# 2 NAVIGATION

"I may not have gone where I intended to go, but I think I have ended up where I needed to be."
—DOUGLAS ADAMS, The Long Dark Tea-Time of the Soul

I'VE BEEN READING about a man named Pius "Mau" Piailug who, in 1976, navigated a large voyaging canoe across the Pacific, traveling more than 3,000 miles from Hawai'i to Tahiti. Piailug sailed without the aid of maps, computer-assisted navigation, or any other equipment—instead, he used the stars, sun, and moon to guide him. In fact, the instrument most helpful to him never made it onto his vessel: a star compass, a ring of shells, coral, or pebbles placed around a center point (FIG 2.1). The simple-looking instrument helped young navigators of Piailug's tradition understand the relationship between the horizon—the outer ring of the compass—and the canoe in the center. This, coupled with years of training at sea, was what helped Piailug complete his journey, and prove that traditional navigation was still relevant in a modern world.

I think of Piailug's journey often, and of his star compass in particular. Because if we've done our job right, a website's navigation should act as a kind of compass: it helps new users orient themselves within a site hierarchy, and guides them to their destination. But with all the different tiers and types of menus our sites contain, designing intuitive, usable navigation can feel like a formidable task.

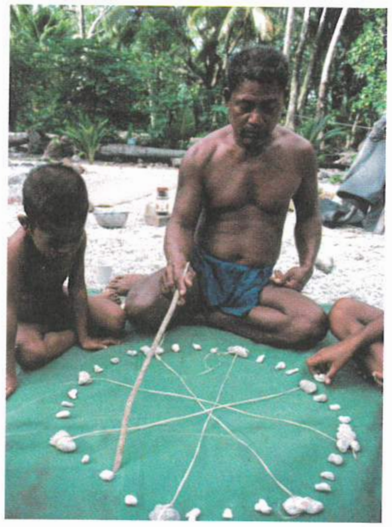


FIG 2.1: Mau Piailug using a star compass to teach navigation, as he was taught in his youth. Photograph by Monte Costa (http://bkaprt.com/rdpp/02-01/).

And those challenges compound themselves when you're designing responsively. How can a complex menu adapt to a smaller screen? What if we want to display more (or less) information depending on the dimensions of the display? Most critically, though, a responsive navigation system doesn't need to look or work the same at every breakpoint, but it does need to offer access to the same content across devices.

These questions might feel daunting, but they demonstrate why navigation is a great example of the *small layout systems* we're going to focus on in this book. The responsive design of a site's navigation poses an almost entirely different challenge than a page's top-level grid. In dealing with challenges of layout, interaction, and visual density, we're forced to ask ourselves: how can we design navigation that's as usable as it is responsive?

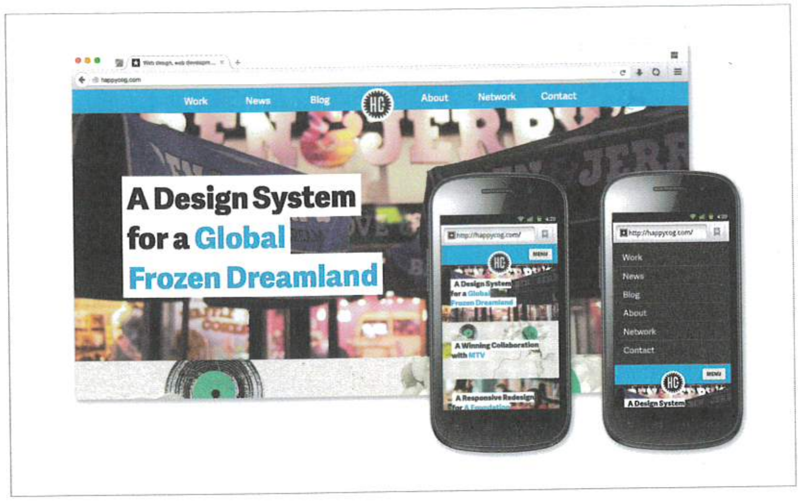


**FIG 2.2:** The responsive navigation for Happy Cog's site seen at sizes wide and small (http://bkaprt.com/rdpp/02-02/).

Thankfully, there are many great attempts at answering that question. In this chapter, we'll look at design patterns both common and not-so-common, and see if we can't find our way through the challenges of responsively designing navigation.

## THE SHOW/HIDE TOGGLE

Open up design agency Happy Cog's responsive site (**FIG 2.2**). On wider screens, the entire navigation is visible, but on smaller viewports, where screen real estate is at a premium, the top of the design only shows a Menu link. If you tap, click on, or select that link with your keyboard, the full menu appears.

This is one of the most common ways of handling complex navigation systems in a responsive design: when the menu doesn't fit, conceal it. This pattern requires two elements at minimum: the navigation, which is concealed at certain breakpoints; and a "trigger" element, which the user interacts with to

reveal the navigation. In fact, we took the same approach with the menu on responsivewebdesign.com (**FIG 2.3**). The design's fairly modest, but I'll briefly walk you through the code to demonstrate how this pattern's often implemented..

First, at the top of the page, we have this markup:

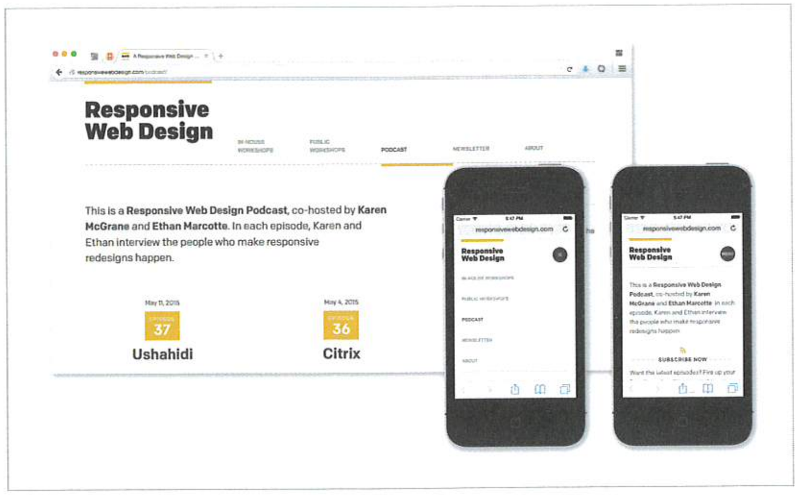
```
<div class="head">
  <h1 class="logo">
  <a href="/"><img src="/images/logo-rwd.png"
    alt="Responsive Web Design" /></a>
  </h1>

  <div id="nav" class="nav">
    <nav>
    <h1><a class="skip" href="#menu">Explore this
      site:</a></h1>

      <ul id="menu">
        <li><a href="/workshop/">Workshop</a></li>
        <li><a href="/events/">Public Events</a>
        </li>
        <li><a href="/podcast/">Podcast</a></li>
        <li><a href="/newsletter/">Newsletter</a>
        </li>
        <li><a href="/about/">About</a></li>
      </ul>
    </nav>
  </div><!-- /end .nav -->
</div>

<!-- [ The page's main content goes here. ] -->
```

I've simplified things a little, but there's not much more to it: the document leads off with our logo, a link to skip to the navigation, and then the navigation itself, marked up as an unordered list. But that HTML is, as you might have guessed, just the foundation. To enhance the menu further, let's begin with a simple JavaScript test:



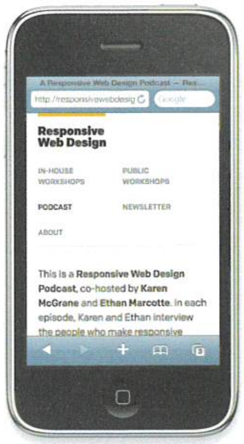
This is a **Responsive Web Design Podcast**, co-hosted by **Karen McGrane** and **Ethan Marcotte**. In each episode, Karen and Ethan interview the people who make responsive redesigns happen.

