitis quibusdam!



Yeah, you're right – it's not our final design. But from this basic-looking foundation, we can layer on a bit more complexity. First, we'll bolster the markup with an extra element around our title and byline:

```
<div class="teaser">
  <div class="teaser-hed">
    <h1 class="article-title"><a href="#">Article
Title</a></h1>
    <h2 class="article-byline">By <a href="#">Author
Name</a></h2>
  </div>
  ...
</div>
```


With that in place, we can use **flexbox** to tweak our layout, like so:

```
.teaser-hed {
  display: flex;
  flex-direction: column-reverse;
}
```

`flex-direction: column-reverse` acts a bit like a change in gravity within our `teaser-hed` element, vertically swapping its two children.

By [Author Name](#) 4 hours ago in [RACE](#)

Article Title Goes Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Accusamus, dolores, modi cum vitae quam reiciendis et mollitia velit deleniti earum fuga culpa magni quo amet laboriosam!  11

Getting closer! But as great as flexbox is, it doesn't do anything for elements *outside* our container, like our little comment count, which is, as you've probably noticed, still stranded at the very bottom of our teaser.

Flexbox is, as you might already know, wonderful! And while it enjoys **incredibly broad support**, there are enough implementations of old versions of Flexbox (in addition to **plenty of bugs**) that I tend to use a feature test to check if the browser's using a sufficiently modern version of flexbox. Here's the one we used:

```
var doc = document.body || document.documentElement;
var style = doc.style;

if ( style.webkitFlexWrap == '' ||
    style.msFlexWrap == '' ||
```

```
style.flexWrap == '' ) {  
  doc.className += " supports-flex";  
}
```

Eagle-eyed readers will note we could have used `@supports` **feature queries** to ask browsers if they support certain CSS properties, removing the JavaScript dependency. But since we wanted to **serve the layout to IE** we opted to write a little question in JavaScript, asking the browser if it supports `flex-wrap`, a property used elsewhere in the design. If the browser passes the test, then a class of `supports-flex` gets applied to our `html` element. And with that class in place, we can safely quarantine our flexbox-enabled layout from less-capable browsers, and finish our teaser's design:

```
.supports-flex .teaser-hed {  
  display: flex;  
  flex-direction: column-reverse;  
}  
.supports-flex .teaser .comment-count {  
  position: absolute;  
  right: 0;  
  top: 1.1em;  
}
```

If the `supports-flex` class is present, we can apply our flexbox layout to the title area, sure – but we can *also* safely use absolute positioning to pull our comment count out of its default position, and anchor it to the top right of our teaser. In other words, the browsers that don't meet

our threshold for our advanced styles are left with an attractive design that matches our HTML's content hierarchy; but the ones that pass our test receive the finished, final design.

Article Title Goes Here

By [Author Name](#) 3 hours ago in [RACE](#)

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias, ut, necessitatibus error quo magni porro perspiciatis esse hic praesentium earum fuga ex laborum voluptate debitis quibusdam!

8

By [Author Name](#) 3 hours ago in [RACE](#)

8

Article Title Goes Here

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias, ut, necessitatibus error quo magni porro perspiciatis esse hic praesentium earum fuga ex laborum voluptate debitis quibusdam!

The Baseline

The Refinement

And with that, our teaser's complete.

DIVING INTO DEVICE-AGNOSTIC DESIGN

This is, admittedly, a pretty modest application of flexbox. (For some truly next-level work, I'd recommend Heydon Pickering's "Flexbox Grid Finesse", or anything Zoe Mickley Gillenwater publishes.) And for such a simple module, you might feel like this is, well, quite a bit of work. And you'd be right! In fact, it's not one layout, but two: a lightly styled content hierarchy served to everyone, with the finished design served conditionally to the browsers that can successfully implement it. But I've found that thinking about my design as existing in broad experience tiers – in layers – is one of the best ways of designing for the modern web. And what's more, it works not just for simple modules like our teaser, but for more complex or interactive patterns as well.

Open video

10-3. Even a simple search form can be conditionally enhanced, given a little layered thinking.

This more layered approach to interface design isn't a new one, mind you: it's been championed by everyone from **Filament Group** to the **BBC**. And with all the challenges we keep uncovering, a more device-agnostic approach is one of the best ways I've found to practice responsive design. As **Trent Walton** once wrote,

Like cars designed to perform in extreme heat or on icy roads, websites should be built to face the reality of the web's inherent variability.

We have a weird job, working on the web. We're designing for the latest mobile devices, sure, but we're increasingly aware that our definition of "smartphone" is **much too narrow**. Browsers have started appearing on **our wrists** and in **our cars' dashboards**, but much of the world's mobile data flows over **sub-3G networks**. After all, the web's evolution has never been charted along a straight line: it's simultaneously getting slower and faster, with devices new and old coming online every day. With all the challenges in front of us, including many we don't yet know about, a more device-agnostic, more layered design process can better prepare our patterns – and ourselves – for the future.

(It won't help you get enough to eat at holiday parties, though.)

ABOUT THE AUTHOR



Ethan Marcotte is an independent designer and developer, and the fellow who coined the term “responsive web design”. He is the author of two books on the topic, *Responsive Web Design* and *Responsive Design: Patterns and Principles*, and has been known to give a conference talk or two. Ethan is passionate about digital design, emerging markets, and ensuring global access, and has been known to link to the odd GIF now and again.

11. Upping Your Web Security Game

Guy Podjarny

24ways.org/201511

When I started working in web security fifteen years ago, web development looked very different. The few non-static web applications were built using a waterfall process and shipped quarterly at best, making it possible to add security audits before every release; applications were deployed exclusively on in-house servers, allowing Info Sec to inspect their configuration and setup; and the few third-party components used came from a small set of well-known and trusted providers. And yet, even with these favourable conditions, security teams were quickly overwhelmed and called for developers to build security in.

If the web security game was hard to win before, it's doomed to fail now. In today's web development, every other page is an application, accepting inputs and private data from users; software is built continuously, designed

to eliminate manual gates, including security gates; **infrastructure is code**, with servers spawned with little effort and even less security scrutiny; and most of the code in a typical application is third-party code, pulled in through open source repositories with rarely a glance at who provided them.

Security teams, when they exist at all, cannot solve this problem. They are vastly outnumbered by developers, and cannot keep up with the application's pace of change. For us to have a shot at making the web secure, we must bring security into the core. We need to give it no less attention than that we give browser compatibility, mobile design or web page load times. More broadly, we should see security as an aspect of quality, expecting both ourselves and our peers to address it, and taking pride when we do it well.

WHERE TO START?

Embracing security isn't something you do overnight.

A good place to start is by reviewing things you're already doing – and trying to make them more secure. Here are three concrete steps you can take to get going.

HTTPS

Threats begin when your system interacts with the outside world, which often means HTTP. As is, HTTP is painfully insecure, allowing attackers to easily steal and manipulate data going to or from the server. HTTPS adds a layer of crypto that ensures the parties know who they're talking to, and that the information exchanged can be neither modified nor sniffed.

HTTPS is relevant to any site. If your non-HTTPS site holds opinions, reading it may get your users in trouble with employers or governments. If your users believe what you say, **attackers can modify your non-HTTPS** to take advantage of and abuse that trust. If you want to use new browser technologies like **HTTP2** and **service workers**, your site will need to be HTTPS. And if you want to be discovered on the web, using HTTPS **can help your Google ranking**. For more details on why I think you should make the switch to HTTPS, check out **this post**, **these slides** and **this video**.

Using HTTPS is becoming easier and cheaper. Here are a few free tools that can help:

- Get free and easy HTTPS delivery from **Cloudflare** (be sure to use “**Full SSL**”!)
- Get a free and automation-friendly certificate from **Let's Encrypt** (now in open beta).
- Test how well your HTTPS is set up using **SSLTest**.

Other vendors and platforms are rapidly simplifying and reducing the cost of their HTTPS offering, as demand and importance grows.

Two-Factor Authentication

The most sensitive data is usually stored behind a login, and the authentication process is the primary gate in front of this data. Making this process secure has many aspects, including using HTTPS when accepting credentials, having a strong password policy, never storing the password, and more.

All of these are important, but the best single step to boost your authentication security is to introduce **two-factor authentication (2FA)**. Adding 2FA usually means prompting users for an additional one-time code when logging in, which they get via SMS or a mobile app (e.g. **Google Authenticator**). This code is short-lived and is extremely hard for a remote attacker to guess, thus vastly reducing the risk a leaked or easily guessed password presents.

The typical algorithm for 2FA is based on an IETF standard called the **time-based one-time password (TOTP) algorithm**, and it isn't that hard to implement. Joel Franusic wrote a **great post** on implementing 2FA; modules like **speakeasy** make it even easier; and you can swap SMS with Google Authenticator or your own app if

you prefer. If you don't want to build 2FA support yourself, you can purchase two/multi-factor authentication services from vendors such as DuoSecurity, Auth0, Clef, Hypr and others.

If implementing 2FA still feels like too much work, you can also choose to offload your entire authentication process to an OAuth-based federated login. Many companies offer this today, including Facebook, Google, Twitter, GitHub and others. These bigger players tend to do authentication well and support 2FA, but you should consider what data you're sharing with them in the process.

Tracking Known Vulnerabilities

Most of the code in a modern application was actually written by third parties, and pulled into your app as frameworks, modules and libraries. While using these components makes us much more productive, along with their functionality we also adopt their security flaws. To make things worse, some of these flaws are **well-known vulnerabilities**, making it easy for hackers to take advantage of them in an attack.

This is a real problem and happens on pretty much every platform. Do you develop in Java? In 2014, **over 6% of Java modules** downloaded from Maven had a known severe security issue, the typical Java applications

containing 24 flaws. Are you coding in Node.js? Roughly 14% of npm packages carry a known vulnerability, and over 60% of dev shops find vulnerabilities in their code. 30% of Docker Hub containers include a high priority known security hole, and 60% of the top 100,000 websites use client-side libraries with known security gaps.

To find known security issues, take stock of your dependencies and match them against language-specific lists such as **Snyk's vulnerability DB** for Node.js, **rubysec** for Ruby, **victims-db** for Python and **OWASP's Dependency Check** for Java. Once found, you can fix most issues by upgrading the component in question, though that may be tricky for indirect dependencies.

This process is still way too painful, which means most teams don't do it. The Snyk team and I are hoping to change that by making it as easy as possible to find, fix and monitor known vulnerabilities in your dependencies. Snyk's wizard will help you find and fix these issues through guided upgrades and patches, and adding Snyk's test to your continuous integration and deployment (CI/CD) will help you stay secure as your code evolves.

Note that newly disclosed vulnerabilities usually impact old code – the one you're running in production. This means you have to stay alert when new vulnerabilities are disclosed, so you can fix them before attackers can exploit

them. You can do so by subscribing to vulnerability lists like US-CERT, OSVDB and NVD. Snyk's monitor will proactively let you know about new disclosures relevant to your code, but only for Node.js for now – you can **register** to get updated when we expand.

SECURING YOURSELF

In addition to making your application secure, you should make the contributors to that application secure – including you. Earlier this year we've seen attackers target mobile app developers with a malicious Xcode. The real target, however, wasn't these developers, but rather the users of the apps they create. That **you** create. Securing your own work environment is a key part of keeping your apps secure, and your users from being compromised.

There's no single step that will make you fully secure, but here are a few steps that can make a big impact:

1. **Use 2FA on all the services related to the application**, notably source control (e.g. **GitHub**), cloud platform (e.g. **AWS**), CI/CD, CDN, DNS provider and domain registrar. If an attacker compromises any one of those, they could modify or replace your entire application. I'd recommend using 2FA on all your personal services too.

2. **Use a password manager** (e.g. 1Password, LastPass) to ensure you have a separate and complex password for each service. Some of these services *will* get hacked, and passwords *will* leak. When that happens, don't let the attackers access your other systems too.
3. **Secure your workstation.** Be careful what you download, lock your screen when you walk away, change default passwords on services you install, run antivirus software, etc. Malware on your machine can translate to malware in your applications.
4. **Be very wary of phishing.** Smart attackers use 'spear phishing' techniques to gain access to specific systems, and can trick even security savvy users. There are even phishing scams targeting users with 2FA. Be alert to phishy emails.
5. **Don't install things through curl** `<somewhere-on-the-web> | sudo bash`, especially if the URL is on GitHub, meaning someone else controls it. Don't do it on your machines, and **definitely** don't do it in your CI/CD systems. Seriously.

Staying secure should be important to you personally, but it's doubly important when you have privileged access to an application. Such access makes you a way to reach many more users, and therefore a more compelling target for bad actors.

A CULTURE OF SECURITY

Using HTTPS, enabling two-factor authentication and fixing known vulnerabilities are significant steps in building security at your core. As you implement them, remember that these are just a few steps in a longer journey.

The end goal is to embrace security as an aspect of quality, and accept we all share the responsibility of keeping ourselves – and our users – safe.

ABOUT THE AUTHOR



Guy is the CEO & Founder of [Snyk.io](https://snyk.io), a Web Security company. Before that, he was the CTO of Akamai's Web Performance business, following its acquisition of Blaze.io (which he co-founded). He has spent over a decade working on Web Application Security, and more specifically the first Web App Firewall (AppShield) and the market leading Web Application Security scanner (AppScan).

He is also the author of *Mobitest*, an open-source mobile web performance testing tool, and is on the programming committee of the Velocity conference, and wrote *Responsive & Fast*, a short book about RWD Performance.

12. Be Fluid with Your Design Skills: Build Your Own Sites

Ros Horner

24ways.org/201512

Just five years ago in 2010, when we were all busy trying to surprise and delight, learning CSS3 and trying to get whole websites onto one page, we had a poster on our studio wall. It was entitled ‘Designers Vs Developers’, an infographic that showed us the differences between the men(!) who created websites.

Designers wore skinny jeans and used Macs and developers wore cargo pants and brought their own keyboards to work. We began to learn that designers and developers were not only doing completely different jobs but were completely different people in every way. This opinion was backed up by hundreds of memes, millions of tweets and pages of articles which used words like *void* and *battle* and *versus*.

Thankfully, things move quickly in this industry; the wide world of web design has moved on in the last five years. There are new devices, technologies, tools – and even a few women. Designers have been helped along by great apps, software, open source projects, conferences, and a community of people who, to my unending pride, love to share their knowledge and their work.

So the world has moved on, and if Miley Cyrus, Ruby Rose and Eliot Sumner are identifying as gender fluid (an identity which refers to a gender which varies over time or is a combination of identities), then I would like to come out as discipline fluid!

OK, I will probably never identify as a developer, but I will identify as fluid! How can we be anything else in an industry that moves so quickly? That's how we should think of our skills, our interests and even our job titles. After all, Steve Jobs told us that "Design is not just what it looks like and feels like. Design is how it works." Sorry skinny-jean-wearing designers – this means we're all designing something together. And it's not just about knowing the right words to use: you have to know how it feels. How it feels when you make something work, when you fix that bug, when you make it work on IE.

Like anything in life, things run smoothly when you make the effort to share experiences, empathise and deeply understand the needs of others. How can designers do

that if they've never built their own site? I'm not talking the big stuff, I'm talking about your portfolio site, your mate's business website, a website for that great idea you've had. I'm talking about doing it yourself to get an unique insight into how it feels.

We all know that designers and developers alike love an ``, so here it is.

TEN REASONS DESIGNERS SHOULD BE FLUID WITH THEIR SKILLS AND BUILD THEIR OWN SITES

1. It's never been easier

Now here's where the definition of 'build' is going to get a bit loose and people are going to get angry, but when I say it's never been easier I mean because of the existence of apps and software like WordPress, Squarespace, Tumblr, et al. It's easy to make something and get it out there into the world, and these are all gateway drugs to hard coding!

2. You'll understand how it feels

How it feels to be so proud that something actually works that you momentarily don't notice if the kerning is off or the padding is inconsistent. How it feels to see your site

appear when you've redirected a URL. How it feels when you just can't work out where that one extra space is in a line of PHP that has killed your whole site.

3. It makes you a designer

Not a better designer, it makes you a designer when you are designing how things look and how they *work*.

4. You learn about movement

Photoshop and Sketch just don't cut it yet. Until you see your site in a browser or your app on a phone, it's hard to imagine how it *moves*. Building your own sites shows you that it's not just about how the content looks on the screen, but how it moves, interacts and feels.

5. You make techie friends

All the tutorials and forums in the world can't beat your network of techie friends. Since I started working in web design I have worked with, sat next to, and co-created with some of the greatest developers. Developers who've shared their knowledge, encouraged me to build things, patiently explained HTML, CSS, servers, divs, web fonts, iOS development. There has been no void, no *versus*, very few battles; just people who share an interest and love of making things.

6. You will own domain names

When something is paid for, online and searchable then it's real and you've got to put the work in. Buying domains has taught me how to stop procrastinating, but also about DNS, FTP, email, and how servers work.

7. People will ask you to do things

Learning about code and development opens a whole new world of design. When you put your own personal websites and projects out there people ask you to do more things. OK, so sometimes those things are "Make me a website for free", but more often it's cool things like "Come and speak at my conference", "Write an article for my magazine" and "Collaborate with me."

8. The young people are coming!

They love typography, they love print, they love layout, but they've known how to put a website together since they started their first blog aged five and they show me clever apps they've knocked together over the weekend! They're new, they're fluid, and they're better than us!