**FIG 2.3:** The responsive masthead for, uh, responsivewebdesign.com. Straightforward design, with a simple toggle to show (or hide) the navigation.

```
// Is this browser sufficiently modern to continue?
if ( !( "querySelector" in document
  && "addEventListener" in window
  && "getComputedStyle" in window) ) {
    return;
}

window.document.documentElement.className +=
  " enhanced";
```

We're asking the user's browser if it supports the DOM features we'll need elsewhere in our JavaScript—features like `document.querySelector`, `window.addEventListener`, and `window.getComputedStyle`. If they're not found, then that `return;` keeps the browser from executing the rest of our JavaScript. The result is that older browsers are left with a perfectly usable experience, albeit a less JavaScript-enabled one (**FIG 2.4**). And when those features are found, our JavaScript applies a class of `enhanced`



This is a **Responsive Web Design Podcast**, co-hosted by **Karen McGrane** and **Ethan Marcotte**. In each episode, Karen and Ethan interview the people who make responsive

**FIG 2.4:** No JavaScript? No problem: our navigation's still accessible, even to modern browsers that can't load our code.

to the HTML element (`window.document.documentElement.className += " enhanced";`).

Why run that test? Well, this JavaScript test lets us build our navigation with a pinch of progressive enhancement: we can design a simpler but *usable* experience that's universally accessible by default, and then enhance the experience *only* for browsers and devices that will actually benefit from it. If the test successfully runs, then the `enhanced` class on the HTML element tells us a given browser is receiving the "enhanced" experience.

This is a fairly common approach for responsive sites, especially at a certain scale. For example, the BBC News team built their responsive design upon a foundation of progressive enhancement (**FIG 2.5**), using a little JavaScript test similar to the one we've used above, which allows them to determine whether a browser "cuts the mustard" (http://bkaprt.com/rdpp/02-03/):

*We make [the browser landscape] manageable in the same [way] you and everyone else in the industry does it: by having a lowest common denominator and developing towards that. So we've taken the decision to split the entire browser market into two, which we are currently calling "feature browsers" and "smart browsers". ...as the [site] loads we earmark incapable*

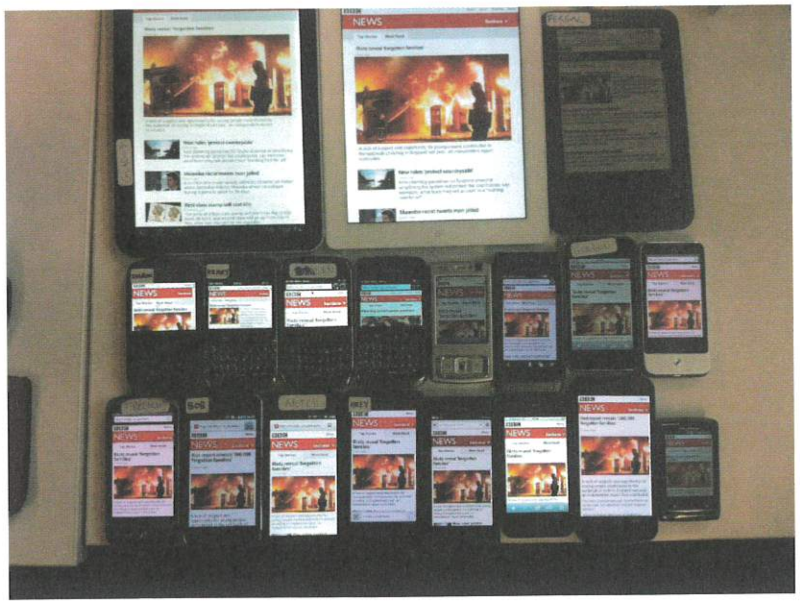


**FIG 2.5:** The BBC News site is accessible—and responsive—on every internet-connected device, but the experience is slightly enhanced on more modern browsers. Photograph courtesy Responsive News (http://bkaprt.com/rdpp/02-04/).

*browsers with the above code and exclude the bulk of the JavaScript-powered UI from them, leaving them with a clean, concise* core *experience.*

Instead of tracking myriad combinations of browsers and devices, the BBC can think of their design as existing in one of two broad *experience* tiers: a baseline responsive experience, and a slightly more advanced experience that's only served to the browsers that can handle it.

On a much smaller scale, that's exactly what we're doing with the navigation on responsivewebdesign.com. With that `enhanced` class in place, we can write more advanced styles

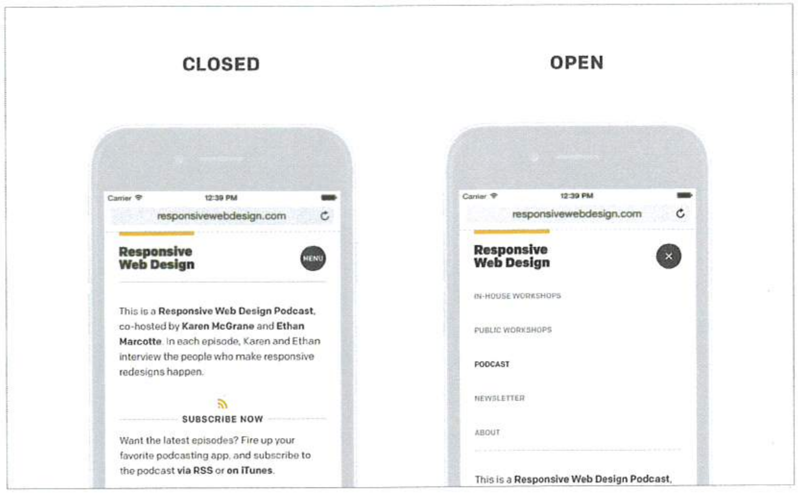
directed at the browsers that pass our test, and build the more advanced view of our navigation:

```
.enhanced .nav .skip {
  position: absolute;
  right: 0;
  top: 1.4em;
  background: #363636;
  border-radius: 50%;
  width: 2.5em;
  height: 2.5em;
}
.enhanced .nav ul {
  max-height: 0;
  overflow: hidden;
}
```

If a browser passes our JavaScript test, this rule will use plain ol' absolute positioning to take that skip link before our navigation—`.nav .skip`—and stick it at the top of the page. At the same time, by applying a pinch of `background: #363636` and `border-radius: 50%`, we can turn that link into a big, gray, circular button. But the second rule is where things get interesting: it selects the `ul` inside `.nav`—that is, the unordered list that contains our navigation links—and uses `overflow: hidden` and `max-height: 0` to turn the list into a `0px`-tall box, effectively hiding our links from view. Hiding them, that is, until a class of `.open` is applied to the list:

```
.enhanced .nav ul.open {
  max-height: 20em;
}
```

With those rules, we now have two states for our navigation: completely hidden and expanded (**FIG 2.6**). Sounds great and all, but you might be wondering how we'll get that class on our `ul`. Well, that's where a little more JavaScript comes into play:



This is a **Responsive Web Design Podcast**, co-hosted by **Karen McGrane** and **Ethan Marcotte**. In each episode, Karen and Ethan interview the people who make responsive redesigns happen.