9. Your portfolio is your portfolio

OK, it's an obvious one, but as designers our work is our CV, our legacy! We need to show our skill, our attention to detail and our creativity in the way we showcase our

work. Building your portfolio is the best way to start building your own websites. (And please be that designer who's bothered to work out how to change the Squarespace favicon!)

10. It keeps you fluid!

Building your own websites is tough. You'll never be happy with it, you'll constantly be updating it to keep up with technology and fashion, and by the time you've finished it you'll want to start all over again. Perfect for forcing you to stay up-to-date with what's going on in the industry.

</o1>

ABOUT THE AUTHOR



Ros Horner is a London based (print turned digital) Design Director and speaker, working on apps and websites for clients such as Adidas, Reebok, McLaren, and Volvo.

13. Designing with Contrast

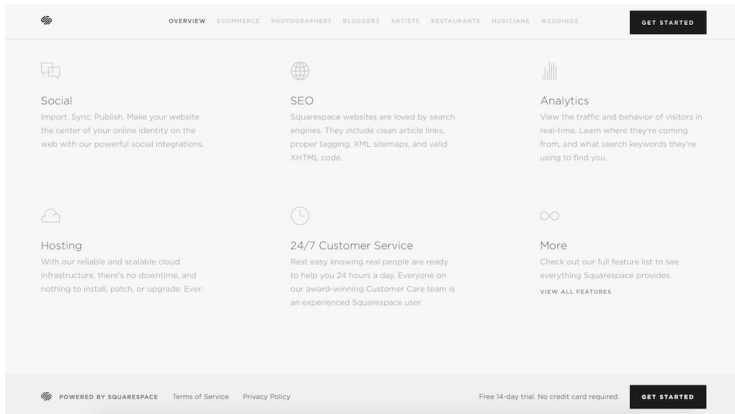
Mark Mitchell

24ways.org/201513

When an appetite for aesthetics over usability becomes the bellwether of user interface design, it's time to reconsider who we're designing for.

Over the last few years, we have questioned the signifiers that gave obvious meaning to the function of interface elements. Strong textures, deep shadows, gradients — imitations of physical objects — were discarded. And many, rightfully so. Our audiences are now more comfortable with an experience that feels native to the technology, so we should respond in kind.

Yet not all of the changes have benefitted users. Our efforts to simplify brought with them a trend of ultra-minimalism where aesthetics have taken priority over legibility, accessibility and discoverability. The trend shows no sign of losing popularity — and it is harming our experience of digital content.



A THIN VENEER

We are in a race to create the most subdued, understated interface. Visual contrast is out. In its place: the thinnest weights of a typeface and white text on bright color backgrounds. Headlines, text, borders, backgrounds, icons, form controls and inputs: all grey.

While we can look back over the last decade and see minimalist trends emerging on the web, I think we can place a fair share of the responsibility for the recent shift in priorities on Apple. The release of iOS 7 ushered in a radical change to its user interface. It paired mobile interaction design to the simplicity and eloquence of Apple's marketing and product design. It was a catalyst. We took what we saw, copied and consumed the aesthetics like pick-and-mix.

New technology compounds this trend. Computer monitors and mobile devices are available with screens of unprecedented resolutions. Ultra-light type and subtle hues, difficult to view on older screens, are more legible on these devices. It would be disingenuous to say that designers have always worked on machines representative of their audience's circumstances, but the gap has never been as large as it is now. We are running the risk of designing VIP lounges where the cost of entry is a Mac with a Retina display.

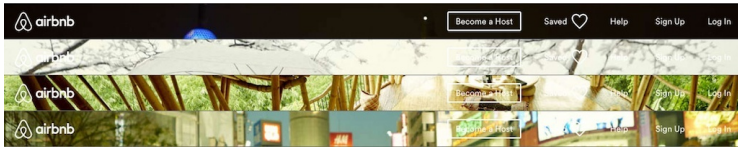
MINIMALIST EXPECTATIONS

Like progressive enhancement in an age of JavaScript, many good and sensible accessibility practices are being overlooked or ignored. We're driving unilateral design decisions that threaten accessibility. We've approached every problem with the same solution, grasping on to the integrity of beauty, focusing on expression over users' needs and content.

Someone once suggested to me that a client's website should include two states. The first state would be the *ideal* experience, with low color contrast, light font weights and no differentiation between links and text. It would be the default. The second state would be presented in whatever way was necessary to meet accessibility standards. Users would have to opt out of the default state via a toggle if it wasn't meeting their needs.

A sort of first-class, upper deck cabin equivalent of graceful degradation. That this would divide the user base was irrelevant, as the aesthetics of the brand were absolute.

It may seem like an unusual anecdote, but it isn't uncommon to see this thinking in our industry. Again and again, we place the burden of responsibility to participate in a usable experience on others. We view accessibility and good design as mutually exclusive. Taking for granted what users will tolerate is usually the forte of monopolistic services, but increasingly we apply the same arrogance to our new products and services.



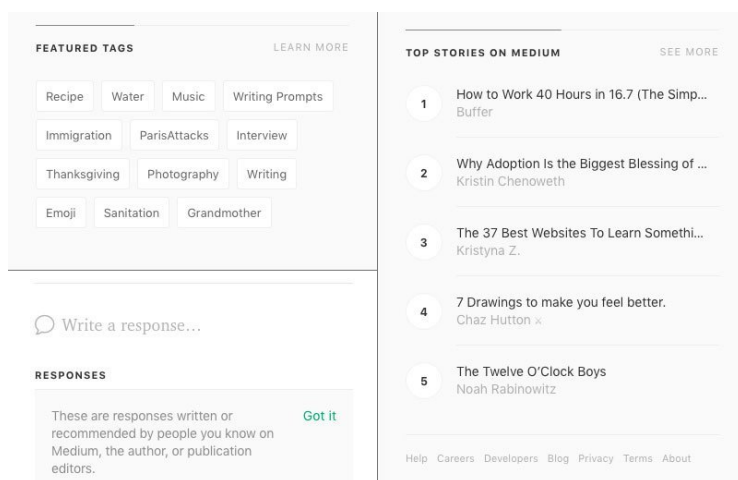
IMITATION WITHOUT REPRESENTATION

All of us are influenced in one way or another by one another's work. We are consciously and unconsciously affected by the visual and audible activity around us. This is important and unavoidable. We do not produce work in a vacuum. We respond to technology and culture. We channel language and geography. We absorb the sights and sounds of film, television, news. To mimic and copy is part and parcel of creating something an audience of

many can comprehend and respond to. Our clients often look first to their competitors' products to understand their success.

However, problems arise when we focus on style without context; form without function; mimicry as method. Copied and reused without any of the ethos of the original, stripped of deliberate and informed decision-making, the so-called look and feel becomes nothing more than paint on an empty facade.

The typographic and color choices so in vogue today with our popular digital products and services have little in common with the brands they are meant to represent.



FOR WANT OF GOOD DESIGN, THE MESSAGE WAS LOST

The question to ask is: does the interface truly reflect the product? Is it an accurate characterization of the brand and organizational values? Does the delivery of the content match the tone of voice?

The answer is: probably not. Because every organization, every app or service, is unique. Each with its own personality, its own values and wonderful quirks. Design is communication. We should do everything in our role as professionals to use design to give voice to the message. Our job is to clearly communicate the benefits of a service and unreservedly allow access to information and content. To do otherwise, by obscuring with fashionable styles and elusive information architecture, does a great disservice to the people who chose to engage with and trust our products.

We can achieve hierarchy and visual rhythm without resorting to extreme reduction. We can craft a beautiful experience with fine detail and curiosity while meeting fundamental standards of accessibility (and strive to meet many more).

STANDARDS OF EXCELLENCE

It isn't always comfortable to step back and objectively question our design choices. We get lost in the flow of our work, using patterns and preferences we've tried and tested before. That our decisions often seem like second nature is a gift of experience, but sometimes it prevents us from finding our blind spots.

I was first caught out by my own biases a few years ago, when designing an interface for the Bank of England. After deciding on the colors for the typography and interactive elements, I learned that the site had to meet AAA accessibility standards. My choices quickly fell apart. It was eye-opening. I had to start again with restrictions and use size, weight and placement instead to construct the visual hierarchy.

Even now, I make mistakes. On a recent project, I used large photographs on an organization's website to promote their products. Knowing that our team had control over the art direction, I felt confident that we could compose the photographs to work with text overlays. Despite our best effort, the cropped images weren't always consistent, undermining the text's legibility. If I had the chance to do it again, I would separate the text and image.

So, what practical things can we consider to give our users the experience they deserve?

Put guidelines in place

- Think about your brand values. Write down keywords and use them as a framework when choosing a typeface. Explore colors that convey the organization's personality and emotional appeal.
- Define a color palette that is web-ready and meets minimum accessibility standards. Note which colors are suitable for use with text. Only very dark hues of grey are consistently legible so keep them for non-essential text (for example, as placeholders in form inputs).
- Find which **background colors you can safely use with white text**, and consider integrating contrast checks into your workflow.
- Use roman and medium weights for body copy. Reserve lighter weights of a typeface for very large text. Thin fonts are usually the first to break down because of aliasing differences across platforms and screens.
- Check that the size, leading and length of your type is always legible and readable. Define lower and upper limits. Small text is best left for captions and words in uppercase.
- Avoid overlaying text on images unless it's guaranteed to be legible. If it's necessary to optimize space in the layout, give the text a container. Scrims aren't always reliable: the text will inevitably overlap a part of the photograph without a contrasting ground.

Test your work

- Review legibility and contrast on different devices. It's just as important as testing the layout of a responsive website. If you have a local device lab, pay it a visit.
- Find a computer monitor near a window when the sun is shining. Step outside the studio and try to read your content on a mobile device with different brightness levels.
- Ask your friends and family what they use at home and at work. It's one way of making sure your feedback isn't always coming from a closed loop.

Push your limits

- You define what the user sees. If you've inherited brand guidelines, **question them**. If you don't agree with the choices, make the case for why they should change.
- Experiment with size, weight and color to find contrast. Objects with low contrast appear similar to one another and undermine the visual hierarchy. Weak relationships between figure and ground diminish visual interest. A balanced level of contrast removes ambiguity and creates focal points. It captures and holds our attention.
- If you're lost for inspiration, look to graphic design in print. We have a wealth of history, full of examples that excel in using contrast to establish visual hierarchy.
- Embrace limitations. Use boundaries as an opportunity to explore possibilities.

MORE THAN JUST A FACADE

Designing with standards encourages legibility and helps to define a strong visual hierarchy. Design without exclusion (through neither negligence or intent) gets around discussions of demographics, speaks to a larger audience and makes good business sense. Following the latest trends not only weakens usability but also hinders a cohesive and distinctive brand.

Users will make means when they need to, by increasing browser font sizes or enabling system features for accessibility. But we can do our part to take as much of that burden off of the user and ask less of those who need it most.

In architecture, it isn't buildings that mimic what is fashionable that stand the test of time. Nor do we admire buildings that tack on separate, poorly constructed extensions to meet a bare minimum of safety regulations. We admire architecture that offers well-considered, remarkable, usable spaces with universal access.

Perhaps we can take inspiration from these spaces. Let's give our buildings a bold voice and make sure the doors are open to everyone.

ABOUT THE AUTHOR



Mark Mitchell is a freelance digital designer & front-end developer in London. Find him on Twitter at [@withoutnations](https://twitter.com/withoutnations) and online at withoutnations.com.

14. What I Learned about Product Design This Year

Meagan Fisher

24ways.org/201514

2015 was a humbling year for me. In September of 2014, I joined a tiny but established startup called SproutVideo as their third employee and first designer. The role interests me because it affords the opportunity to see how design can grow a solid product with a loyal user-base into something even better.

The work I do now could also have a real impact on the brand and user experience of our product for years to come, which is a thrilling prospect in an industry where much of what I do feels small and temporary. I got in on the ground floor of something special: a small, dedicated, useful company that cares deeply about making video hosting effortless and rewarding for our users.

I had (and still have) grand ideas for what thoughtful design can do for a product, and the smaller-scale product design work I've done or helped manage over the past few years gave me enough eager confidence to dive in head first. Readers who have experience redesigning complex existing products probably have a knowing smirk on their face right now. As I said, it's been humbling. A year of focused product design, especially on the scale we are trying to achieve with our small team at SproutVideo, has taught me more than any projects in recent memory. I'd like to share a few of those lessons.

PRODUCT DESIGN IS VERY DIFFERENT FROM MARKETING DESIGN

The majority of my recent work leading up to SproutVideo has been in marketing design. These projects are so fun because their aim is to communicate the value of the product in a compelling and memorable way. In order to achieve this goal, I spent a lot of time thinking about content strategy, responsive design, and how to create striking visuals that tell a story. These are all pursuits I love.

Product design is a different beast. When designing a homepage, I can employ powerful imagery, wild gradients, and somewhat-quirky fonts. When I began redesigning the SproutVideo product, I wanted to draw on all the

beautiful assets I've created for our marketing materials, but big gradients, textures, and display fonts made no sense in this new context.

That's because the product isn't about us, and it isn't about telling our story. Product design is about getting out of the way so people can do their job. The visual design is there to create a pleasant atmosphere for people to work in, and to help support the user experience. Learning to take "us" out of the equation took some work after years of creating gorgeous imagery and content for the sales-driven side of businesses.

I've learned it's very valuable to design both sides of the experience, because marketing and product design flex different muscles. If you're currently in an environment where the two are separate, consider switching teams in 2016. Designing for product when you've mostly done marketing, or vice versa, will deepen your knowledge as a designer overall. You'll face new unexpected challenges, which is the only way to grow.

PRODUCT DESIGN CAN NOT START WITH WHAT LOOKS GOOD ON DRIBBBLE

I have an embarrassing confession: when I began the redesign, I had a secret goal of making something that would look gorgeous in my portfolio. I have a collection of product shots that I admire on Dribbble; examples of

beautiful dashboards and widgets and UI elements that look good enough to frame. I wanted people to feel the same way about the final outcome of our redesign. Mistakenly, this was a factor in my initial work. I opened Photoshop and crafted pixel-perfect static buttons and form elements and color palettes that — when applied to our actual product — looked like a toddler beauty pageant. It added up to a lot of unusable shininess, noise, and silliness.

I was disappointed; these elements seemed so lovely in isolation, but in context, they felt tacky and overblown. I realized: I'm not here to design the world's most beautiful drop down menu. Good design has nothing to do with ego, but in my experience designers are, at least a little bit, secret divas. I'm no exception. I had to remind myself that I am not working in service of a bigger Dribbble following or to create the most Pinterest-ing work. My function is solely to serve the users — to make life a little better for the good people who keep my company in business.

This meant letting go of pixel-level beauty to create something bigger and harder: a system of elements that work together in harmony in many contexts. The visual style exists to guide the users. When done well, it becomes a language that users understand, so when they encounter a new feature or have a new goal, they already feel comfortable navigating it. This meant stripping back my gorgeous animated menu into something that didn't

detract from important neighboring content, and could easily fit in other parts of the app. In order to know what visual style would support the users, I had to take a wider view of the product as a whole.

JUST ACCEPT THAT DESIGNING A GREAT PRODUCT – LIKE MANY WORTHWHILE PURSUITS – IS INITIALLY LABORIOUS AND MESSY

Once I realized I couldn't start by creating the most Dribbble-worthy thing, I knew I'd have to begin with the unglamorous, frustrating, but weirdly wonderful work of mapping out how the product's content could better be structured. Since we're redesigning an existing product, I assumed this would be fairly straightforward: the functionality was already in place, and my job was just to structure it in a more easily navigable way.

I started by handing off a few wireframes of the key screens to the developer, and that's when the questions began rolling in: "If we move this content into a modal, how will it affect this similar action here?" "What happens if they don't add video tags, but they do add a description?" "What if the user has a title that is 500 characters long?" "What if they want their video to be private to some users, but accessible to others?"

How annoying (but really, fantastic) that people use our product in so many ways. Turns out, product design isn't about laying out elements in the most ideal scenario for the user that's most convenient for you. As product designers, we have to foresee every outcome, and anticipate every potential user need.

Which brings me to another annoying epiphany: if you want to do it well, and account for every user, product design is so much more snarly and tangled than you'd expect going in. I began with a simple goal: to improve the experience on just one of our key product pages.

However, every small change impacts every part of the product to some degree, and that impact has to be accounted for. Every decision is based on assumptions that have to be tested; I test my assumptions by observing users, talking to the team, wireframing, and prototyping. Many of my assumptions are wrong. There are days when it's incredibly frustrating, because an elegant solution for users with one goal will complicate life for users with another goal. It's vital to solve as many scenarios as possible, even though this is slow, sometimes mind-bending work.

As a side bonus, wireframing and prototyping every potential state in a product is tedious, but your developers will thank you for it. It's not their job to solve what happens when there's an empty state, error, or edge

case. Showing you've accounted for these scenarios will win a developer's respect; failing to do so will frustrate them.

WHEN YOU'VE CREATED AND TESTED A SYSTEM THAT SUPPORTS USER NEEDS, IT WILL BE BEAUTIFUL

Remember what I said in the beginning about wanting to create a Dribbble-worthy product? When I stopped focusing on the visual details of the design (color, spacing, light and shadow, font choices) and focused instead on structuring the content to maximize usability and delight, a beautiful design began to emerge naturally.