Want the latest episodes? Fire up your favorite podcasting app, and subscribe to the podcast **via RSS** or **on iTunes**.

This is a **Responsive Web Design Podcast**,

**FIG 2.6:** Our CSS now allows us to show or hide the navigation. But how do we make it interactive?

```
var nav = document.querySelector( ".nav ul" ),
navToggle = document.querySelector( ".nav .skip" );

if ( navToggle ) {
  navToggle.addEventListener( "click",
    function( e ) {
    if ( nav.className == "open" ) {
      nav.className = "";
    } else {
    nav.className = "open";
    }

    e.preventDefault();
  }, false );
}
```

If JavaScript's not your thing, don't worry—the code's more straightforward than it looks, I promise. Remember that skip

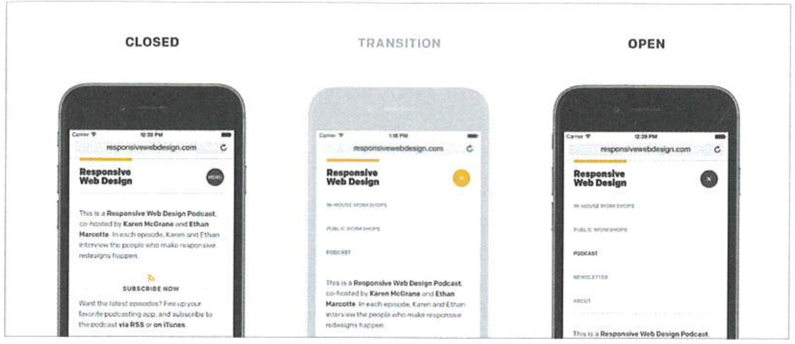


**FIG 2.7:** Why not include a little max-height transition, you say? Great idea!

link inside our `.nav` element? Well, we're using some JavaScript to look for it (`document.querySelector( ".nav .skip" )`) and, if it's found, add some functionality whenever it's clicked or tapped (`navToggle.addEventListener( "click", … );`). When a user taps or clicks on that link, our code checks to see if our unordered list has a class of `open` (`if ( nav.className === "open" ) { … }`). If it doesn't, the JavaScript adds the class to reveal the links; and if it *does*, it removes the class, and hides the navigation from view.

And if we want to get a little fancy—and of *course* we want to get a little fancy—we can add a CSS transition on the `max-height`, allowing the list to subtly telescope in and out of view (**FIG 2.7**):

```
.enhanced .nav ul {
  max-height: 0;
  overflow: hidden;
  transition: max-height 0.25s ease-out;
}
```

And we're done! With a little bit of JavaScript, we can show (or hide) an element of our design when it's clicked on, all by adding (or removing) a class.

(Quick aside: while `overflow` is a CSS property older than time, it's worth noting that an astonishingly high number of mobile browsers don't implement it correctly. If any part of your design uses `overflow: auto` to create scrollable areas, I recommend Filament Group's Overthrow.js library (http://bkaprt.com/rdpp/02-05/), which properly detects support for `overflow` while weeding out the browsers that claim to support the property but don't.)

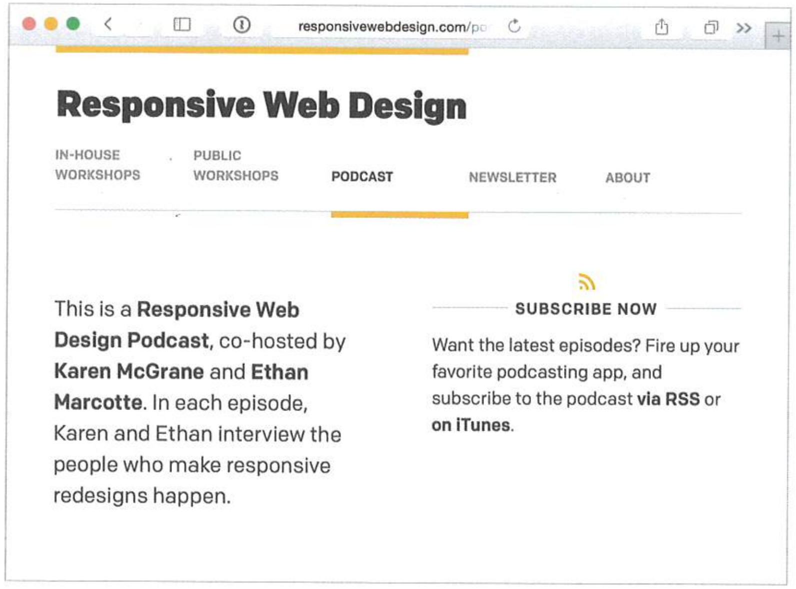
While the show/hide toggle works beautifully, that doesn't mean the effect's necessarily appropriate for *all* breakpoints. The toggle's really only valuable on smaller viewports, where the layout's a bit tighter; when the viewport gets wider, we can display the entire navigation, locked-up with the logo (**FIG 2.8**). But since the entire effect is driven by our CSS, we can override it above a certain breakpoint with a media query:

```
@media screen and (min-width: 39em) {
  .page .nav ul {
    overflow: auto;
    max-height: inherit;
  }
}
```

Now, when the viewport reaches a minimum width of `39em`, we've disabled the `overflow: hidden` on the list, and returning its `max-height` to a normal, default value. As a result, our list is no longer hidden from view, allowing us to style it like a more traditional masthead.

...phew!

That might seem like a lot of work, but we're simply adding or removing a class with a little JavaScript, and using that class to control the visibility of our navigation. And really, that's the basic mechanism of nearly all show/hide toggles. MSNBC.com's responsive site does this very thing, in fact (**FIG 2.9**): on widescreen displays, tapping or clicking on the primary categories reveals secondary menus; but on smaller displays, tapping on an icon reveals the entire navigation, with submenus also expandable within it.


responsivewebdesign.com/po

**Responsive Web Design**

IN-HOUSE WORKSHOPS · PUBLIC WORKSHOPS · PODCAST · NEWSLETTER · ABOUT

This is a **Responsive Web Design Podcast**, co-hosted by **Karen McGrane** and **Ethan Marcotte**. In each episode, Karen and Ethan interview the people who make responsive redesigns happen.

**SUBSCRIBE NOW**

Want the latest episodes? Fire up your favorite podcasting app, and subscribe to the podcast **via RSS** or **on iTunes**.

**FIG 2.8:** For wider screens, we can disable the show/hide toggle, and just keep our links in view.

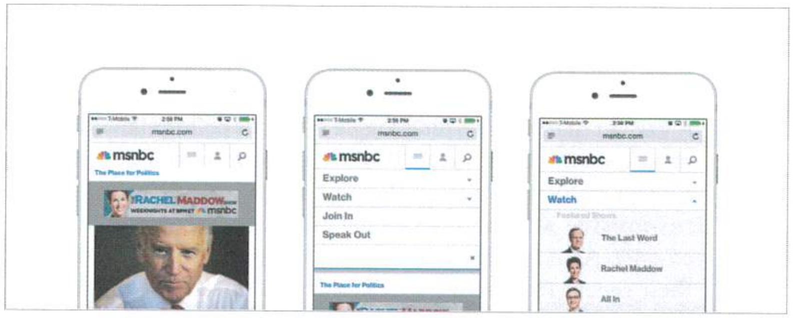


**FIG 2.9:** MSNBC's responsive navigation uses a top-level toggle to reveal its menu on smaller screens. Additionally, users can open nested menus by tapping or clicking on the relevant sections.

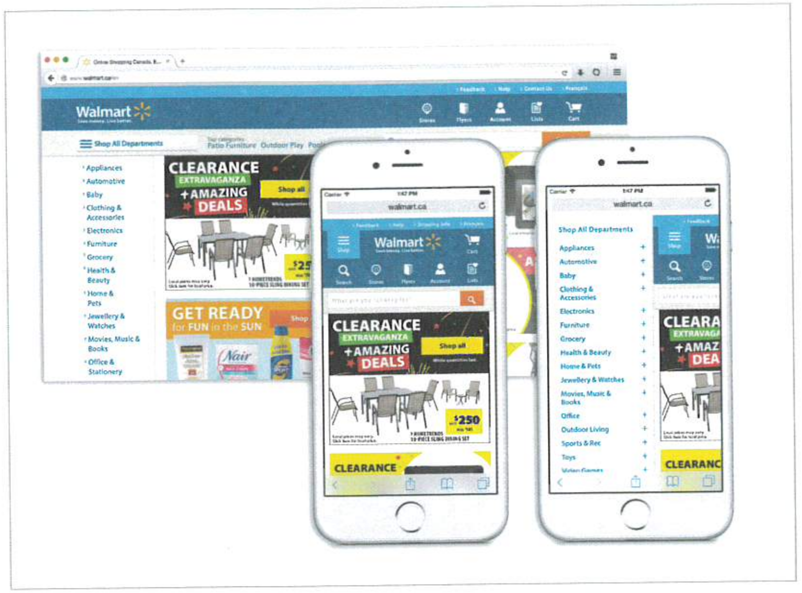


**FIG 2.10:** Walmart.ca's navigation is hidden "off-canvas" on narrower viewports, but visible by default on wider ones.

## THE OFF-CANVAS MENU

A variant of the show/hide toggle is what's colloquially referred to as the *off-canvas menu*. While this pattern first gained traction in native mobile applications, it's recently seen use in responsive and mobile websites (http://bkaprt.com/rdpp/02-06/). As it happens, Walmart.ca adopted this approach in their recent responsive redesign (**FIG 2.10**). On wider screens, the navigation's visible at the left. But on smaller screens, tapping or clicking on the Shop icon causes the entire navigation to *slide* in from the left, positioned just beyond the visible canvas.

From a mechanical standpoint, this isn't considerably different from our old, trusty show/hide toggle: we're still concealing our navigation, and then asking our users to interact with an element to toggle its visibility. If executed well, the off-canvas menu can convey an extra layer of depth and dimensionality in

your layout. It does, however, require extra care to make sure it's built accessibly, and that it doesn't break the experience for all but the latest browsers (http://bkaprt.com/rdpp/02-07/).

## CONDITIONALLY LOADED MENUS

FiveThirtyEight.com, an American news and entertainment site, launched a responsive redesign in 2014 (**FIG 2.11**). On smaller displays, their team opted for a show/hide toggle for their navigation. While they kept that toggle on wider viewports, links to their main article categories—Politics, Economics, Science, and so on—are pulled out of the hidden menu, and made visible by default. But there's something else at play here: on wider screens, tapping or clicking links in the masthead reveals a dropdown menu, teasing additional content from that section of the site (**FIG 2.12**).

Here's a high-level look at one item in FiveThirtyEight's navigation menu:

```
<ul class="menu">
  <li class="menu-item">
    <a href="http://fivethirtyeight.com">Menu</a>
    <div class="dropdown">
    <!-- Subnav content goes here -->

    </div>
  </li>
<!-- Subnav content goes here -->

</ul>
```

Each top-level link sits inside a list item (`li.menu-item`), which also contains a `div` with a class of `dropdown`. And those `div`s contain—you guessed it!—the dropdown menu that appears on wider screens. That's some straightforward markup, to be sure—but it provides a foundation for the CSS that displays the dropdowns on wider screens:

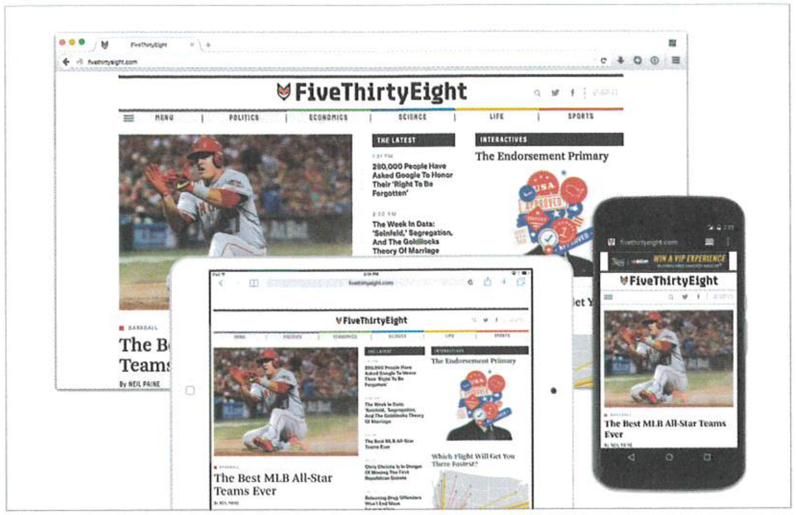


FIG 2.11: The responsive redesign for FiveThirtyEight.com. As news sites go, it's a stately, analysis-rich affair.

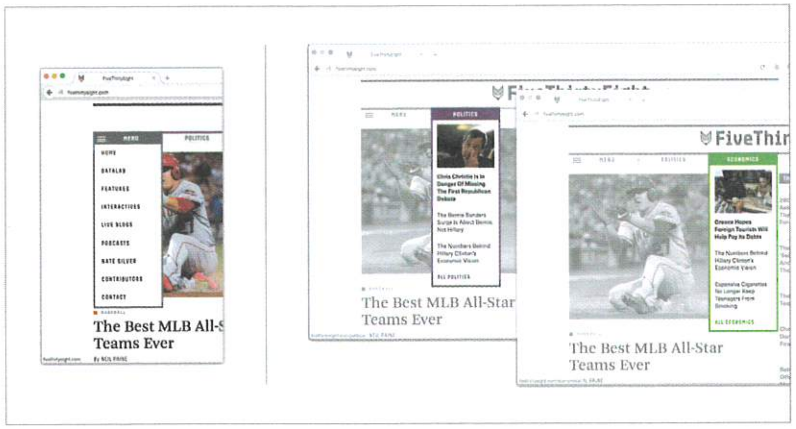


FIG 2.12: FiveThirtyEight's responsive navigation is a simple show/hide toggle on all displays, with key links promoted to the masthead on wider screens.

```
li.menu-item {
  position: relative;
}
.dropdown {
  display: none;
  position: absolute;
}
@media screen and (min-width: 768px) {
  li.menu-item:hover .dropdown {
    display: block;
  }
}
```

I'm simplifying FiveThirtyEight's styles a bit, but the underlying mechanics are still the same: on small screens, the `.dropdown` blocks are hidden by default with `display: none`; but by using a media query, they can reveal the menus on wider viewports when someone hovers over the containing list item (`li.menu-item:hover .dropdown`).