I began with grayscale, flat wireframes as a strategy to keep me from getting pulled into the visual style before the user experience was established. As I created a system of elements that worked in harmony, the visual style choices became obvious. Some buttons would need to be brighter and sit off the page to help the user spot important actions. Some elements would need line separators to create a hierarchy, where others could stand on their own as an emphasized piece of content. As the user experience took shape, the visual style emerged naturally to support it. The result is a product that feels beautiful to use, because I was thoughtful about the experience first.



A big takeaway from this process has been that my assumptions will often be proven wrong. My assumptions about how to design a great product, and how users will interact with that product, have been tested and revised repeatedly. At SproutVideo we're about to undertake the biggest test of our work; we're going to launch a small part of the product redesign to our users. If I've learned anything, it's that I will continue to be humbled by the ongoing effort of making the best product I can, which is a wonderful thing.

Next year, I hope you all get to do work that takes you out of our comfort zone. Be regularly confounded and embarrassed by your wrong assumptions, learn from them, and come back and tell us what you learned in 2016.

ABOUT THE AUTHOR



Meagan Fisher is passionate about owls, coffee, and web design. In her ongoing mission to make the web a better place, she's partnered with some of the best designers in the industry, such as [SimpleBits](#), [Happy Cog](#), and [Crush + Lovely](#). When she's not creating interfaces, she's speaking, [tweeting](#), writing on [Owltastic](#), or posting coffee art photography to [Art in my Coffee](#).

15. Grid, Flexbox, Box Alignment: Our New System for Layout

Rachel Andrew

24ways.org/201515

Three years ago for 24 ways 2012, I wrote an article about a new CSS layout method I was excited about. A specification had emerged, developed by people from the Internet Explorer team, bringing us a proper grid system for the web. In 2015, that Internet Explorer implementation is still the only public implementation of CSS grid layout. However, in 2016 we should be seeing it in a new improved form ready for our use in browsers.

Grid layout has developed hidden behind a flag in Blink, and in nightly builds of WebKit and, latterly, Firefox. By being developed in this way, breaking changes could be safely made to the specification as no one was relying on the experimental implementations in production work.

Another new layout method has emerged over the past few years in a more public and perhaps more painful way. Shipped prefixed in browsers, The flexible box layout module (flexbox) was far too tempting for developers not to use on production sites. Therefore, as changes were made to the specification, we found ourselves with three different flexboxes, and browser implementations that did not match one another in completeness or in the version of specified features they supported.

Owing to the different ways these modules have come into being, when I present on grid layout it is often the very first time someone has heard of the specification. A question I keep being asked is whether CSS grid layout and flexbox are competing layout systems, as though it might be possible to back the loser in a CSS layout competition. The reality, however, is that these two methods will sit together as one system for doing layout on the web, each method playing to certain strengths and serving particular layout tasks.

If there is to be a loser in the battle of the layouts, my hope is that it will be the layout frameworks that tie our design to our markup. They have been a necessary placeholder while we waited for a true web layout system, but I believe that in a few years time we'll be easily able to date a website to circa 2015 by seeing `<div class="row">` or `<div class="col-md-3">` in the markup.

In this article, I'm going to take a look at the common features of our new layout systems, along with a couple of examples which serve to highlight the differences between them.

To see the grid layout examples you will need to enable grid in your browser. The easiest thing to do is to enable the **experimental web platform features** flag in Chrome. Details of current browser support can be found [here](#).

RELATIONSHIP

Items only become flex or grid items if they are a direct child of the element that has `display: flex`, `display: grid` or `display: inline-grid` applied. Those direct children then understand themselves in the context of the complete layout. This makes many things possible. It's the lack of relationship between elements that makes our existing layout methods difficult to use. If we float two columns, left and right, we have no way to tell the shorter column to extend to the height of the taller one. We have expended a lot of effort trying to figure out the best way to make full-height columns work, using techniques that were never really designed for page layout.

At a very simple level, the relationship between elements means that we can easily achieve full-height columns. In flexbox:

See the Pen Flexbox equal height columns by rachelandrew (@rachelandrew) on CodePen.

And in grid layout (requires a CSS grid-supporting browser):

See the Pen Grid equal height columns by rachelandrew (@rachelandrew) on CodePen.

ALIGNMENT

Full-height columns rely on our flex and grid items understanding themselves as part of an overall layout. They also draw on a third new specification: the box alignment module. If vertical centring is a gift you'd like to have under your tree this Christmas, then this is the box you'll want to unwrap first.

The box alignment module takes the alignment and space distribution properties from flexbox and applies them to other layout methods. That includes grid layout, but also other layout methods. Once implemented in browsers, this specification will give us true vertical centring of *all the things*.

Our examples above achieved full-height columns because the default value of `align-items` is `stretch`. The value ensured our columns stretched to the height of the tallest. If we want to use our new vertical centring

abilities on all items, we would set `align-items:center` on the container. To align one flex or grid item, apply the `align-self` property.

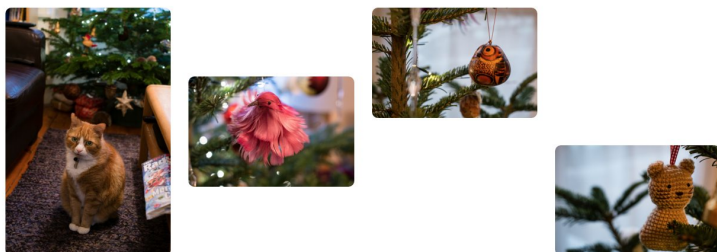
The examples below demonstrate these alignment properties in both grid layout and flexbox. The portrait image of Widget the cat is aligned with the default stretch. The other three images are aligned using different values of `align-self`.

Take a look at an example in flexbox:

See the [Pen Flexbox alignment](#) by rachelandrew (@rachelandrew) on CodePen.

And also in grid layout (requires a CSS grid-supporting browser):

See the [Pen Grid alignment](#) by rachelandrew (@rachelandrew) on CodePen.



15-1. The alignment properties used with CSS grid layout.

FLUID GRIDS

A cornerstone of responsive design is the concept of fluid grids.

“[...]every aspect of the grid—and the elements laid upon it—can be expressed as a proportion relative to its container.”

—Ethan Marcotte, “Fluid Grids”

The method outlined by Marcotte is to divide the target width by the context, then use that value as a percentage value for the width property on our element.

```
h1 {  
  margin-left: 14.575%; /* 144px / 988px = 0.14575 */  
  width: 70.85%; /* 700px / 988px = 0.7085 */  
}
```

In more recent years, we’ve been able to use `calc()` to simplify this (at least, for those of us able to **drop support for Internet Explorer 8**). However, flexbox and grid layout make fluid grids simple.

The most basic of flexbox demos shows this fluidity in action. The `justify-content` property – another property defined in the box alignment module – can be used to create an equal amount of space between or around items. As the available width increases, more space is assigned in proportion.

In this demo, the list items are flex items due to `display: flex` being added to the `ul`. I have given them a maximum width of 250 pixels. Any remaining space is distributed equally between the items as the `justify-content` property has a value of `space-between`.

See the Pen [Flexbox: justify-content](#) by [rachelandrew \(@rachelandrew\)](#) on CodePen.

For true fluid grid-like behaviour, your new flexible friends are `flex-grow` and `flex-shrink`. These properties give us the ability to assign space in proportion.

The flexbox `flex` property is a shorthand for:

- `flex-grow`
- `flex-shrink`
- `flex-basis`

The `flex-basis` property sets the default width for an item. If `flex-grow` is set to 0, then the item will not grow larger than the `flex-basis` value; if `flex-shrink` is 0, the item will not shrink smaller than the `flex-basis` value.

- `flex: 1 1 200px`: a flexible box that can grow and shrink from a 200px basis.
- `flex: 0 0 200px`: a box that will be 200px and cannot grow or shrink.

- `flex: 1 0 200px`: a box that can grow bigger than 200px, but not shrink smaller.

In this example, I have a set of boxes that can all grow and shrink equally from a 100 pixel basis.

See the Pen [Flexbox: flex-grow](#) by [rachelandrew \(@rachelandrew\)](#) on [CodePen](#).

What I would like to happen is for the first element, containing a portrait image, to take up less width than the landscape images, thus keeping it more in proportion. I can do this by changing the `flex-grow` value. By giving all the items a value of 1, they all gain an equal amount of the available space after the 100 pixel basis has been worked out.

If I give them all a value of 3 and the first box a value of 1, the other boxes will be assigned three parts of the available space while box 1 is assigned only one part. You can see what happens in this demo:

See the Pen [Flexbox: flex-grow](#) by [rachelandrew \(@rachelandrew\)](#) on [CodePen](#).

Once you understand `flex-grow`, you should easily be able to grasp how the new *fraction unit* (`fr`, defined in the CSS grid layout specification) works. Like `flex-grow`, this

unit allows us to assign available space in proportion. In this case, we assign the space when defining our track sizes.

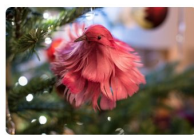
In this demo (which requires a CSS grid-supporting browser), I create a four-column grid using the fraction unit to define my track sizes. The first track is 1fr in width, and the others 2fr.

```
grid-template-columns: 1fr 2fr 2fr 2fr;
```

See the [Pen Grid fraction units](#) by [rachelandrew \(@rachelandrew\)](#) on CodePen.



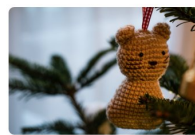
Widget disapproves of Christmas



Widget is annoyed by this bird



This owl is irritating



Widget thinks this bear/cat thing is weird

15-2. The four-track grid.

SEPARATION OF CONCERNS

My younger self petitioned my peers to stop using tables for layout and to move to CSS. One of the rallying cries of that movement was the concept of separating our source and content from how they were displayed. It was something of a failed promise given the tools we had available: the display leaked into the markup with the

need for redundant elements to cope with browser bugs, or visual techniques that just could not be achieved without supporting markup.

Browsers have improved, but even now we can find ourselves compromising the ideal document structure so we can get the layout we want at various breakpoints. In some ways, the situation has returned to tables-for-layout days. Many of the current grid frameworks rely on describing our layout directly in the markup. We add `divs` for rows, and classes to describe the number of desired columns. We nest these constructions of `divs` inside one another.

Here is a snippet from the **Bootstrap** grid examples – **two columns with two nested columns**:

```
<div class="row">
  <div class="col-md-8">
    .col-md-8
    <div class="row">
      <div class="col-md-6">
        .col-md-6
      </div>
      <div class="col-md-6">
        .col-md-6
      </div>
    </div>
  </div>
  <div class="col-md-4">
```

```
        .col-md-4
    </div>
</div>
```

Not a million miles away from something I might have written in 1999.

```
<table>
  <tr>
    <td class="col-md-8">
      .col-md-8
      <table>
        <tr>
          <td class="col-md-6">
            .col-md-6
          </td>
          <td class="col-md-6">
            .col-md-6
          </td>
        </tr>
      </table>
    </td>
    <td class="col-md-4">
      .col-md-4
    </td>
  </tr>
</table>
```

Grid and flexbox layouts *do not need to be described in markup*. The layout description happens entirely in the CSS, meaning that elements can be moved around from within the presentation layer.

Flexbox gives us the ability to reverse the flow of elements, but also to set the order of elements with the `order` property. This is demonstrated here, where Widget the cat is in position 1 in the source, but I have used the `order` property to display him *after* the things that are currently unimpressive to him.

See the Pen [Flexbox: order](#) by rachelandrew (@rachelandrew) on CodePen.

Grid layout takes this a step further. Where flexbox lets us set the order of items in a single dimension, grid layout gives us the ability to position things in two dimensions: both rows and columns. Defined in the CSS, this positioning can be changed at any breakpoint without needing additional markup. Compare the source order with the display order in this example (requires a CSS grid-supporting browser):

See the Pen [Grid positioning in two dimensions](#) by rachelandrew (@rachelandrew) on CodePen.



This owl is irritating



Widget disapproves of Christmas



Widget thinks this bear/cat thing is weird



What is this big cat face doing?



Widget is annoyed by this bird

15-3. Laying out our items in two dimensions using grid layout.

As these demos show, a straightforward way to decide if you should use grid layout or flexbox is whether you want to position items in one dimension or two. If two, you want grid layout.

A note on accessibility and reordering

The issues arising from this powerful ability to change the way items are ordered visually from how they appear in the source have been the subject of much discussion. The current flexbox editor's draft states

“Authors *must* use order only for visual, not logical, reordering of content. Style sheets that use order to perform logical reordering are non-conforming.”

—CSS Flexible Box Layout Module Level 1,
Editor’s Draft (3 December 2015)

This is to ensure that non-visual user agents (a screen reader, for example) can rely on the document source order as being correct. Take care when reordering that you do so from the basis of a sound document that makes sense in terms of source order. Avoid using visual order to convey meaning.

AUTOMATIC CONTENT PLACEMENT WITH RULES

Having control over the order of items, or placing items on a predefined grid, is nice. However, we can often do that already with one method or another and we have frameworks and tools to help us. Tools such as **Susy** mean we can even get away from stuffing our markup full of grid classes. However, our new layout methods give us some interesting new possibilities.

Something that is useful to be able to do when dealing with content coming out of a CMS or being pulled from some other source, is to define a bunch of rules and then say, “Display this content, using these rules.”

As an example of this, I will leave you with a Christmas poem displayed in a document alongside Widget the cat and some of the decorations that are bringing him no Christmas cheer whatsoever.

The poem is displayed first in the source as a set of paragraphs. I've added a class identifying each of the four paragraphs but they are displayed in the source as one text. Below that are all my images, some landscape and some portrait; I've added a class of `landscape` to the landscape ones.

The mobile-first grid is a single column and I use line-based placement to explicitly position my poem paragraphs. The grid layout auto-placement rules then take over and place the images into the empty cells left in the grid.

At wider screen widths, I declare a four-track grid, and position my poem around the grid, keeping it in a readable order.

I also add rules to my `landscape` class, stating that these items should span two tracks. Once again the grid layout auto-placement rules position the rest of my images without my needing to position them. You will see that grid layout takes items out of source order to fill gaps in the grid. It does this because I have set the property `grid-auto-flow` to `dense`. The default is `sparse` meaning that grid will not attempt this backfilling behaviour.

Grid, Flexbox, Box Alignment: Our New System for Layout

Take a look and play around with the full demo (requires a CSS grid layout-supporting browser):

See the [Pen Grid auto-flow with rules](#) by [rachelandrew \(@rachelandrew\)](#) on CodePen.



'Twas the night before Christmas, when all thro' the house Not a creature was stirring, not even a mouse;
The stockings were hung by the chimney with care, In hopes that St. Nicholas soon would be there;
The children were nestled all snug in their beds, While visions of sugar plums danc'd in their heads,
And Mama in her 'kerchief, and I in my cap, Had just settled our brains for a long winter's nap —
When out on the lawn there arose such a clatter, I sprang from the bed to see what was the matter.
Away to the window I flew like a flash, Tore open the shutters, and threw up the sash.
The moon on the breast of the new fallen snow, Gave the luster of mid-day to objects below;



When, what to my wondering eyes should appear, But a miniature sleigh, and eight tiny reindeer,
With a little old driver, so lively and quick, I knew in a moment it must be St. Nick.
More rapid than eagles his coursers they came, And he whistled, and shouted, and call'd them by name:
"Now! Dasher, now! Dancer, now! Prancer and Vixen, "On! Comet, on! Cupid, on! Dunder and Blitzen;
"To the top of the porch! To the top of the wall! "Now dash away! Dash away! Dash away all!"
As dry leaves that before the wild hurricane fly, When they meet with an obstacle, mount to the sky;
So up to the house-top the coursers they flew, With the sleigh full of toys — and St. Nicholas too:
And then in a twinkling, I heard on the roof The prancing and pawing of each little hoof.

As I drew in my head, and was turning around, Down the chimney St. Nicholas came with a bound:
He was dress'd all in fur, from his head to his foot, And his clothes were all tarnish'd with ashes and soot;
A bundle of toys was flung on his back, And he look'd like a peddler just opening his pack:
His eyes — how they twinkled! His dimples: how merry, His cheeks were like roses, his nose like a cherry;
His droll little mouth was drawn up like a bow, And the beard of his chin was as white as the snow;
The stump of a pipe he held tight in his teeth, And the smoke it encircled his head like a wreath.
He had a broad face, and a little round belly That shook when he laugh'd, like a bowl full of jelly:



He was chubby and plump, a right jolly old elf, And I laugh'd when I saw him in spite of myself;
A wink of his eye and a twist of his head Soon gave me to know I had nothing to dread.
He spoke not a word, but went straight to his work, And fill'd all the stockings; then turn'd with a jerk,
And laying his finger aside of his nose And giving a nod, up the chimney he rose.
He sprang to his sleigh, to his team gave a whistle, And away they all flew, like the down of a thistle:
But I heard him exclaim, ere he drove out of sight — Happy Christmas to all, and to all a good night.



15-4. The final automatic placement example.

MY WISH FOR 2016

I really hope that in 2016, we will see CSS grid layout finally emerge from behind browser flags, so that we can start to use these features in production — that we can start to move away from using the wrong tools for the job.

However, I also hope that we'll see developers fully embracing these tools as the new system that they are. I want to see people exploring the possibilities they give us, rather than trying to get them to behave like the grid systems of 2015. As you discover these new modules, treat them as the new paradigm that they are, get creative with them. And, as you find the edges of possibility with them, take that feedback to the CSS Working Group. Help improve the layout systems that will shape the look of the future web.