The approach FiveThirtyEight's using here is common, but not without its drawbacks. Relying on `:hover` is a potential liability, as the CSS assumes all widescreen devices are mouse-enabled. And there are plenty of devices that buck that trend, from tablets to touch-enabled laptops. But more broadly, there's a considerable drawback to using CSS to hide information on smaller screens: namely, that the browser will still download all the HTML for a hidden element, even if the styles hide it from view. In other words, the small screen users of FiveThirtyEight.com—and of other sites that use this `display: none` pattern—will be downloading extra data they won't use. And if your readers are on a metered data connection, that can be a potentially costly design decision.

A more responsible alternative would be to use conditional loading: to load that extra content only under certain conditions, ensuring it's only loaded on the screens that will use it. When the *Boston Globe*'s responsive site launched, the design team and I adopted a similar pattern for their masthead: on smaller screens, the entire site's navigation would be accessible by toggling its visibility (FIG 2.13). But as the design widened

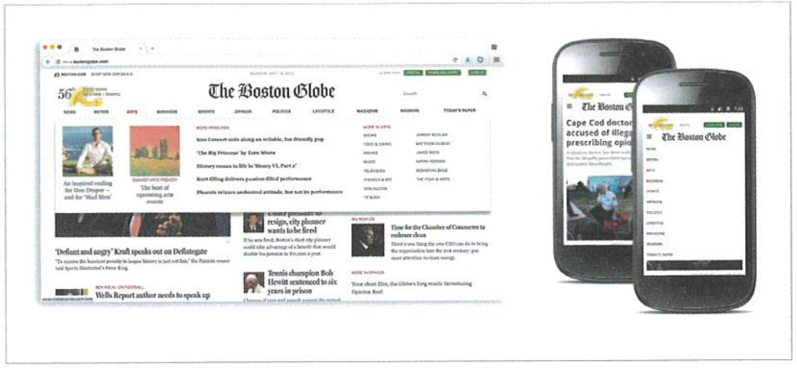


**FIG 2.13:** When the site first launched, the Boston Globe's responsive navigation toggled its visibility on smaller displays. But on wider displays, the navigation was visible—and included teasers for key stories from each section.

and there was more space to work with, we took advantage of the space to promote key stories in each section.

Most important, those panels aren't available on smaller screens—but their markup isn't included in the page by default, and then hidden with CSS. Instead, the extra HTML is conditionally loaded using a bit of JavaScript. Personally, I quite like Filament Group's Ajax-Include pattern (http://bkaprt.com/rdpp/02-08/) for managing conditionally loaded content:

```
<ul>
  <li data-append="/politics/latest-stories.html"
    data-media="(min-width: 39em)">
    <a href="/politics/">Politics</a>
  </li>
</ul>
```

The Ajax-Include pattern works by applying an HTML5 `data-` attribute to a part of your HTML, which describes where the conditionally loaded content should be placed (`data-before`, `data-after`, `data-append`, or `data-replace`); if you like, you can also specify a media query (via the optional `data-media` attribute) to note that the content should only

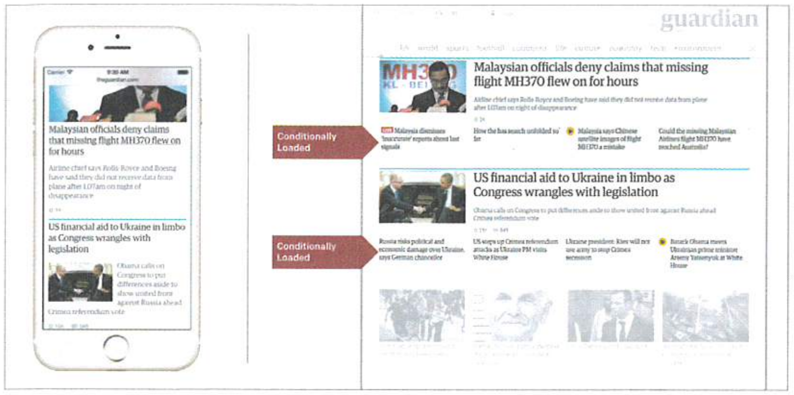


**FIG 2.14:** Certain modules on the Guardian's site used conditional loading to reduce information density across breakpoints.

be loaded *if* certain conditions are met in the client. So in the above snippet, the Ajax-Include JavaScript fetches the content of `/politics/latest-stories.html`, presumably just a snippet of HTML, and *appends* it to our list item—but only if the viewport has a minimum width of `39em`.

As you may have guessed, conditional loading isn't just handy for navigation—it can benefit other types of content as well. The *Guardian,* for example, had several kinds of conditionally loaded content on an earlier version of their site. When viewed on a wider screen, certain lead stories were accompanied by a row of related articles. But when you looked at the same module on a smaller screen, only the lead stories were visible. Now, those secondary stories weren't hidden with a bit of CSS. Instead of taking the performance hit of downloading the content on every device, the site loaded the related articles only if the user's browser supported JavaScript *and* was wider than a specified width (**FIG 2.14**). Otherwise, the most important content—the lead stories—was seen by everyone.

It's worth noting that conditional loading isn't about providing "desktop" users with more content, or "mobile" users with less. Rather, conditional loading can help us address problems of *density* in your design, ensuring that the information shown to our readers never overwhelms them. The Guardian and the Boston Globe have identified the content that's important to *all* of their users, and used that as the basis for both the wide- and small-screen views of each module.

In other words, it's not about removing or hiding extra information on smaller screens. Instead, try thinking about your content through this three-part framework:

1. Identify the content critical to the smaller screen.
2. Once you've done that, consider that content to be the information accessible to all your readers, regardless of how wide (or small) their screen happens to be.
3. If there's additional information you'd like to include on wider viewports, consider it an *enhancement*.

Adopting this "mobile first" mindset won't necessarily change the implementation, but it will inform the way you plan the design of your conditionally loaded content. It'll help avoid the trap of thinking of smaller screens as somehow deserving of "less" content—especially when our audiences are becoming increasingly (if not exclusively) mobile-oriented.

## HOUSTON, WE MAY HAVE A HAMBURGER PROBLEM

As you've seen in this chapter so far, there's no truly perfect way to manage navigation. And this isn't an exhaustive list of navigation patterns: everyone from Filament Group (http://bkaprt.com/rdpp/02-09/) to Mozilla (http://bkaprt.com/rdpp/02-10/) has weighed in on various approaches to building responsive, multi-device-friendly navigation systems.

But as distinct as these approaches are, many of them happen to share one element in common (**FIG 2.15**). Namely, an icon of three stacked, horizontal bars, colloquially—and perhaps unfortunately—referred to as the "hamburger": ≡.

According to Quora (http://bkaprt.com/rdpp/02-11/) and the BBC (http://bkaprt.com/rdpp/02-12/), our little hamburger icon

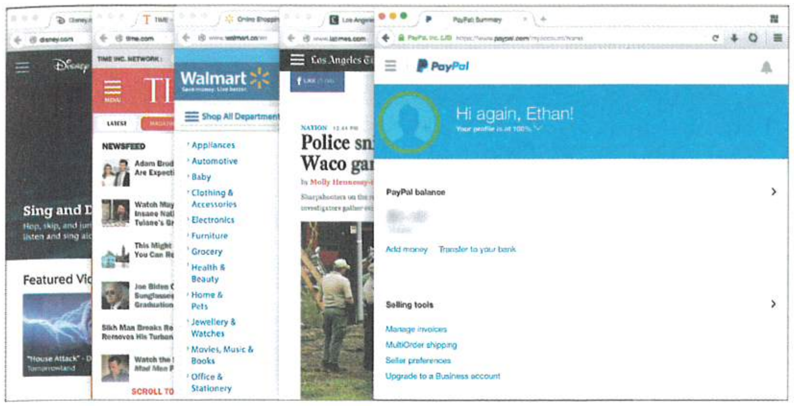


**FIG 2.15:** Behold the "hamburger": a common icon toggle for responsive navigation systems.

was first used digitally on the Xerox Star, a little workstation that launched in 1981 and established a number of standards for modern personal computing: a multi-button mouse, windows-based graphical interfaces, and the like (**FIG 2.16**).

At the time, the ≡ icon was used to open a contextually relevant menu. But more recently, it's been used as a trigger to reveal a site's entire navigation, especially at smaller resolutions. And there are some real benefits to working with the icon: it's incredibly compact, and is quite legible even on smaller screens; also, it's very easy to include it in your designs, whether using SVG (http://bkaprt.com/rdpp/02-13/), some fancy CSS-based animations (http://bkaprt.com/rdpp/02-14/), or a plain ol' HTML entity (`&#9776;`).

But as wonderful as the hamburger is, and as ubiquitous as it seems to be, I'm here to suggest that it might have some problems. And maybe we should talk about them a bit—preferably before we slap the icon on all of our responsive sites.

(Also, is anyone else hungry?)

LET'S BEGIN WITH the responsive website for *Time* magazine (http://bkaprt.com/rdpp/02-15/). Launched in 2014, it features a layout as flexible as its aesthetic is bold. (**FIG 2.17**). And while

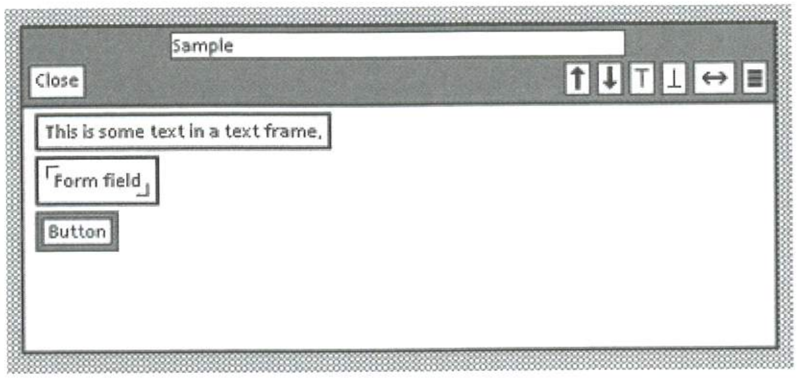


**FIG 2.16:** The hamburger's gotten a lot more mileage lately, but was first seen on screens in the early 1980s, as part of the Xerox Star's graphical interface (http://bkaprt.com/rdpp/02-11/).

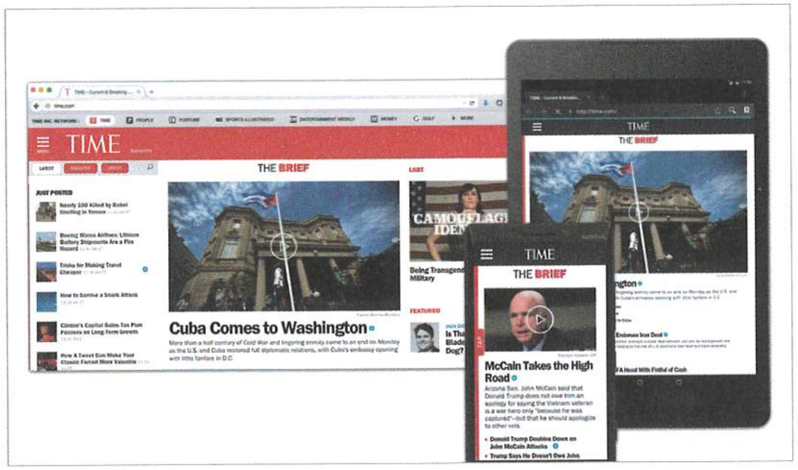


**FIG 2.17:** The newly responsive website for *Time,* as flexible as it is fashionable (http://bkaprt.com/rdpp/02-15/).

*Time*'s pages are packed with content, they never overwhelm: the design's simple palette and clear hierarchy allows the reader to quickly identify the stories most relevant to her, regardless of how small—or large—her screen might be.

One aspect of the redesign I find especially impressive is the sheer amount of navigation on each page. (I should note I've never been accused of being especially "cool.") There are, as best as I can tell, four separate navigation elements on the homepage:

1. On wider viewports, a menubar appears at the top of the page, allowing the user to leap to other sites in the Time Inc. network (**FIG 2.18**).
2. The page's footer contains a list of links to core sections of the website (**FIG 2.19**).
3. Supplemental content appears on the left edge of the page. On the homepage, for example, you see a feed of recent stories, a stock ticker, and *Time*'s site search. But that's just the *widescreen* view of this part of the design. On smaller screens, there's a tab labeled "TAP" that allows the user to open the content as a panel, which covers the page. When they're finished, they can tap or click on the tab to close it (**FIG 2.20**).
4. The primary navigation is concealed until a user taps or clicks on the hamburger icon. On smaller viewports, the navigation covers the entire design. On wider screens, the drawer covers the left side of the page (**FIG 2.21**).

Sounds like quite a lot, doesn't it? But again, I think *Time* balances the density well. Noisier navigation menus are concealed when space is at a premium, and as the screen gets progressively wider, each menu is shown by default *only when there's sufficient space to do so*. But the one navigation element that's *always* hidden on all breakpoints is, interestingly, the site's primary navigation. Yes, that's right: it's tucked behind our beloved hamburger icon.

In fact, when the responsive TIME.com first launched, the hamburger was treated in a fairly novel way: when the page loaded for the first time, an overlay appeared next to the icon, informing the reader that she could use the icon to reveal the site's navigation. What's more, if her browser had a mouse, she

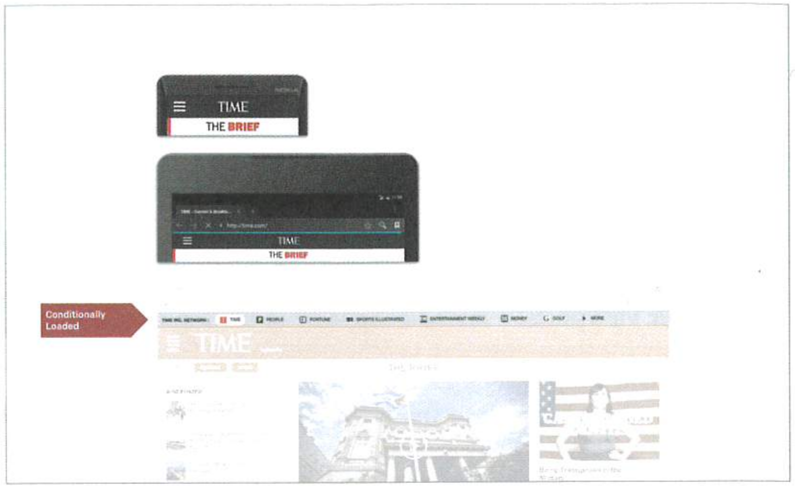


**FIG 2.18:** The menu at the top of the page is conditionally loaded on wider viewports, treating it as a widescreen enhancement.