SOME FURTHER READING

- I maintain a site of grid layout examples and resources at [Grid by Example](#).
- The three CSS specifications I've discussed can be found as editor's drafts: [CSS grid](#), [flexbox](#), [box alignment](#).
- I wrote about the last three years of my interest in CSS grid layout, which gives something of a history of the specification.

- More examples of box alignment and grid layout.
- My presentation at Fronteers earlier this year, in which I explain more about these concepts.

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, and is a regular columnist for **A List Apart**.

Grid, Flexbox, Box Alignment: Our New System for Layout

She curates a popular email newsletter on CSS Layout, and will be launching a CSS Layout online workshop in early 2016.

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.

16. Beyond the Style Guide

Paul Lloyd

24ways.org/201516

Much like baking a Christmas cake, designing for the web involves creating an experience in layers. Starting with a solid base that provides the core experience (the fruit cake), we can add further layers, each adding refinement (the marzipan) and delight (the icing).

Don't worry, this isn't a misplaced **cake recipe**, but an evaluation of modular design and the role **style guides** can play in acknowledging these different concerns, be they presentational or programmatic.

THE AUTEUR'S STYLE GUIDE

Although trained as a graphic designer, it was only when I encountered the immediacy of the web that I felt truly empowered as a designer. Given a desire to control every aspect of the resulting experience, I slowly **adopted the role of an auteur**, exploring every part of the web stack:

front-end to back-end, and everything in between. A few years ago, I dreaded using the command line. Today, the terminal is a permanent feature in my Dock.

In straddling the realms of graphic design and programming, it's the point at which they meet that I find most fascinating, with each discipline valuing the creation of effective systems, **be they for communication or code efficiency**. Front-end style guides live at this intersection, demonstrating both the modularity of code and the application of visual design.

PAINTING BY NUMBERS

In our rush to build modular systems, design frameworks have grown in popularity. While enabling quick assembly, these come at the cost of originality and creative expression – perhaps one reason why we're seeing the homogenisation of web design.

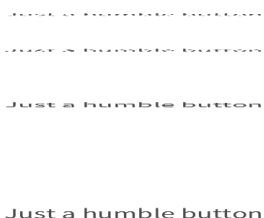
In editorial design, layouts should accentuate content and present it in an engaging manner. Yet on the web we see a practice that seeks templated predictability. In 'Design Machines' Travis Gertz argued that (emphasis added):

Design systems still feel like a novelty in screen-based design. We nerd out over grid systems and modular scales and obsess over style guides and pattern libraries. We're pretty good at using them to build repeatable components and site-wide standards, but that's sort of where it ends. [...] But to stop there is to ignore the true purpose and potential of a design system.

Unless we consider how interface patterns fully embrace the design systems they should be built upon, style guides may exacerbate this paint-by-numbers approach, encouraging conformance and suppressing creativity.

ANATOMY OF A BUTTON

Let's take a look at that most canonical of components, the button, and consider what we might wish to document and demonstrate in a style guide.



16-1. The different layers of our button component.

Content

The most variable aspect of any component. Content guidelines will exert the most influence here, dictating things like tone of voice (whether we should we use stiff, formal language like ‘Submit form’, or adopt a more friendly tone, perhaps ‘Send us your message’) and appropriate language. For an internationalised interface, this may also impact word length and text direction or orientation.

Structure

HTML provides a limited vocabulary which we can use to structure content and add meaning. For interactive elements, the choice of element can also affect its behaviour, such as whether a button submits form data or links to another page:

```
<button type="submit">Button text</button>
```

```
<a href="/index.html">Button text</a>
```

Note: One of the reasons I prefer to use `<button>` instead of `<input type="button">`, besides allowing the inclusion of content other than text, is that it has a markup structure similar to links, therefore keeping implementation differences to a minimum.

We should also think about each component within the broader scope of our particular product. For this we can **employ a further vocabulary**, which can be expressed by adding values to the `class` attribute. For a newspaper, we might use names like *lede*, *standfirst* and *headline*, while a social media application might see us reach for words like *stream*, *action* or *avatar*.

Presentation

The appearance of a component can never be considered in isolation. Informed by its relationship to other elements, style guides may document different stylistic variations of a component, even if the underlying function remains unchanged: primary and secondary button styles, for example.

Behaviour

A component can exhibit **various states**: blank, loading, partial, error and ideal, and a style guide should reflect that. Our button component is relatively simple, yet even here we need to consider hover, focused, active and disabled states.

Transcending layers

This overview reinforces **Ethan's note** from earlier in this series:

I've found that thinking about my design as existing in broad experience tiers – in layers – is one of the best ways of designing for the modern web.

While it's tempting to describe a component as series of layers, certain aspects will transcend several of these. The accessibility of a component, for example, may influence the choice of language, the legibility of text, colour contrast and which affordances are provided in different states.

VISUAL DESIGN LANGUAGE: DOCUMENTING THE MISSING PIECE

Even given this small, self-contained component, we can see several concerns at play, and in reviewing our button it seems we have most things covered. However, a few questions remain unanswered. Why does it have a blue background? Why are the borders 2px thick, with a radius of 4px? Why are we using that font, at that size and with that weight?

These questions can be answered by our visual design language. More than a set of type choices and colour palettes, a design language can dictate common measures, ratios and the resulting grid(s) these influence. Ideally governed by a set of broader design principles, it can also

inform an illustration style, the type of photography sourced or commissioned, and the behaviour of any animations.

Whereas a style guide ensures conformity, having it underpinned by an effective design language will allow for flexibility; only by knowing the rules can you know how to break them!

Pairings + Styles

To support both more contemporary and more traditional web design aesthetics, this font system offers recommended font pairings. Each pairing includes web hierarchy guidance on font family, weight, size, and spacing which express either more modern or more classical type design.

Note: Some pairings require more font weights than others. While this allows more typographic expression, the use of more than four font weights will have a negative impact on page load performance. Find the balance that works for your product.

Default: Merriweather headings, Source Sans Pro body (lite)

A simple serif and sans serif combination designed to communicate warmth and credibility. Strong Merriweather heading weights offer clear information hierarchy and when paired with Source Sans Pro's easy-to-read body text, create a clean and professional feel.

This pairing is included in our design standards.

Recommended applications: digital services that feature forms; basic and text heavy sites.

Font weights included in this package:

1. Merriweather, Bold 700
2. Source Sans Pro, Regular 400
3. Source Sans Pro, Bold 700
4. Source Sans Pro, Italic 400

WEB HIERARCHY

Display	font-family: 'Merriweather' font-weight: 700 font-size: 52px line-height: 1.3em/68px
Heading 1	font-family: 'Merriweather' font-weight: 700 font-size: 48px line-height: 1.3em/52px

PAGE PERFORMANCE

FAST

Ideal number of fonts
Will allow for optimal
page load performanc

EXAMPLE

[EPA eManifest](#)
(screenshot of non-pa

Merriweather headings, Source Sans Pro Body (robust)

A variation of the previous font pairing, expanded to include an additional Merriweather weight. The slimmer Merriweather headings creates an elegance that compliments weights and allows you to intentionally move users' attention around a page.

Recommended applications: text heavy sites and more visual promotional sites.

Font weights included in this package:

1. Merriweather, Bold 700
2. Merriweather, Light 300
3. Source Sans Pro, Regular 400
4. Source Sans Pro, Bold 700
5. Source Sans Pro, Italic 400

PAGE PERFORMANCE

MEDIUM

Exceeds ideal number
of fonts by one. May
negatively impact pa
ge load performance.

EXAMPLE

[U.S. Web Design
Standards homepage](#)

WEB HIERARCHY

Display 1	font-family: 'Merriweather' font-weight: 700 font-size: 52px line-height: 1.3em/68px
Display 2	font-family: 'Merriweather' font-weight: 300 font-size: 48px line-height: 1.3em/52px
Heading 1	font-family: 'Merriweather' font-weight: 700 font-size: 48px line-height: 1.3em/52px
Heading 2	font-family: 'Merriweather' font-weight: 700 font-size: 30px line-height: 1.3em/39px
Heading 3	font-family: 'Merriweather' font-weight: 700 font-size: 20px line-height: 1.3em/26px

16-2. Type pairings in the US Web Design Standards guide.

For a style guide to thoroughly articulate a visual design system, the spectrum of choices it allows for should be acknowledged. A fantastic example of this can be found in

the US Web Design Standards. By virtue of being a set of standards designed to apply to a number of different sites, this guide offers a range of type pairings (that take into account performance considerations) and provides primary, secondary and tertiary palette relationships, with shades and tones thereof:

Primary colors

This palette's primary colors are blue, gray, and white. Blue is commonly associated with trust, confidence, and sincerity; it is also used to represent calmness and responsibility.



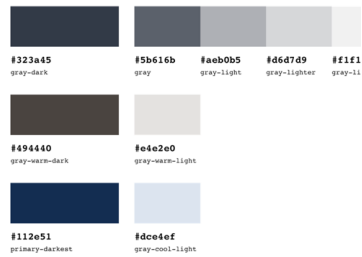
Secondary colors

These are accent colors to provide additional lightness and style to pages looking for a more modern flair. These colors should be used to highlight important features on a page, such as buttons, or for visual style elements, such as illustrations. They should be used sparingly and never draw the eye to more than one piece of information at a time.



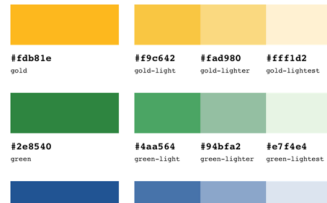
Background colors

These colors are used largely for background blocks and large content areas. When alternating between tones, be sure to use enough contrast between adjacent colors.



Tertiary colors

These colors are used primarily for content-specific needs, such as alerts and illustrations. They should never overpower the primary colors.



16-3. Colour palettes in the US Web Design Standards guide.

A VISUAL LANGUAGE IN CODE FORM

Properly documenting our design language in a style guide is a good start, yet even better if it can be expressed in code. This is where CSS preprocessors become a powerful ally.

In Sass, methods like mixins and maps can help us represent relationships between values. Variables (and **CSS variables**) extend the vocabulary provided natively by CSS, meaning we can describe patterns in terms of our own visual language. These tools effectively become an interface to our design system. Furthermore, they help maintain a separation of concerns, with visual presentation remaining where it should be: in our style sheets.

Take this simple example, an article summary on a website counting down the best Christmas movies:

12 / Scrooged (1988)

It's unlikely that Bill Murray could ever have got through his career without playing a version of Scrooge. His deadpan delivery was made for this comic take on Dickens' festive moral tale, in which his TV exec oversees a broadcast of 'A Christmas Carol'.

Director: Richard Donner

Stars: Bill Murray, Buddy Hackett, Karen Allen

16-4. The design for our simple component example.

Our markup is as follows, using appropriate semantic HTML elements and incorporating the vocabulary from our collection of design patterns (expressed using the BEM methodology):

```
<article class="summary">
  <h1 class="summary__title">
    <a href="scrooged.html">
      <span class="summary__position">12</span>
      Scrooged (1988)
    </a>
  </h1>

  <div class="summary__body">
    <p>It's unlikely that Bill Murray could ever
have got through his career without playing a version of
Scrooge...</p>
  </div>

  <footer class="summary__meta">
```

```
        <strong>Director:</strong> Richard Donner<br/>
        <strong>Stars:</strong> Bill Murray, Buddy
Hackett, Karen Allen
    </footer>
</article>
```

We can then describe the presentation of this HTML by using **Sass maps to define our palettes**, mixins to include predefined font metrics, and variables to recall common measurements:

```
.summary {
    margin-bottom: ($baseline * 4)
}

.summary__title {
    @include font-family(display-serif);
    @include font-size(title);
    color: palette(neutral, dark);
    margin-bottom: ($baseline * 4);
    border-top: $rule-height solid palette(primary,
purple);
    padding-top: ($baseline * 2);
}

.summary__position {
    @include font-family(display-sans, 300);
    color: palette(neutral, mid);
}

.summary__body {
    @include font-family(text-serif);
    @include font-size(body);
```

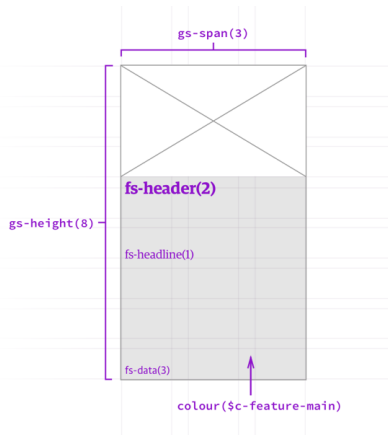
```

margin-bottom: ($baseline * 2);
}

.summary__meta {
  @include font-family(text-sans);
  @include font-size(caption);
}

```

Of course, this is a simplistic example for the purposes of demonstration. However, such thinking was employed at a much larger scale at the Guardian. Using a set of Sass components, complex patterns could be described using a language familiar to everyone on the product team, be they a designer, developer or product owner:



16-5. The design of a component on the Guardian website, described in terms of its Sass-powered design system.

UNLOCKING POSSIBILITY

Alongside tools like preprocessors, newer CSS layout modules like flexbox and grid layout mean the friction we've long been accustomed to when creating layouts on the web is no longer present, and **the full separation of presentation from markup is now possible**. Now is the perfect time for graphic designers to advocate design systems that these developments empower, and ensure they're fully represented in both documentation and code. That way, together, we can build systems that allow for greater visual expression. After all, there's more than one way to bake a Christmas cake.

ABOUT THE AUTHOR



Paul Robert Lloyd is an independent **designer, writer and speaker** who helps organisations like the Guardian, UNICEF and Mozilla create purposeful digital products.

If not indulging in his **love of train travel**, Paul can be found in a coffee shop, either writing for his **blog**, or blathering on **Twitter**.

17. The Accessibility Mindset

Eric Eggert

24ways.org/201517

Accessibility is often characterized as additional work, hard to learn and only affecting a small number of people. Those myths have no logical foundation and often stem from outdated information or misconceptions.

Indeed, it is an additional skill set to acquire, quite like learning new JavaScript frameworks, CSS layout techniques or new HTML elements. But it isn't particularly harder to learn than those other skills.

A World Health Organization (WHO) report on disabilities states that,

[i]ncluding children, over a billion people (or about 15% of the world's population) were estimated to be living with disability.

Being disabled is not as unusual as one might think. Due to chronic health conditions and older people having a higher risk of disability, we are also currently paving the cowpath to an internet that we can still use in the future.

Accessibility has a very close relationship with usability, and advancements in accessibility often yield improvements in the usability of a website. Websites are also more adaptable to users' needs when they are built in an accessible fashion.

BEYOND THE BARE MINIMUM

In the time of table layouts, web developers could create code that passed validation rules but didn't adhere to the underlying semantic HTML model. We later developed best practices, like using lists for navigation, and with HTML5 we started to wrap those lists in nav elements. Working with accessibility standards is similar. The **Web Content Accessibility Guidelines (WCAG) 2.0** can inform your decision to make websites accessible and can be used to test that you met the success criteria. What it can't do is measure *how well* you met them.

W3C developed a long list of techniques that can be used to make your website accessible, but you might find yourself in a situation where you need to adapt those techniques to be the most usable solution for your particular problem.

The checkbox below is implemented in an accessible way: The input element has an id and the label associated with the checkbox refers to the input using the for attribute. The hover area is shown with a yellow background and a black dotted border:

Open video

The label is clickable and the checkbox has an accessible description. Job done, right? Not really. Take a look at the space between the label and the checkbox:

Open video

The gutter is created using a right margin which pushes the label to the right. Users would certainly expect this space to be clickable as well. The simple solution is to wrap the label around the checkbox and the text:

Open video

You can also set the label to `display: block;` to further increase the clickable area:

Open video

And while we're at it, users might expect the whole box to be clickable anyway. Let's apply the CSS that was on a wrapping `div` element to the `label` directly:

Open video

The result enhances the usability of your form element tremendously for people with lower dexterity, using a voice mouse, or using touch interfaces. And we only used basic HTML and CSS techniques; no JavaScript was added and not one extra line of CSS.

```
<form action="#">
  <label for="uniquecheckboxid">
    <input type="checkbox" name="checkbox"
id="uniquecheckboxid" />
    Checkbox 4
  </label>
</form>
```

Button Example

The button below looks like a typical edit button: a pencil icon on a **real** button element. But if you are using a screen reader or a braille keyboard, the button is just read as “button” without any indication of what this button is for.

Open video

17-1. A screen reader announcing a button. Contains audio.