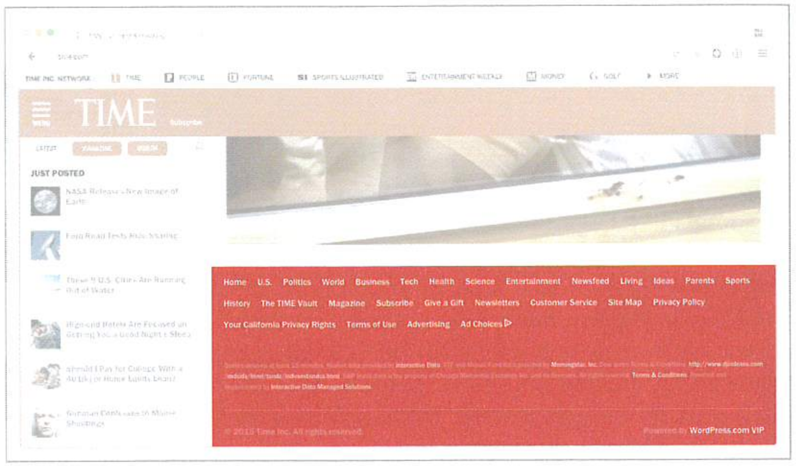

TIME
JUST POSTED
Home U.S. Politics World Business Tech Health Science Entertainment Newsfeed Living Ideas Parents Sports
History The TIME Vault Magazine Subscribe Give a Gift Newsletters Customer Service Site Map Privacy Policy
Your California Privacy Rights Terms of Use Advertising Ad Choices
Powered by WordPress.com VIP

**FIG 2.19:** Behold *Time*'s footer. Dense, perhaps, but rich with relevant links.

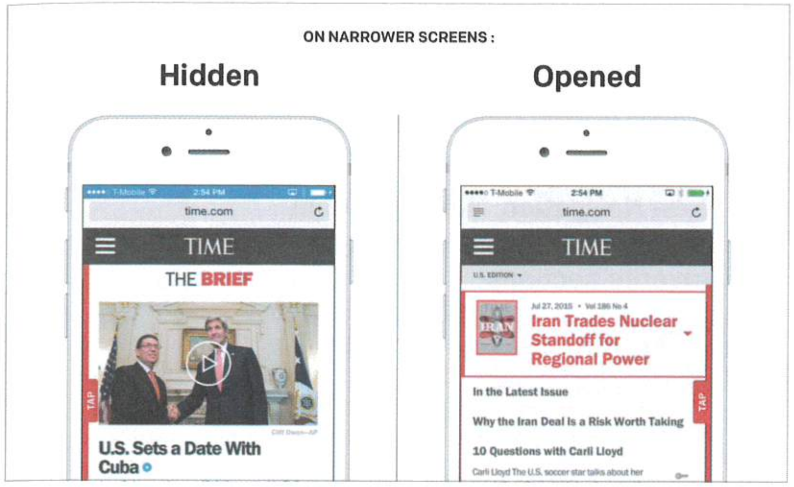

ON NARROWER SCREENS:
Hidden
Opened
time.com
TIME
THE BRIEF
TAP
U.S. Sets a Date With Cuba
Iran Trades Nuclear Standoff for Regional Power
In the Latest Issue
Why the Iran Deal Is a Risk Worth Taking
10 Questions with Carli Lloyd

**FIG 2.20:** Less critical content is placed in a drawer that's shown by default on wider screens. But on smaller screens, it's concealed by default until a user taps or clicks to reveal it.

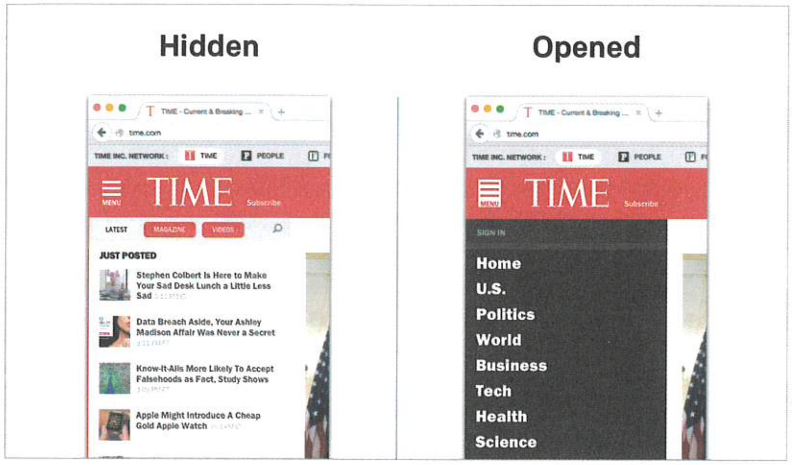
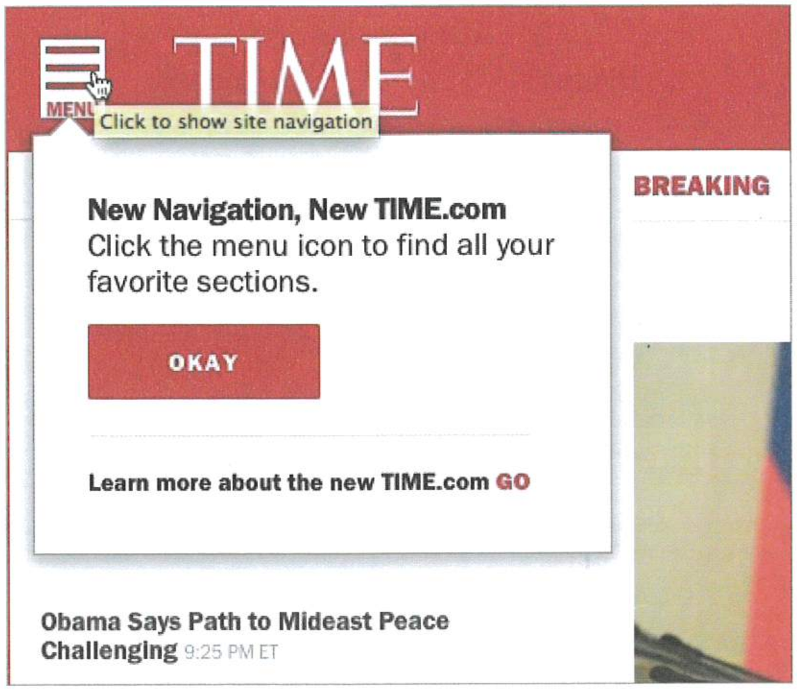

Hidden
Opened
TIME
JUST POSTED
Stephen Colbert Is Here to Make Your Sad Desk Lunch a Little Less Sad
Data Breach Aside, Your Ashley Madison Affair Was Never a Secret
Know-It-Alls More Likely To Accept Falsehoods as Fact, Study Shows
Apple Might Introduce A Cheap Gold Apple Watch
Home
U.S.
Politics
World
Business
Tech
Health
Science

**FIG 2.21:** *Time*'s primary navigation is the real showpiece, hidden behind a hamburger icon.



Click to show site navigation

New Navigation, New TIME.com
Click the menu icon to find all your favorite sections.

Learn more about the new TIME.com GO

Obama Says Path to Mideast Peace Challenging 9:25 PM ET

**FIG 2.22:** The *Time* hamburger, with helpful hints hovering overhead.

could also hover over the icon for a helpful tooltip informing her she could "Click to show site navigation." (**FIG 2.22**). In other words, the icon had *three* separate levels of text explaining it: the "Menu" label, the overlay, and the tooltip. While I don't have any insights into the redesign, this degree of explanation suggests a couple of possible motives: either the site's stakeholders weren't confident in the icon's ability to clearly identify itself as a critical navigation element, or the hamburger icon didn't perform well in usability tests before launch.