The code snippet shows why the button is not properly announced:

```
<button>  
  <span class="icon icon-pencil"></span>  
</button>
```

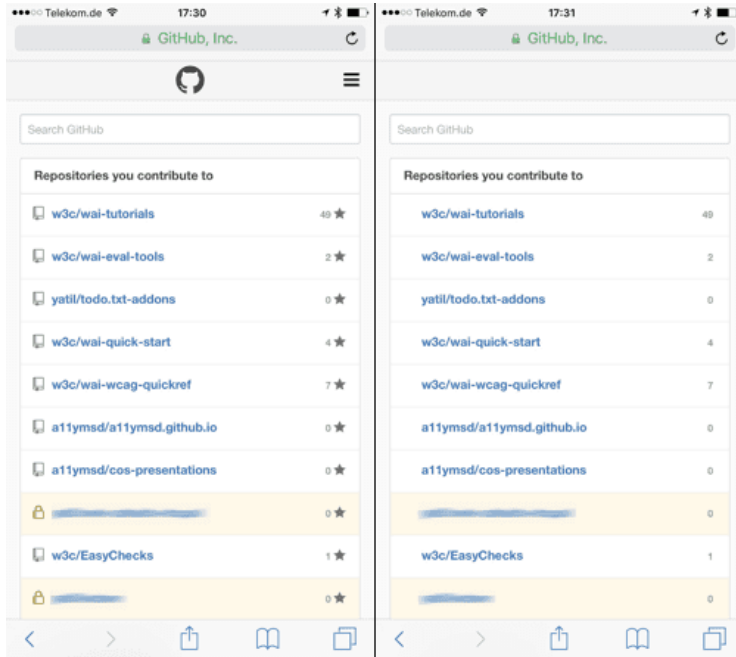
An icon font is used to display the icon and no text alternative is given. A possible solution to this problem is to use the `title` or `aria-label` attributes, which solves the alternative text use case for screen reader users:

Open video

17-2. A screen reader announcing a button with a title.

However, screen readers are not the only way people with and without disabilities interact with websites. For example, users can reset or change font families and sizes at will. This helps many users make websites easier to read, including **people with dyslexia**. Your icon font might be replaced by a font that doesn't include the glyphs that are icons. Additionally, the icon font may not load for users on slow connections, like on mobile phones inside

trains, or because users decided to block external fonts altogether. The following screenshots show the mobile GitHub view with and without external fonts:



17-3. The mobile GitHub view with and without external fonts.

Even if the `title/aria-label` approach was used, the lack of visual labels is a barrier for most people under those circumstances. One way to tackle this is using the old-fashioned `img` element with an appropriate `alt` attribute, but surprisingly not every browser displays the alternative text visually when the image doesn't load.

```
<button>  
    
</button>
```

Providing always visible text is an alternative that can work well if you have the space. It also helps users understand the meaning of the icons.

```
<button>  
  <span class="icon icon-pencil"></span> Edit  
</button>
```

This also reads just fine in screen readers:

Open video

17-4. A screen reader announcing the revised button.

Clever usability enhancements don't stop at a technical implementation level. Take the BBC iPlayer pages as an example: when a user navigates the "captioned videos" or "audio description" categories and clicks on one of the videos, captions or audio descriptions are automatically switched on. Small things like this enhance the usability and don't need a lot of engineering resources. It is more about connecting the usability dots for people with disabilities. [Read more about the BBC iPlayer accessibility case study.](#)

MORE INFORMATION

W3C has created several documents that make it easier to get the gist of what web accessibility is and how it can benefit everyone. You can find out “**How People with Disabilities Use the Web**”, there are “**Tips for Getting Started**” for developers, designers and content writers. And for the more seasoned developer there is a set of **tutorials** on web accessibility, including information on crafting accessible forms and how to use images in an accessible way.

CONCLUSION

You can only produce a web project with long-lasting accessibility if accessibility is not an afterthought. Your organization, your division, your team need to think about accessibility as something that is the foundation of your website or project. It needs to be at the same level as performance, code quality and design, and it needs the same attention. Users often don’t notice when those fundamental aspects of good website design and development are done right. But they’ll always know when they are implemented poorly.

If you take all this into consideration, you can create accessibility solutions based on the available data and bring accessibility to people who didn’t know they’d need it:

Open video

In this video from the latest Apple keynote, the Apple TV is operated by voice input through a remote. When the user asks “What did she say?” the video jumps back fifteen seconds and captions are switched on for a brief time. All three, the remote, voice input and captions have their roots in assisting people with disabilities. Now they benefit everyone.

ABOUT THE AUTHOR



Eric Eggert is an accessibility advocate living in Essen, Germany, currently working for the W3C's **Web Accessibility Initiative**, helping the **WCAG** and **EO** Working Groups to make accessibility information easier to find and use. He is co-editor of the **W3C Web Accessibility Tutorials**. He also co-owns a small agency for web development and consulting, called **outline**. When he is not working or giving talks about web topics, he enjoys a game of pool, a festival, or a movie.

18. Cooking Up Effective Technical Writing

Sally Jenkinson

24ways.org/201518

Merry Christmas! May your preparations for this festive season of gluttony be shaping up beautifully. By the time you read this I hope you will have ordered your turkey, eaten twice your weight in Roses/Quality Street (let's not get into that argument), and your Christmas cake has been baked and is now quietly absorbing regular doses of alcohol.

Some of you may be reading this and scoffing "Of course! I've also made three batches of mince pies, a seasonal chutney and enough gingerbread men to feed the whole street!" while others may be laughing "Bake? Oh no, I can't cook to save my life."

For beginners, recipes are the step-by-step instructions that hand-hold us through the cooking process, but even as a seasoned expert you're likely to refer to a recipe at some point. Recipes tell us what we need, what to do with it, in what order, and what the outcome will be. It's the documentation behind our ideas, and allows us to take the

blueprint for a tasty morsel and to share it with others so they can recreate it. In fact, this is a little like the open source documentation and tutorials that we put out there, similarly aiming to guide other developers through our creations.

THE ‘JUST’IFICATION OF DOCUMENTATION

Lately it feels like we’re starting to consider the importance of our words, and the impact they can have on others. Brad Frost warned us of the dangers of “Just” when it comes to offering up solutions to queries:

“Just use this software/platform/toolkit/methodology...”

“Just” makes me feel like an idiot. “Just” presumes I come from a specific background, studied certain courses in university, am fluent in certain technologies, and have read all the right books, articles, and resources. “Just” is a dangerous word.

“Just” by Brad Frost

I can really empathise with these sentiments. My relationship with code started out as many good web tales do, with good old HTML, CSS and JavaScript. University years involved some time with Perl, PHP, Java and C. In my first job I worked primarily with ColdFusion, a bit of

ActionScript, some classic ASP and pinch of Java. I'd do a bit of PHP outside work every now and again. .NET came in, but we never really got on, and eventually I started learning some Ruby, Python and Node. It was a broad set of learnings, and I enjoyed the similarities and differences that came with new languages. I don't develop day in, day out any more, and my interests and work have evolved over the years, away from full-time development and more into architecture and strategy. But I still make things, and I still enjoy learning.

I have often found myself bemoaning the lack of tutorials or courses that cater for the middle level – someone who may be learning a new language, but who has enough programming experience under their belt to not need to revise the concepts of how loops or objects work, and is perfectly adept at googling the syntax for getting a substring. I don't want snippets out of context; I want an understanding of architectural principles, of the strengths and weaknesses, of the type of applications that work well with the language.

I'm caught in the place between snoozing off when 'Using the Instagram API with Ruby' hand-holds me through what REST is, and feeling like I'm stupid and need to go back to dev school when I can't get my environment and dependencies set up, let alone work out how I'm meant to get any code to run.

It's seems I'm not alone with this – Erin McKean seems to have been here too:

“Some tutorials (especially coding tutorials) like to begin things in media res. Great for a sense of dramatic action, bad for getting to “Step 1” without tears. It can be really discouraging to fire up a fresh terminal window only to be confronted by error message after error message because there were obligatory steps 0.1.0 through 0.9.9 that you didn’t even know about.”

“Tips for Learning What You Don’t Know You Don’t Know”
by Erin McKean

I’m sure you’ve been here too. Many tutorials suffer badly from the fabled ‘how to draw an owl’-itis.

How to draw an owl

1.



2.



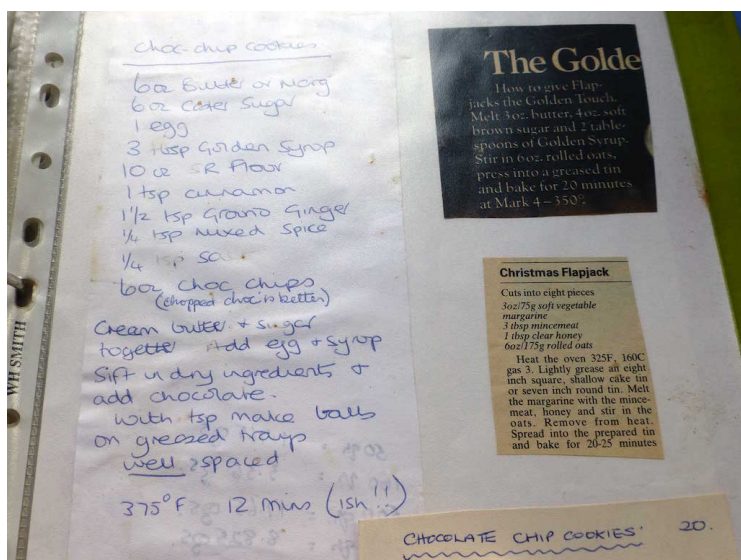
1. Draw some circles

2. Draw the rest of the owl

It's the kind of feeling you can easily get when sifting through recipes as well as with code. Far from being the simple instructions that let us *just* follow along, they too can be a minefield. Fall in too low and you may be skipping over an explanation of what simmering is, or set your sights too high and you may get stuck at the point where you're trying to sous vide a steak using your bathtub and a Ziploc bag.

DON'T BE A TURKEY, USE YOUR LOAF!

My mum is a great cook in my eyes (aren't all mums?). I love her handcrafted collection of gathered recipes from over the years, including the one below, which is a great example of how something may make complete sense to the writer, but could be impermeable to a reader.



Depending on your level of baking knowledge, you may ask: “What’s SR flour? What’s a tsp? Should I use salted or unsalted butter? Do I use sticks of cinnamon or ground? Why is chopped chocolate better? How do I cream things? How big should the balls be? How well is “well spaced”? How much leeway do I have for “(ish!!)”? Does the “20” on the other cookie note mean I’ll end up with twenty?” At

any point, making a wrong call could lead to rubbish cookies, and lead to someone heading down the path of an “I can’t cook” mentality.

You may be able to cook (or follow recipes), but you may not understand the local terms for ingredients, may not be able to acquire something and need to know what kind of substitutes you can use, or may need to actually do some prep before you jump into the main bit.

However, if we look at *good* examples of recipes, I think there’s a lot we can apply when it comes to technical writing on the web. I’ve written before about the benefit of breaking documentation into small, reusable parts, and this will help us, but we can also take it a bit further. Here are my five top tips for better technical writing.

1. STRUCTURE AND STANDARDISE YOUR INFORMATION

Think of the structure of a recipe. We very often have some common elements and they usually follow roughly the same format. We have standards and conventions that allow us to understand very quickly what a recipe is and how it should be used.

Title	
Ingredients	End goal
	Prep
	Instructions
Variations	

Great recipes help their chefs know what they need to get ready in advance, both in terms of buying ingredients and putting together their kit. They then talk through the process, using appropriate language, and without making assumptions that the person can fill in any gaps for themselves; they explain why things are done the way they are. The *best* recipes may also suggest how you can take what you've done and put your own spin on it. For instance, a good recipe for the simple act of boiling an egg will explain cooking time in relation to your preference for yolk gooiness. There are also different flavour combinations to try, accompaniments, or presentation suggestions.

By breaking down your technical writing into similar sections, you can help your audience understand the elements they'll be working with, what they need to do once they have these, and how they can move on from your self-contained illustration.

Title	Ensure your title is suitably descriptive and representative of the result. <i>Getting Started with Python</i> perhaps isn't as helpful as <i>Learn Python: General Syntax and Basics</i> .
Result	<p>Many recipes include a couple of lines as an overview of what you'll end up with, and many include a photo of the finished dish. With our technical writing we can do the same:</p> <p>"In this tutorial we're going to learn how to set up our development environment, and we'll then undertake some exercises to explore the general syntax, finishing by building a mini calculator."</p>
Ingredients	<p>What are the components we'll be working with, whether in terms of versions, environment, languages or the software packages and libraries you'll need along the way? Listing these up front gives the reader a great summary of the things they'll be using, and any gotchas.</p> <p>Being able to provide a small amount of supporting information will also help less experienced users. Ideally, explain briefly what things are and why we're using it.</p>

Prep	<p>As we heard from Erin above, not fully understanding the prep needed can be a huge source of frustration. Attempting to run a code snippet without context will often lead to failure when the prerequisites and process aren't clear. Be sure to include information around any environment set-up, installation or config you'll need to have done before you start.</p> <p>Stu Robson's Simple Sass documentation aims to do this before getting into specifics, although ideally this would also include setting up Sass itself.</p>
Instructions	<p>The body of the tutorial itself is the whole point of our writing. The next four tips will hopefully make your tutorial much more successful.</p>
Variations	<p>Like our ingredients section, as important as explaining why we're using something in this context is, it's also great to explain alternatives that could be used instead, and the impact of doing so.</p>

Perhaps go a step further, explaining ways that people can change what you have done in your tutorial/readme for use in different situations, or to provide further reading around next steps. What happens if they want to change your static array of demo data to use JSON, for instance? By giving some thought to follow-up questions, you can better support your readers.

While not in a separate section, the source code for **GreenSock's GSAP JS basics** explains:

“We'll use a `window.onload` for simplicity, but typically it is best to use either jQuery's `$(document).ready()` or `$(window).load()` or cross-browser event listeners so that you're not limited to one.”

Keep in mind to both:

- Explain what variations are possible.
- Explain why certain options may be more desirable than others in different situations.

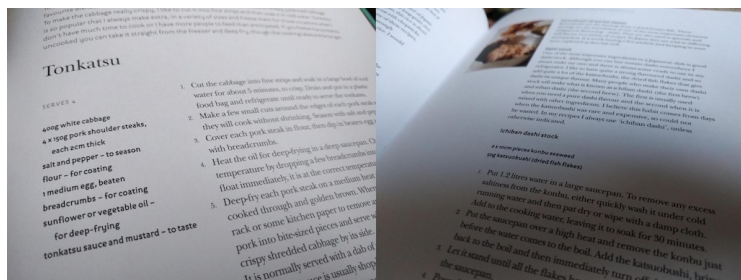
2. SMALL, REUSABLE COMPONENTS

Reusable components are for life, not just for Christmas, and they're certainly not just for development. If you start to apply the structure above to your writing, you're probably going to keep coming across the same elements: "Do I really have to explain how to install Sass and Node.js *again*, Sally?" The danger with more clarity is that our writing becomes bloated and overly convoluted for advanced readers, those who don't need to be told how to beat an egg for the hundredth time.

Instead, by making our writing reusable and modular, and by creating smaller, central resources, we can provide context and extra detail where needed without diluting our core message. These could be references we create, or those already created well by others.



This recipe for katsudon makes use of this concept. Rather than explaining how to make tonkatsu or dashi stock, these each have their own page. Once familiar, more advanced readers will likely skip over the instructions for the component parts.



3. PROVIDE CONTEXT TO AID ACCESSIBILITY

Here I'm talking about accessibility in the broadest sense. Small, isolated snippets can be frustrating to those who don't fully understand the wider context of how our examples work.

Showing an exciting standalone JavaScript function is great, but giving someone the full picture of how and when this is called, and how it should be included in relation to other HTML and CSS is even better. Giving your readers the ability to view a big picture version, and ideally the ability to download a full version of the source, will help to reduce some of the frustrations of trying to get your component to work in their set-up.

4. BE YOUR OWN TECH EDITOR

A good editor can be invaluable to your work, and wherever possible I'd recommend that you try to get a neutral party to read over your writing. This may not always be possible, though, and you may need to rely on yourself to cast a critical eye over your work.

There are many tips out there around general editing, including printing out your work onto paper, or changing the font size: both will force your eyes to review it in a new light. Beyond this, I'd like to encourage you to think about the following:

- Explain what things are. For example, instead of referencing Grunt, in the first instance perhaps reference “Grunt (a JavaScript task runner that minimises repetitive activities through automation).”
- Explain how you get things, even if this is a link to official installers and documentation. Don’t leave your readers having to search.
- Why are you using this approach/technology over other options?
- What happens if I use something else? What depends on this?
- Avoid exclusionary lingo or acronyms.

Airbnb’s JavaScript Style Guide includes useful pointers around their reasoning:

Use computed property names when creating objects with dynamic property names.

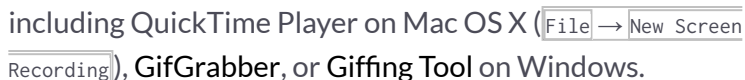
Why? They allow you to define all the properties of an object in one place.

The language we use often makes assumptions, as we saw with “just”. An article titled “ES6 for Beginners” is hugely ambiguous: is this truly for beginner coders, or actually for people who have a good pre-existing understanding of JavaScript but are new to these features? Review your writing with different types of readers in mind. How might you confuse or mislead them? How can you better answer their questions?

This doesn't necessarily mean supporting everyone – your audience may need to have advanced skills – but even if you're providing low-level, deep-dive, reference material, trying not to make assumptions or take shortcuts will hopefully lead to better, clearer writing.

5. A PICTURE IS WORTH A THOUSAND WORDS...

...or even better: use a thousand pictures, stitched together into a quick video or animated GIF. People learn in different ways. Just as recipes often provide visual references or a video to work along with, providing your technical information with alternative demonstrations can really help get your point across. Your audience will be able to see exactly what you're doing, what they should expect as interaction responses, and what the process looks like at different points.

There are many, many options for recording your screen, including QuickTime Player on Mac OS X () **Recording**), GifGrabber, or Giffing Tool on Windows.

Paul Swain, a UX designer, uses GIFs to provide additional context within his documentation, improving communication:

“My colleagues (from across the organisation) love animated GIFs. Any time an interaction is referenced, it’s accompanied by a GIF and a shared understanding of what’s being designed. The humble GIF is worth so much more than a thousand words; and it’s great for cats.”

Paul Swain



Next time you’re cooking up some instructions for readers, think back to what we can learn from recipes to help make your writing as accessible as possible. Use structure, provide reusable bitesize morsels, give some context, edit wisely, and don’t scrimp on the GIFs. And above all, have a great Christmas!

ABOUT THE AUTHOR



Sally Jenkinson is a consultant and digital solutions architect based in the UK, who, through her company Records Sound the Same, helps businesses from big to small with their discovery, requirements, and strategic digital decisions.

Central to this are Sally's views of responsibly using technology to enhance experiences, improving older systems and processes through transformation work, and talking about technical things in a way that isn't scary or boring to her clients. She has

worked with people including Inghams, Nokia, Macmillan Cancer Support, and Electronic Arts, and is also a speaker, an author, and overenthusiastic jasmine tea drinker.

You can find out more about Sally's work at recordssoundthesame.com, and she tweets as [@sjenkinson](https://twitter.com/sjenkinson) when she's not got her head stuck in a comic book or her hands wrapped around an Xbox One controller.

19. Being Responsive to the Small Things

Jonathan Snook

24ways.org/201519

It's that time of the year again to trim the tree with decorations. Or maybe a DOM tree?

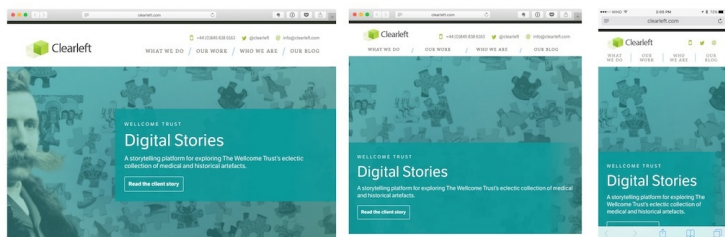
Any web page is made of HTML elements that lay themselves out in a tree structure. We start at the top and then have multiple branches with branches that branch out from there.



To decorate our tree, we use CSS to specify which branches should receive the tinsel we wish to adorn upon it. It's all so lovely.

In years past, this was rather straightforward. But these days, our trees need to be versatile. They need to be responsive!

Responsive web design is pretty wonderful, isn't it? Based on our viewport, we can decide how elements on the page should change their appearance to accommodate various constraints using media queries.

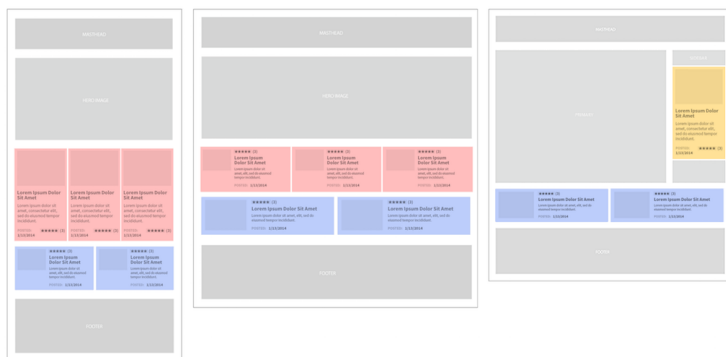


19-1. Clearleft have a delightfully clean and responsive site

Alas, it's not all sunshine, lollipops, and rainbows.

With complex layouts, we may have design chunks — let's call them components — that appear in different contexts. Each context may end up providing its own constraints on the design, both in its default state and in its possibly various responsive states.

Being Responsive to the Small Things



Media queries, however, limit us to the context of the entire viewport, not individual containers on the page. For every container our component lives in, we need to specify how to rearrange things in that context. The more complex the system, the more contexts we need to write code for.

```
@media (min-width: 800px) {  
  .features > .component { }  
  .sidebar > .component {}  
  .grid > .component {}  
}
```

Each new component and each new breakpoint just makes the entire system that much more difficult to maintain.

```
@media (min-width: 600px) {  
  .features > .component { }  
  .grid > .component {}  
}
```

```
@media (min-width: 800px) {  
  .features > .component { }  
  .sidebar > .component {}  
  .grid > .component {}  
}  
  
@media (min-width: 1024px) {  
  .features > .component { }  
}
```

ENTER CONTAINER QUERIES

Container queries, also known as element queries, allow you to specify conditional CSS based on the width (or maybe height) of the container that an element lives in. In doing so, you no longer have to consider the entire page and the interplay of all the elements within.

With container queries, you'll be able to consider the breakpoints of just the component you're designing. As a result, you end up specifying less code and the components you develop have fewer dependencies on the things around them. (I guess that makes your components *more* independent.)

Awesome, right?

There's only one catch.

Browsers can't do container queries. There's not even an official specification for them yet. The **Responsive Issues (née Images) Community Group** is looking into solving how such a thing would actually work.

See, container queries are tricky from an implementation perspective. The contents of a container can affect the size of the container. Because of this, you end up with troublesome circular references.

For example, if the width of the container is under 500px then the width of the child element should be 600px, and if the width of the container is over 500px then the width of the child element should be 400px.

Can you see the dilemma? When the container is under 500px, the child element resizes to 600px and suddenly the container is 600px. If the container is 600px, then the child element is 400px! And so on, forever. This is bad.

I guess we should all just go home and sulk about how we just got a pile of socks when we really wanted the Millennium Falcon.

OUR SAVIOUR THIS CHRISTMAS: JAVASCRIPT

The three wise men — Tim Berners-Lee, Håkon Wium Lie, and Brendan Eich — brought us the gifts of HTML, CSS, and JavaScript.

To date, there are a handful of open source solutions to fill the gap until a browser implementation sees the light of day.

- **Elementary** by Scott Jehl
- **ElementQuery** by Tyson Matanich
- **EQ.js** by Sam Richards
- **CSS Element Queries** from Marcj

Using any of these can sometimes feel like your toy broke within ten minutes of unwrapping it.

Each take their own approach on how to specify the query conditions. For example, Elementary, the smallest of the group, only supports `min-width` declarations made in a `:before` selector.

```
.mod-foo:before {  
  content: "300 410 500";  
}
```

The script loops through all the elements that you specify, reading the content property and then setting an attribute value on the HTML element, allowing you to use CSS to style that condition.

```
.mod-foo[data-minwidth~="300"] {  
  background: blue;  
}
```

To get the script to run, you'll need to set up event handlers for when the page loads and for when it resizes.

```
window.addEventListener( "load", window.elementary,  
false );  
window.addEventListener( "resize", window.elementary,  
false );
```

This works okay for static sites but breaks down on pages where elements can expand or contract, or where new content is dynamically inserted.

In the case of EQ.js, the implementation requires the creation of the breakpoints in the HTML. That means that you have implementation details in HTML, JavaScript, and CSS. (Although, with the JavaScript, once it's in the build system, it shouldn't ever be much of a concern unless you're tracking down a bug.)

Another problem you may run into is the use of content delivery networks (CDNs) or cross-origin security issues. The ElementQuery and CSS Element Queries libraries need to be able to read the CSS file. If you are unable to set up proper cross-origin resource sharing (CORS) headers, these libraries won't help.

At Shopify, for example, we had all of these problems. The admin that store owners use is very dynamic and the CSS and JavaScript were being loaded from a CDN that prevented the JavaScript from reading the CSS.

To go responsive, the team built their own solution — one similar to the other scripts above, in that it loops through elements and adds or removes classes (instead of data attributes) based on minimum or maximum width.

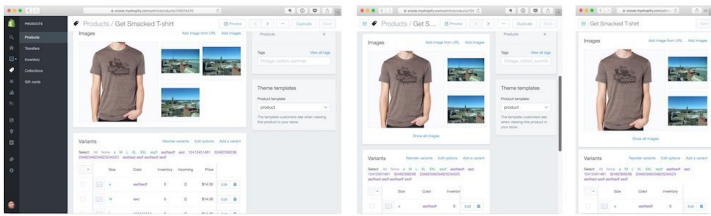
The caveat to this particular approach is that the declaration of breakpoints had to be done in JavaScript.

```
elements = [  
  { 'module': ".carousel", "className": 'alpha',  
    minWidth: 768, maxWidth: 1024 },  
  { 'module': ".button", "className": 'beta', minWidth:  
    768, maxWidth: 1024 },  
  { 'module': ".grid", "className": 'cappa', minWidth:  
    768, maxWidth: 1024 }  
]
```

With that done, the script then had to be set to run during various events such as inserting new content via Ajax calls. This sometimes reveals itself in flashes of unstyled breakpoints (FOUB). An unfortunate side effect but one largely imperceptible.

Using this approach, however, allowed the Shopify team to make the admin responsive really quickly. Each member of the team was able to tackle the responsive story for a particular component without much concern for how all the other components would react.

Being Responsive to the Small Things



Each element responds to its own breakpoint that would amount to dozens of breakpoints using traditional breakpoints. This approach allows for a truly fluid and adaptive interface for all screens.

CHRISTMAS IS OVER

I wish I were the bearer of greater tidings and cheer. It's not all bad, though. We may one day see browsers implement container queries natively. At which point, we shall all rejoice!

ABOUT THE AUTHOR



Jonathan Snook writes about tips, tricks, and bookmarks on his blog at Snook.ca. He has also written for A List Apart and .net magazine, and has co-authored two books, *The Art and Science of CSS* and *Accelerated DOM Scripting*. He has also authored and received world-wide acclaim for the self-published book, *Scalable and Modular Architecture for CSS* sharing his experience and best practices on CSS architecture.

Photo: Patrick H. Lauke

20. Make a Comic

Rebecca Cottrell

24ways.org/201520

For something slightly different over Christmas, why not step away from your computer and make a comic?



20-1. Definitely not the author working on a comic in the studio, with the desk displaying some of the things you need to make a comic on paper.

WHY MAKE A COMIC?

First of all, it's truly fun and it's not that difficult. If you're a designer, you can use skills you already have, so why not take some time to indulge your aesthetic whims and make something for yourself, rather than for a client or your company. And you can use a computer – or not.

If you're an interaction designer, it's likely you've already made a storyboard or flow, or designed some characters for personas. This is a wee jump away from that, to the

realm of storytelling and navigating human emotions through characters who may or may not be human. Similar medium and skills, different content.

It's not a client deliverable but something that stands by itself, and you've nobody's criteria to meet except those that exist in your imagination!

Thanks to your brain and the alchemy of comics, you can put nearly anything in a sequence and your brain will find a way to make sense of it. Scott McCloud wrote about the non sequitur in comics:

“There is a kind of alchemy at work in the space between panels which can help us find meaning or resonance in even the most jarring of combinations.”

Here's an example of a non sequitur from Scott McCloud's *Understanding Comics* – the images bear no relation to one another, but since they're in a sequence our brains do their best to understand it:



Once you know this it takes the pressure off somewhat. It's a fun thing to keep in mind and experiment with in your comics!

MATERIALS NEEDED

- A4 copy/printing paper
- HB pencil for light drawing
- Dip pen and waterproof Indian ink
- Bristol board (or any good quality card with a smooth, durable surface)

STEP 1: GET IDEAS

You'd be surprised where you can take a small grain of an idea and develop it into an interesting comic. Think about a funny conversation you had, or any irrational fears, habits, dreams or anything else. Just start writing and drawing. Having ideas is hard, I know, but you will get some ideas when you start working.

One way to keep track of ideas is to keep a sketch diary, capturing funny conversations and other events you could use in comics later.

You might want to just sketch out the whole comic very roughly if that helps. I tend to sketch the story first, but it usually changes drastically during step 2.

STEP 2: EDIT YOUR STORY USING THUMBNAILS



20-2. How thumbnailing works.

Why use thumbnails? You can move them around or get rid of them!

Drawings are harder and much slower to edit than words, so you need to draw something very quick and very rough. You don't have to care about drawing quality at this point.

You might already have a drafted comic from the previous step; now you can split each panel up into a thumbnail like the image above.

Get an A4 sheet of printing paper and tear it up into squares. A thumbnail equals a comic panel. Start drawing one panel per thumbnail. This way you can move scenes

and parts of the story around as you work on the pacing. It's an extremely useful tip if you want to expand a moment in time or draw out a dialogue, or if you want to just completely cut scenes.

STEP 3: PLAN A LAYOUT

So you've got the story more or less down: you now need to know how they'll look on the page. Sketch a layout and arrange the thumbnails into the layout.

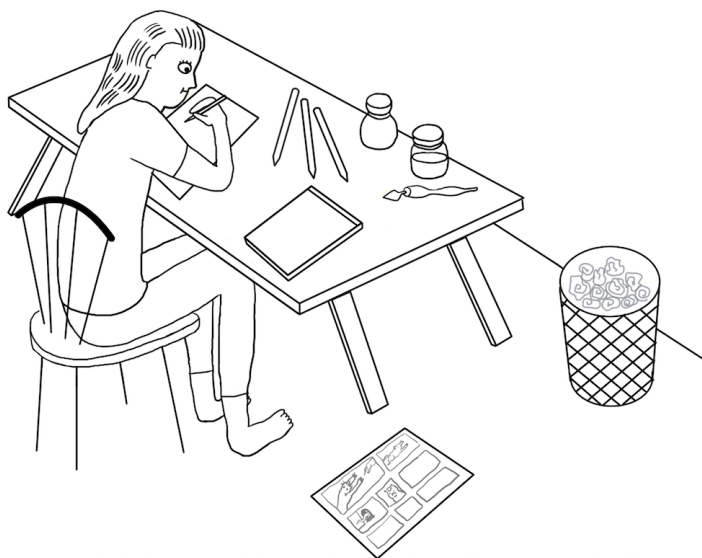
The simplest way to do this is to divide an A4 page into equal panels — say, nine. But if you want, you can be more creative than that. *The Gigantic Beard That Was Evil* by Stephen Collins is an excellent example of the scope for using page layout creatively. You can really push the form: play with layout, scale, story and what you think of as a comic.

STEP 4: DRAW THE COMIC

I recommend drawing on A4 Bristol board paper since it has a smooth surface, can tolerate a lot of rubbing out and holds ink well. You can get it from any art shop.

Using your thumbnails for reference, draw the comic lightly using an HB pencil. Don't make the line so heavy that it can't be erased (since you'll ink over the lines later).

STEP 5: INK THE COMIC



20-3. Image before colour was added.

You've drawn your story. Well done!

Now for the fun part. I recommend using a dip pen and some waterproof ink. Why waterproof? If you want, you can add an ink wash later, or even paint it.

If you don't have a dip pen, you could also use any quality pen. Carefully go over your pencilled lines with the pen, working from top left to right and down, to avoid smudging it. It's unfortunately easy to smudge the ink from the dip pen, so I recommend practising first.

You've made a comic!

STEP 6: ADDING COLOUR

Comics traditionally had a limited colour palette before computers (here's an in-depth explanation if you're curious). You can actually do a huge amount with a restricted colour palette. Ellice Weaver's comics show how very nicely how you can paint your work using a restricted palette. So for the next step, resist the temptation to add ALL THE COLOURS and consider using a limited palette.

Once the ink is completely dry, erase the pencilled lines and you'll be left with a beautiful inked black and white drawing.

You could use a computer for this part. You could also photocopy it and paint straight on the copy. If you're feeling really brave, you could paint straight on the original. But I'd suggest not doing this if it's your first try at painting!

What follows is an extremely basic guide for painting using Photoshop, but there are hundreds of brilliant articles out there and different techniques for digital painting.

How to paint your comic using Photoshop

- **Scan the drawing and open it in Photoshop.** You can adjust the levels (`Image → Adjustments → Levels`) to make the lines darker and crisper, and the paper invisible. At this stage, you can erase any smudges or mistakes. With a Wacom tablet, you could even completely redraw parts! Computers are just amazing. Keep the line art as its own layer.
- **Add a new layer on top of the lines, and set the layer state from normal to multiply.** This means you can paint your comic without obscuring your lines. Rename the layer something else, so you can keep track.
- **Start blocking in colour.** And once you're happy with that, experiment with adding tone and texture.

CHRISTMAS COMIC CHALLENGE!

Why not challenge yourself to make a short comic over Christmas? If you make one, share it in the comments. Or show me on **Twitter** — I'd love to see it.



Credit: Many of these techniques were learned on the Royal Drawing School's brilliant 'Drawing the Graphic Novel' course.

ABOUT THE AUTHOR



Rebecca Cottrell is an independent interaction designer, currently working with the Government Digital Service on GOV.UK. She lives in London, owns two adorable lovebirds, and likes drawing comics. Some of which you can see on her tumblr. She is on twitter.

21. What's Ahead for Your Data in 2016?

Heather Burns

24ways.org/201521

Who owns your data? Who decides what can you do with it? Where can you store it? What guarantee do you have over your data's privacy? Where can you publish your work? Can you adapt software to accommodate your disability? Is your tiny agency subject to corporate regulation? Does another country have rights over your intellectual property?

If you aren't the kind of person who is interested in international politics, I hate to break it to you: in 2016 the legal foundations which underpin our work on the web are being revisited in not one but *three* major international political agreements, and every single one of those questions is up for grabs. These agreements – the draft EU Data Protection Regulation (EUDPR), the Trans-Pacific Partnership (TPP), and the draft Transatlantic Trade and Investment Partnership (TTIP) – stand poised to have a major impact on your data, your workflows, and

your digital rights. While some proposed changes could protect the open web for the future, other provisions would set the internet back several decades.

In this article we will review the issues you need to be aware of as a digital professional. While each of these agreements covers dozens of topics ranging from climate change to food safety, we will focus solely on the aspects which pertain to the work we do on the web.

THE TRANS-PACIFIC PARTNERSHIP

The Trans-Pacific Partnership (TPP) is a free trade agreement between the US, Japan, Malaysia, Vietnam, Singapore, Brunei, Australia, New Zealand, Canada, Mexico, Chile and Peru – a bloc comprising 40% of the world's economy. The agreement is expected to be signed by all parties, and thereby to come into effect, in 2016. This agreement is ostensibly about the bloc and its members working together for their common interests. However, the latest draft text of the TPP, which was formulated entirely in secret, has only been made publicly available on a **Medium blog** published by the U.S. Trade Representative which features a patriotic banner at the top proclaiming “TPP: Made in America.” The message sent about who holds the balance of power in this agreement, and whose interests it will benefit, is clear.

By far the most controversial area of the TPP has centred around the **provisions on intellectual property**. These include copyright terms of up to 120 years, mandatory takedowns of allegedly infringing content in response to just *one* complaint regardless of that complaint's validity, heavy and disproportionate penalties for alleged violations, and – most frightening of all – government seizures of equipment *allegedly* used for copyright violations. All of these provisions have been raised without regard for the fact that a trade agreement is not the appropriate venue to negotiate intellectual property law.

Other draft TPP provisions would restrict the digital rights of people with disabilities by banning the workarounds they use every day. These include no exemptions for the adaptations of copywritten works for use in accessible technology (such as text-to-speech in ebook readers), a ban on circumventing DRM or digital locks in order to convert a file to an accessible format, and requiring the takedown of adapted works, such as a video with added subtitles, even if that adaptation would normally have fallen under the definition of fair use.

The e-commerce provisions would prohibit data localisation, the practice of requiring data to be physically stored on servers within a country's borders. Data localisation is **growing in popularity** following the Snowden revelations, and some of your own personal

data may have been recently “localised” in response to the **Safe Harbor verdict**. Prohibiting data localisation through the TPP would address the symptom but not the cause.

The Electronic Frontier Foundation has published an **excellent summary** of the digital rights issues raised by the agreement along with suggested actions American readers can take to speak out.

TRANSATLANTIC TRADE AND INVESTMENT PARTNERSHIP

TTIP stands for the Transatlantic Trade and Investment Partnership, a draft free trade agreement between the United States and the EU. The plan has been hugely controversial and divisive, and the internet and digital provisions of the draft form just a small part of that contention.

The most striking digital provision of TTIP is an attempt to circumvent and override European data protection law. As EDRI, a European digital rights organisation, noted:

“the US proposal would authorise the transfer of EU citizens’ personal data to any country, trumping the EU data protection framework, which ensures that this data can only be transferred in clearly defined circumstances. For years, the US has been trying to bypass the default requirement for storage of personal data in the EU. It is therefore not surprising to see such a proposal being {introduced} in the context of the trade negotiations.”

This draft provision was written *before* the Safe Harbor data protection agreement between the EU and US **was invalidated** by the Court of Justice of the European Union. In other words, there is no longer any protective agreement in place, and our data is as vulnerable as this political situation. However, data protection is a matter of its own law, the acting Data Protection Directive and the draft EU Data Protection Reform. A trade agreement, be it the TTIP or the TPP, is not the appropriate place to revamp a law on data protection.

Other digital law issues raised by TTIP include the possibility of **renegotiating standards on encryption** (which in practice means lowering them) and **renegotiating intellectual property rights** such as copyright. The spectre of **net neutrality** has even put in an appearance, with an attempt to introduce rules on access to the internet itself being introduced as provisions.

TTIP is still under discussion, and this month the **EU trade representative** said that “we agreed to further intensify our work during 2016 to help negotiations move forward rapidly.” This has been cleverly worded: this means the agreement has little chance of being passed or coming into effect in 2016, which buys civil society more precious time to speak out.

THE EU DATA PROTECTION REGULATION

On 15 December 2015 the European Commission announced their agreement on the text of the draft **General Data Protection Regulation**. This law will replace its predecessor, the EU Data Protection Regulation of 1995, which has done a remarkable job of protecting data privacy across the continent throughout two decades of constant internet evolution.

The goal of the reform process has been to **return power over data, and its uses, to citizens**. Users will have more control over what data is captured about them, how it is used, how it is retained, and how it can be deleted. Businesses and digital professionals, in turn, will have to restructure their relationships with client and customer data. Compliance obligations will increase, and difficult choices will have to be made. However, this time should be seen as an opportunity to rethink our relationship with data. After Snowden, Schrems, and Safe Harbor, it is clear that we cannot go back to the way things were before. In

an era of where every one of our heartbeats is recorded on a wearable device and uploaded to a surveilled data centre in another country, the need for reform has never been more acute.

While texts of the draft GDPR are available, there is not enough mulled wine in the world that will help you get through them. Instead, the law firm **Fieldfisher** Waterhouse has produced this helpful infographic which will give you a good idea of the changes we can expect to see (view full size):

Debunking EU Data Protection Reform fieldfisher

10 things you should know

- 1 The pool of data which is potentially personal gets deeper**
Under the new law, if information that is not directly linked to an individual can be linked to an individual by other means, it is personal data. This includes data that is not directly linked to an individual but can be linked to an individual by other means. This includes data that is not directly linked to an individual but can be linked to an individual by other means.
- 2 Out-of-scope today, is scope in the future – what's caught?**
Even if you are not established in the EU, you may be caught by the new rules if you are a data controller whose processing activities are directed at EU residents or if you are processing personal data in relation to offering goods or services to, or monitoring the behaviour of, EU residents. Data processors established in the EU will be subject to new obligations.
- 3 You may need to "re-think" your legal justification for processing personal data**
The ability of third parties to rely on the "legitimate interests" ground for processing is under threat. Further, "consent" as a lawful ground for processing is likely to be subject to very strict conditions. Businesses should revisit existing data collection practices and consider consent mechanisms which provide more flexibility in the future.
- 4 Individuals' new enhanced rights**
The proposal requires the provision of additional information in response to individual access requests and a new way of exercising the right to object to direct marketing. New rights for individuals are also proposed, e.g. "the right to be forgotten" and a right of "data portability" which may require providing copies of personal data records in a standardised electronic format.
- 5 Your big Data analysis and profiling activities may be narrowly curtailed**
Explicit consent is likely to be required in most instances to process personal data for profiling. Businesses should assess the nature of any profiling activities that they undertake and their compliance with possible consent mechanisms.
- 6 You should design for compliance**
Under the proposed "privacy by design" requirement, you will need to develop robust policies, procedures and systems at the outset of product development. The proposed "privacy by design" principle is that, by default, any personal data that is necessary for a specific purpose are to be processed. This may mean that certain types of personal data will need to be switched off by default.
- 7 Accountability principles – more paperwork?**
The draft proposal introduces an accountability principle which requires significant documentation requirements. Businesses should ensure they existing data protection policies and procedures are robust and that they meet the expected standards.
- 8 You may need to appoint a DPO**
Under the proposal, a data controller or processor must designate a data protection officer (DPO) for a minimum period of 2 years in certain specified circumstances, if frequently and systematically monitor handling activities to be reported in response to processing operations which cause the designation.
- 9 Data transfer restrictions are here to stay, but so are IBCs**
You will still have to jump through hoops in order to lawfully transfer data outside of Europe. The Commission's proposal expressly acknowledges the validity of Binding Corporate Rules (BCRs) including IBCs. In this respect, as a valid legal solution – IBCs are here to stay.
- 10 One stop shop enforcement – with greater teeth**
There are likely to be substantial fines for non-compliance – as high as 2% of global turnover. For a data controller established in more than one Member State, the data protection authority of the country of main establishment is the competent authority. This is known as the "one stop shop" principle.

Europe's new General Data Protection Regulation ("Regulation") remains subject to some significant negotiation and is unlikely to come into force until 2017 – 2018. Any business touching upon personal data should start considering the future rules and what they mean for their business, not least as there are potentially some significant changes. The overall intent is to update the rules and to introduce greater harmonisation of these rules across EU Member States. As you would expect, the Fieldfisher Privacy, Security and Information team is monitoring the progress of the legislative changes and we are naturally cautious of recommending any significant preparatory steps until the exact make-up of the final rules are better understood (expected during the early part of 2016). This "Infographic" prepares you for the potential impact of the proposed Regulation.

The most surprising outcome announced on 15 December was the new regulation's teeth. Under the new law, companies that fail to heed the updated data protection rules will face fines of up to 4% of their global turnover.

Additionally, the law expands the liability for data protection to both the controller (the company hosting the data) and the data processor (the company using the data). The new law will also introduce a one-stop shop for resolving concerns over data misuse. Companies will no longer be able to headquarter their European operations in countries which are perceived to have relatively light-touch data protection enforcement (that means you, Ireland) as a means of automatically rejecting any complaints filed by citizens outside that country.

For digital professionals, the most immediate concern is analytics. In fact, I am going to make a prediction: in 2016 we will begin to see the same misguided war on analytics that we saw on cookies. By increasing the legal liabilities for both data processors and controllers – in other words, the company providing the analytics as well as the site administrator studying them – the new regulation risks creating disproportionate burdens as well as the same “guilt by association” risks we saw in 2012. There have already been statements made by some within the privacy community that analytics are tracking, and tracking is surveillance, therefore analytics are evil. Yet “just don’t use analytics,” as was suggested by one advocate, is simply not an option. European regulators should consult with the web community to gain a clear understanding of why analytics are vital to everyday site administrators, and must find a happy medium that protects users’ data

without criminalising every website by default. No one wants a repeat of the crisis of consent, as well as the scaremongering, caused by the cookie law.

Assuming the text is adopted in 2016, the new EU Data Protection Regulation would not come into effect until 2018. We have a considerable challenge ahead, but we also have plenty of time to get it right.

ABOUT THE AUTHOR



What's Ahead for Your Data in 2016?

Heather Burns is a digital law specialist in Glasgow, Scotland. Her focus is researching, writing, and speaking about internet laws and policies which impact the professions of web design and development. She has been a professional web designer since 2007 and earned a postgraduate certification in internet law in 2015.

22. How Tabs Should Work

Remy Sharp

24ways.org/201522

Tabs in browsers (not *browser tabs*) are one of the oldest custom UI elements in a browser that I can think of. They've been done to death. But, sadly, most of the time I come across them, the tabs have been badly, or rather partially, implemented.

So this post is my definition of how a tabbing system should work, and *one* approach of implementing that.

BUT... TABS ARE EASY, RIGHT?

I've been writing code for tabbing systems in JavaScript for coming up on a decade, and at one point I was pretty proud of how small I could make the JavaScript for the tabbing system:

```
var tabs = $(' .tab').click(function () {  
    tabs.hide().filter(this.hash).show();  
}).map(function () {  
    return $(this.hash)[0];  
});
```

```
});
```

```
$('.tab:first').click();
```

Simple, right? Nearly fits in a tweet (ignoring the whole jQuery library...). Still, it's riddled with problems that make it a far from perfect solution.

REQUIREMENTS: WHAT MAKES THE PERFECT TAB?

1. All content is navigable and available without JavaScript (crawler-compatible and low JS-compatible).
2. ARIA roles.
3. The tabs are anchor links that:
 - are clickable
 - have block layout
 - have their href pointing to the id of the panel element
 - use the correct cursor (i.e. `cursor: pointer`).
4. Since tabs are clickable, the user can *open in a new tab/window* and the page correctly loads with the correct tab open.
5. Right-clicking (and Shift-clicking) *doesn't* cause the tab to be selected.
6. Native browser Back/Forward button correctly changes the state of the selected tab (think about it working exactly as if there were no JavaScript in place).

The first three points are all to do with the semantics of the markup and how the markup has been styled. I think it's easy to do a good job by thinking of tabs as links, and not as some part of an application. Links are navigable, and they should work the same way other links on the page work.

The last three points are JavaScript problems. Let's investigate that.

THE SHITMUS TEST

Like a litmus test, here's a couple of quick ways you can tell if a tabbing system is poorly implemented:

- Change tab, then use the Back button (or keyboard shortcut) and it breaks
- The tab isn't a link, so you can't open it in a new tab

These two basic things are, to me, the bare minimum that a tabbing system should have.

WHY IS THIS IMPORTANT?

The people who push their so-called native apps on users can't have more reasons why the web sucks. If something as basic as a tab doesn't work, obviously there's more ammo to push a closed native app or platform on your users.

If you're going to be a web developer, one of your responsibilities is to maintain established interactivity paradigms. This doesn't mean don't innovate. But it *does* mean: stop fucking up my scrolling experience with your poorly executed scroll effects. </rant> :breath:

URI FRAGMENT, ABSOLUTE URL OR QUERY STRING?

A URI fragment (AKA the # hash bit) would be using *mysite.com/config#content* to show the **content** panel. A fully addressable URL would be *mysite.com/config/content*. Using a query string (by way of filtering the page): *mysite.com/config?tab=content*.

This decision really depends on the context of your tabbing system. FOr something like GitHub's tabs to **view a pull request**, it makes sense that the full URL changes.

For our problem though, I want to solve the issue when the page doesn't do a full URL update; that is, your regular run-of-the-mill tabbing system.

I used to be from the school of using the hash to show the correct tab, but I've recently been exploring whether the query string can be used. The biggest reason is that multiple hashes don't work, and comma-separated hash fragments don't make any sense to control multiple tabs (since it doesn't actually link to anything).

For this article, I'll keep focused on using a single tabbing system and a hash on the URL to control the tabs.

MARKUP

I'm going to assume subcontent, so my markup would look like this (yes, this is a cat demo...):

```
<ul class="tabs">
  <li><a class="tab" href="#dizzy">Dizzy</a></li>
  <li><a class="tab" href="#ninja">Ninja</a></li>
  <li><a class="tab" href="#missy">Missy</a></li>
</ul>

<div id="dizzy">
  <!-- panel content -->
</div>
<div id="ninja">
  <!-- panel content -->
</div>
<div id="missy">
  <!-- panel content -->
</div>
```

It's important to note that in the markup the link used for an individual tab references its panel content using the hash, pointing to the `id` on the panel. This will allow our content to connect up without JavaScript and give us a bunch of features for free, which we'll see once we're on to writing the code.

URL-DRIVEN TABBING SYSTEMS

Instead of making the code responsive to the user's *input*, we're going to exclusively use the browser URL and the hashchange event on the window to drive this tabbing system. This way we get Back button support for free.

With that in mind, let's start building up our code. I'll assume we have the jQuery library, but I've also provided the full code working without a library (vanilla, if you will), but it depends on relatively new (polyfillable) tech like `classList` and `dataset` (which generally have IE10 and all other browser support).

Note that I'll start with the simplest solution, and I'll refactor the code as I go along, like in places where I keep calling jQuery selectors.

```
function show(id) {
    // remove the selected class from the tabs,
    // and add it back to the one the user selected
    $(' .tab').removeClass('selected').filter(function () {
        return (this.hash === id);
    }).addClass('selected');

    // now hide all the panels, then filter to
    // the one we're interested in, and show it
    $(' .panel').hide().filter(id).show();
}

$(window).on('hashchange', function () {
    show(location.hash);
});
```

```
});
```

```
// initialise by showing the first panel  
show('#dizzy');
```

This works pretty well for such little code. Notice that we don't have any click handlers for the user and the Back button works right out of the box.

However, there's a number of problems we need to fix:

1. The initialised tab is hard-coded to the first panel, rather than what's on the URL.
2. If there's no hash on the URL, all the panels are hidden (and thus broken).
3. If you scroll to the bottom of the example, you'll find a "top" link; clicking that will break our tabbing system.
4. I've purposely made the page long, so that when you click on a tab, you'll see the page scrolls to the top of the tab. Not a huge deal, but a bit annoying.

From our criteria at the start of this post, we've already solved items 4 and 5. Not a terrible start. Let's solve items 1 through 3 next.

Using the URL to initialise correctly and protect from breakage

Instead of arbitrarily picking the first panel from our collection, the code should read the current `location.hash` and use that if it's available.

The problem is: what if the hash on the URL isn't actually for a tab?

The solution here is that we need to cache a list of known panel IDs. In fact, well-written DOM scripting won't continuously search the DOM for nodes. That is, when the show function kept calling `$('.tab').each(...)` it was wasteful. The result of `$('.tab')` should be cached.

So now the code will collect all the tabs, then find the related panels *from* those tabs, and we'll use that list to double the values we give the show function (during initialisation, for instance).

```
// collect all the tabs
var tabs = $('.tab');

// get an array of the panel ids (from the anchor hash)
var targets = tabs.map(function () {
    return this.hash;
}).get();

// use those ids to get a jQuery collection of panels
var panels = $(targets.join(','));

function show(id) {
    // if no value was given, let's take the first panel
    if (!id) {
        id = targets[0];
    }
    // remove the selected class from the tabs,
    // and add it back to the one the user selected
```

```

    tabs.removeClass('selected').filter(function () {
        return (this.hash === id);
    }).addClass('selected');

    // now hide all the panels, then filter to
    // the one we're interested in, and show it
    panels.hide().filter(id).show();
}

$(window).on('hashchange', function () {
    var hash = location.hash;
    if (targets.indexOf(hash) !== -1) {
        show(hash);
    }
});

// initialise
show(targets.indexOf(location.hash) !== -1 ?
location.hash : '');

```

The core of working out which tab to initialise with is solved in that last line: is there a `location.hash`? Is it in our list of valid targets (panels)? If so, select that tab.

The second breakage we saw in the original demo was that clicking the “top” link would break our tabs. This was due to the `hashchange` event firing and the code didn’t validate the hash that was passed. Now this happens, the panels don’t break.

So far we’ve got a tabbing system that:

- Works without JavaScript.

- Supports right-click and `Shift`-click (and doesn't select in these cases).
- Loads the correct panel if you start with a hash.
- Supports native browser navigation.
- Supports the keyboard.

The only annoying problem we have now is that the page jumps when a tab is selected. That's due to the browser following the default behaviour of an internal link on the page. To solve this, things are going to get a little hairy, but it's all for a good cause.

Removing the jump to tab

You'd be forgiven for thinking you just need to hook a click handler and return `false`. It's what I started with. Only that's not the solution. If we add the click handler, it breaks all the right-click and `Shift`-click support.

There may be another way to solve this, but what follows is the way I found – and it works. It's just a bit... hairy, as I said.

We're going to strip the `id` attribute off the target panel when the user tries to navigate to it, and then put it back on once the `show` code starts to run. This change will mean the browser has nowhere to navigate to for that moment, and won't jump the page.

The change involves the following:

1. Add a click handle that removes the `id` from the target panel, and cache this in a `target` variable that we'll use later in `hashchange` (see point 4).
2. In the same click handler, set the `location.hash` to the current link's hash. This is important because it forces a `hashchange` event regardless of whether the URL actually changed, which prevents the tabs breaking (try it yourself by removing this line).
3. For each panel, put a backup copy of the `id` attribute in a data property (I've called it `old-id`).
4. When the `hashchange` event fires, if we have a `target` value, let's put the `id` *back* on the panel.

These changes result in this final code:

```
/*global $*/

// a temp value to cache *what* we're about to show
var target = null;

// collect all the tabs
var tabs = $(' .tab').on('click', function () {
    target = $(this.hash).removeAttr('id');

    // if the URL isn't going to change, then hashchange
    // event doesn't fire, so we trigger the update
    manually
    if (location.hash === this.hash) {
        // but this has to happen after the DOM update has
        // completed, so we wrap it in a setTimeout 0
        setTimeout(update, 0);
    }
});
```

```

    }
  });

  // get an array of the panel ids (from the anchor hash)
  var targets = tabs.map(function () {
    return this.hash;
  }).get();

  // use those ids to get a jQuery collection of panels
  var panels = $(targets.join(',')).each(function () {
    // keep a copy of what the original el.id was
    $(this).data('old-id', this.id);
  });

  function update() {
    if (target) {
      target.attr('id', target.data('old-id'));
      target = null;
    }

    var hash = window.location.hash;
    if (targets.indexOf(hash) !== -1) {
      show(hash);
    }
  }

  function show(id) {
    // if no value was given, let's take the first panel
    if (!id) {
      id = targets[0];
    }
    // remove the selected class from the tabs,
    // and add it back to the one the user selected
    tabs.removeClass('selected').filter(function () {

```

```

        return (this.hash === id);
    }).addClass('selected');

    // now hide all the panels, then filter to
    // the one we're interested in, and show it
    panels.hide().filter(id).show();
}

$(window).on('hashchange', update);

// initialise
if (targets.indexOf(window.location.hash) !== -1) {
    update();
} else {
    show();
}

```

This version now meets all the criteria I mentioned in my original list, *except* for the ARIA roles and accessibility. Getting this support is actually very cheap to add.

ARIA roles

This article on [ARIA tabs](#) made it very easy to get the tabbing system working as I wanted.

The tasks were simple:

1. Add `aria-role` set to tab for the tabs, and panel for the panels.
2. Set `aria-controls` on the tabs to point to their related panel (by id).

3. I use JavaScript to add `tabindex=0` to all the tab elements.
4. When I add the selected class to the tab, I also set `aria-selected` to true and, inversely, when I remove the selected class I set `aria-selected` to false.
5. When I hide the panels I add `aria-hidden=true`, and when I show the specific panel I set `aria-hidden=false`.

And that's it. Very small changes to get full sign-off that the tabbing system is bulletproof and accessible.

Check out the [final version](#) (and the [non-jQuery version](#) as promised).

IN CONCLUSION

There's a *lot* of tab implementations out there, but there's an equal amount that break the browsing paradigm and the simple linkability of content. Clearly there's a special hell for those tab systems that don't even use links, but I think it's clear that even in something that's relatively simple, it's the small details that make or break the user experience.

Obviously there are corners I've not explored, like when there's more than one set of tabs on a page, and equally whether you should deliver the initial markup with the

correct tab selected. I think the answer lies in using query strings in combination with hashes on the URL, but maybe that's for another year!

ABOUT THE AUTHOR



Remy Sharp is the founder and curator of Full Frontal, the UK based JavaScript conference. He also ran jQuery for Designers, co-authored *Introducing HTML5* (adding all the JavaScripty bits) and likes to grumble on Twitter.

Whilst he's not writing articles or running and speaking at conferences, he runs his own development and training company in Brighton called **Left Logic**. And he built these too: Confwall, jsbin.com, html5demos.com, remote-tilt.com, responsivepx.com, nodemon, inliner, mit-license.org, snapbird.org, 5 minute fork and jsconsole.com!

23. Blow Your Own Trumpet

Andy Clarke

24ways.org/201523

Even if your own trumpet's tiny and fell out of a Christmas cracker, blowing it isn't something that everyone's good at. Some people find selling themselves and what they do difficult. But, you know what? Boo hoo hoo. If you want people to buy something, the reality is you'd better get good at selling, especially if that something is you.

For web professionals, the best place to tell potential business customers or possible employers about what you do is on your own website. You can write what you want and how you want, but that doesn't make knowing what to write any easier. As a matter of fact, writing for yourself often proves harder than writing for someone else.

I spent this autumn thinking about what I wanted to say about **Stuff & Nonsense** on the website we relaunched recently. While I did that, I spoke to other designers about how they struggled to write about their businesses.

If you struggle to write well, don't worry. You're not on your own. Here are five ways to hit the right notes when writing about yourself and your work.

BE GENUINE ABOUT WHO YOU ARE

I've known plenty of talented people who run a successful business pretty much single-handed. Somehow they still feel awkward presenting themselves as individuals. They wonder whether describing themselves as a company will give them extra credibility. They especially agonise over using "we" rather than "I" when describing what they do. These choices get harder when you're a one-man band trading as a limited company or LLC business entity.

If you mainly work alone, don't describe yourself as anything other than "I". You might think that saying "we" makes you appear larger and will give you a better chance of landing bigger and better work, but the moment a prospective client asks, "How many people are you?" you'll have some uncomfortable explaining to do. This will distract them from talking about your work and derail your sales process. There's no need to be anything other than genuine about how you describe yourself. You should be proud to say "I" because working alone isn't something that many people have the ability, business acumen or talent to do.

EXPLAIN WHAT YOU ACTUALLY DO

How many people do precisely the same job as you? Hundreds? Thousands? The same goes for companies. If yours is a design studio, development team or UX consultancy, there are countless others saying exactly what you're saying about what you do. Simply stating that you code, design or – God help me – “handcraft digital experiences” isn't enough to make your business sound different from everyone else. Anyone can and usually does say that, but people buy more than deliverables. They buy something that's unique about you and your business.

Potentially thousands of companies deliver code and designs the same way as Stuff & Nonsense, but our clients don't just buy page designs, prototypes and websites from us. They buy our taste for typography, colour and layout, summed up by our “It's the taste” tagline and **bowler hat tip** to the PG Tips chimps. We hope that potential clients will understand what's unique about us. Think beyond your deliverables to what people actually buy, and sell the uniqueness of that.

DESCRIBE WORK IN PROGRESS

It's sad that current design trends have made it almost impossible to tell one website from another. So many designers now demonstrate finished responsive website

designs by pasting them onto iMac, MacBook, iPad and iPhone screens that their portfolios don't fare much better. Every designer brings their own experience, perspective and process to a project. In my experience, it's understanding those differences which forms a big part of how a prospective client makes a decision about who to work with. Don't simply show a prospective client the end result of a previous project; explain your process, the development of your thinking and even the wrong turns you took.

Traditional case studies, like the one I've just written about **Stuff & Nonsense's** work for **WWF UK**, can take a lot of time. That's probably why many portfolios get out of date very quickly. Designers make new work all the time, so there must be a better way to show more of it more often, to give prospective clients a clearer understanding of what we do. At **Stuff & Nonsense** our solution was to create a **feed** where we could post fragments of design work throughout a project. This also meant rewriting our **Contract Killer** to give us permission to publish work before someone signs it off.

OUTLINE A CLIENT'S EXPERIENCE

Recently a client took me to one side and offered some valuable advice. She told me that our website hadn't described anything about the experience she'd had while working with us. She said that knowing more about how we work would've helped her make her buying decision.

When a client chooses your business, they're hoping for more than a successful outcome. They want their project to run smoothly. They want to feel that they made a correct decision when they chose you. If they work for an organisation, they'll want their good judgement to be recognised too. Our client didn't recognise her experience because we hadn't made our own website part of it. Remember, the challenge of creating a memorable user experience starts with selling to the people paying you for it.

ADDRESS YOUR IDEAL CLIENT

It's important to understand that a portfolio's job isn't to document your work, it's to attract new work from clients you want. Make sure that work you show reflects the work you want, because what you include in your portfolio often leads to more of the same.

When you're writing for your portfolio and elsewhere on your website, imagine that you're addressing your ideal client. Picture them sitting opposite and answer the

questions they'd ask as you would in conversation. Be direct, funny if that's appropriate and serious when it's not. If it helps, ask a friend to read the questions aloud and record what you say in response. This will help make what you write sound natural. I've found this technique helps clients write copy too.

TOOT YOUR OWN HORN

Some people confuse expressing confidence in yourself and your work as boastfulness, but in a competitive world the reality is that if you are to succeed, you need to show confidence so that others can show their confidence in you. If you want people to hear you, pick up your trumpet and blow it.

ABOUT THE AUTHOR



Andy Clarke is an art director and web designer at the UK website design studio 'Stuff & Nonsense.' There he designs websites and applications for clients from around the world. Based in North Wales, Andy's also the author of two web design books, 'Transcending CSS' and the new 'Hardboiled Web Design Fifth Anniversary Edition' and is well known for his many conference presentations and over ten years of contributions to the web design industry. Jeffrey Zeldman once called him a "triple talented bastard." If you know of Jeffrey, you'll know how happy that made him.

24. Solve the Hard Problems

Drew McLellan

24ways.org/201524

So, here we find ourselves on the cusp of 2016. We've had a good year – the web is still alive, no one has switched it off yet. Clients still have websites, teenagers still have phone apps, and there continue to be plenty of online brands to meaningfully engage with each day. Good job team, high fives all round.

As it's the time to make resolutions, I wanted to share three small ideas to take into the new year.

GET GOOD AT WHAT YOU DO

"How do you get to Carnegie Hall?" the old joke goes.

"Practise, practise, practise."

We work in an industry where there is an awful lot to learn. There's a lot to learn to get started and then once you do, there's a lot more to learn to keep your skills current. Just when you think you've mastered something, it changes.

This is true of many industries, of course, but the sheer pace of change for us makes learning not an annual activity, but daily. Learning takes time, and while I'm not convinced that every skill takes the fabled ten thousand hours to master, there is certainly no escaping that to remain current we must reinvest time in keeping our skills up to date.

Picking where to spend your time

One of the hardest aspects of this thing of ours is just choosing *what* to learn. If you, like me, invested any time in learning the Less CSS preprocessor over the last few years, you'll probably now be spending your time relearning Sass instead. If you spent time learning Grunt, chances are you'll now be thinking about whether you should switch to Gulp. It's not just that there are new types of tools, there are new tools and frameworks to do the things you're already doing, but, well, differently.

Deciding what to learn is hard and the costs of backing the wrong horse can seriously mount up; so much so that by the time you've learned and then relearned the tools

everyone says you need for your job, there's rarely enough time to spend really getting to know how best to use them.

Practise, practise, practise

Do you know how you *don't* get to Carnegie Hall? By learning a new instrument each week. It takes time and experience to really learn something well. That goes for a new JavaScript framework as much as a violin. If you flit from one shiny new thing to another, you're destined to produce amateurish work forever.

Learn the new thing, but then stick with it long enough to get really good at it – even if Twitter trolls try to convince you it's not cool. What's really not cool is living as a forevernoob.

If you're still not sure what to learn, go back to basics. Considering a new CSS or JavaScript framework? Invest that time in learning the underlying CSS or JavaScript really well instead. Those skills will stand the test of time.

AUDIENCE AND PURPOSE

Back when I was in school, my English teacher (a nice Welsh lady, who I appreciate more now than I did back then) used to love to remind us that every piece of writing should have an audience and a purpose. So much so that *audience and purpose* almost became her catch phrase. For

every essay, article or letter, we were reminded to consider who we were writing it for and what we were trying to achieve.

It's something I think about a lot; certainly when writing, but also in almost every other creative endeavour. Asking *who is this for* and *what am I trying to achieve* applies equally to designing a logo or website, through to composing music or writing software.

Being productive

It seems like everyone wants to have a product these days. As someone who used to do client services work and now has a product company, I often talk with people who are interested in taking something they've built in-house and turning it into a product. You know the sort of thing: a design agency with its own CMS or project management web app; the very logical thought process of: if this helps our business, maybe others will find it valuable too; the question that inevitably follows: could we turn this into a product?

Whether consciously or not, the audience and purpose influence nearly every aspect of your creative process. Once written or designed or developed or created, revising a work to change the audience and purpose can be quite a challenge. No matter how much you want to turn the tension-building, atmospheric music for a horror

film into a catchy chart hit, it's going to be a struggle. Yes, it's music, but that's neither the audience nor purpose for which it was created.

The same is absolutely true for your in-house tools – those were also designed for a specific audience and purpose. Your in-house CMS would have been designed with an audience of your own development team, who are busy implementing sites for clients. The purpose is to make that team more productive overall, taking into account considerations of maintaining multiple sites on a common codebase, training clients, a more mature and stable platform and all the other benefits of resuing the same code for each project. The audience is *your team* and the purpose *increased productivity*.

That's very different from a customer who wants to buy a polished system to use off-the-shelf. If their needs perfectly aligned with yours then they wouldn't be in the market for your product – they would have built their own.

Sometimes you hear the advice to “scratch your own itch” when it comes to product design. I don't completely agree. Got an itch? Great. Find other itchy people and sell them a backscratcher.

Building a product, like designing a website, is a lot of work. It requires knowing your audience and purpose inside out. You can't fudge it and you can't just hope you'll find an audience for some old thing you have lying around.

Always consider the audience and purpose for everything you create. It's often the difference between success and failure.

SOLVE THE HARD PROBLEMS

Human beings have a natural tendency to avoid hard problems. In digital design (websites, software, whatever) the received wisdom is often that we can get 80% of the way towards doing the hard thing by doing something that's not very hard.

Do you know what you get at the end of it? Paid. But nothing really great ever happens that way.

I worked on a client project a while back where one of the big challenges was making full use of the massive image library they had built up over the years. The client had tens of thousands of photographs, along with a fair amount of video and a large MP3 audio library too. If it wasn't managed carefully, storage sizes would get out of control, content would go unattributed, and everything would get very messy very quickly.

I could tell from the outset that this aspect of the project was going to be a constant problem. So we tackled it head-on. We designed and built a media management system to hold and process all the assets, and added an API so the content management system could talk to it. Every time the site needed a photo at a new size, it made an API request to the system and everything was handled seamlessly.

It was a daunting job to invest all the time and effort in building that dedicated system and API, but it really paid off. Instead of having the constant troubles of a vast library of media, it became one of the strongest parts of the project.

Turn your hardest problems into your biggest strengths

There's a funny thing about hard problems. The hardest problems are the most fun to solve and have the biggest impact.

Maybe you're the sort of person who clocks in for work, does their job and clocks out at 5pm without another thought. But I don't think you are, because you're here reading this. If you really love what you do, I don't think you can be satisfied in your work unless you're seeking out and working on those hard problems. That's where the magic is.



The new year is a helpful time to think about breaking bad habits. Whether it's smoking a bit less, or going to the gym a bit more, the ticking over of the calendar can provide the motivation for a new start. I have some suggestions for you.

1. **Get good at what you do.** Practise your skills and don't just flit from one shiny thing to the next.
2. **Remember who you're doing it for and why.** Consider the audience and purpose for everything you create.
3. **Solve the hard problems.** It's more interesting, more satisfying, and has a greater impact.

As we move into 2016, these are the things I'm going to continue to work on. Maybe you'd like to join me.

ABOUT THE AUTHOR



Drew McLellan is lead developer on your favourite content management systems, **Perch** and **Perch Runway**. He is Director and Senior Developer at edgeofmyseat.com in Bristol, England, and is 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**, **tweets a lot** and tries to stay upright on his bicycle.