Once again, I'm conjecturing. But if *Time* did uncover usability issues with their use of the hamburger icon, they wouldn't be the first to do so. Designer James Foster ran a thorough usability study on a large site's responsive navigation, and found

that the word "Menu" consistently outperformed the icon alone, with a 12.9% higher conversion rate (http://bkaprt.com/rdpp/02-16/). As a result, his team abandoned the hamburger icon, and moved to a slightly more verbose trigger for their responsive navigation. It's not just happening on websites, either. The designers of Beamly's native app discovered that ditching hideable drawers in favor of always-visible navigation dramatically improved user engagement and satisfaction (http://bkaprt.com/rdpp/02-17/).

This isn't to say that our ubiquitous little icon can't be successful. (Heck, it's presumably performing quite well for *Time*.) In fact, the UX team at Booking.com recently surveyed their users and found the hamburger icon worked just fine for their site *and* its audience (http://bkaprt.com/rdpp/02-18/). In fact, changing the icon to the word "Menu" had no significant impact on their users' behavior.

So some sites say the hamburger's no good for them, while others say it's perfectly fine—what gives? In the face of some seemingly incompatible results, I think this demonstrates that the hamburger icon, like all design patterns, is worth testing on *your* site. When he shared his defense of the hamburger icon, Booking.com's Michel Ferreira said it best:

> *There is a lesson here for all of us on the nature of A/B testing. You are never solely testing a UI element, pattern, or feature. You are testing these things against a very specific user base in a very specific scenario. What works for Booking.com may not work for you and your users. This is the reason we A/B test in the first place, because the findings of others...are all unproven until they've been tested against our customers, on our platform.*

Beautifully said. There's nothing inherently wrong with the hamburger icon itself—but assuming it's a safe default for *every* responsive navigation system can be problematic. After all, what works for one site may not work for yours. So by all means, hamburger your sites! Just be sure to test those hamburgers before serving them up to your audience.

## THE DRAWER DILEMMA

But let's put our icon issues aside for a moment. Because there's another, possibly larger problem with our ability to conceal navigation.

Regard the responsive Disney.com, launched in 2012. And as responsive looks go, it's, well, lovely: replete with lavish images, videos, and content from Disney's various companies, it's an immersive, well-presented piece of responsive design, especially for such a well-known brand (**FIG 2.23**). And as with most responsive sites these days, they decided to hamburger their navigation. On smaller and midsize breakpoints, tapping on the hamburger reveals their site navigation which, if you'll count, contains every link ever created on the World Wide Web (**FIG 2.24**). I saw a Geocities page I designed in 1998 somewhere in there, three or four levels deep.

...okay, I'm trolling. (Sorry, Mickey.) But I hope it's clear I'm trolling out of *love*. Because while the layout and aesthetics of Disney's navigation are skillfully executed, their design illustrates a larger problem with concealing navigation: that, given the option to hide them, our menus can easily become filled with an overwhelming (and perhaps unhelpful) number of links. In discussing common design issues in apps for iOS, Mike Stern, a user experience evangelist at Apple, covered a number of the issues with hidden navigation drawers (http://bkaprt.com/rdpp/02-19/). While most of his design critiques are most pertinent to iOS apps, his last point is relevant to any digital designer, whether native- or web-focused:

> *And finally, the downside of being able to show a lot of options is that you can show a lot of options. Is that you will show a lot of options. The potential for bloat and misuse is tremendous... Look, drawers of any kind have a nasty tendency to fill with junk.*

I couldn't agree more. Like many responsive sites that collapse their navigation, Disney's navigation drawer is visually lovely, but suffers from an overabundance of content. That's why conditionally revealed navigation patterns work best when

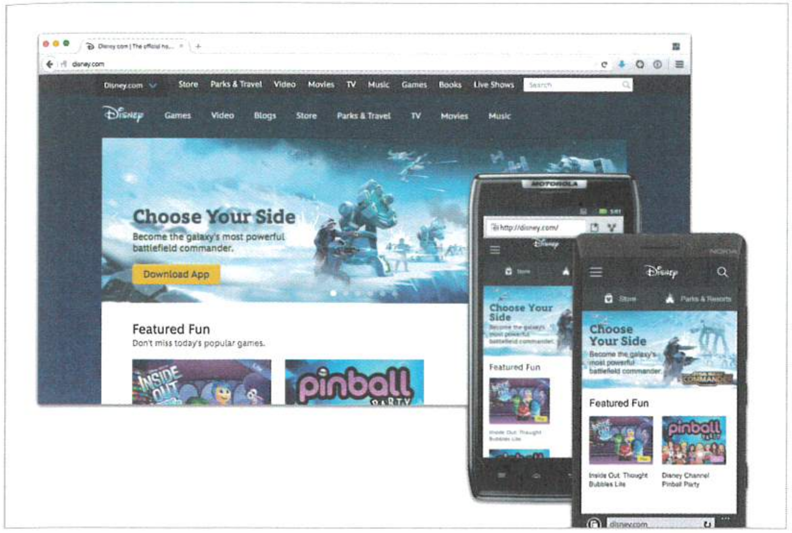


**FIG 2.23:** Disney.com: responsively redesigned, and gorgeously so.

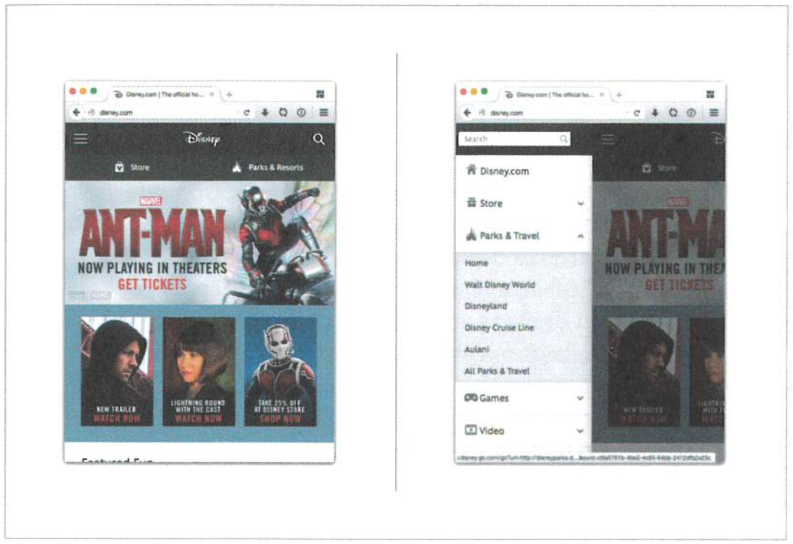


**FIG 2.24:** Disney's gradual reveal of their navigation, seen here *avec* hamburger.

paired with an almost aggressive curation of the content inside them. In fact, in his original essay on "mobile first," Luke Wroblewski notes that beginning a design project with smaller screens is a boon to products—and to their users (http://bkaprt.com/rdpp/02-20/) (emphasis mine):

> *Mobile devices require software development teams to focus on only the most important data and actions in an application. There simply isn't room in a 320 by 480 pixel screen for extraneous, unnecessary elements.* You have to prioritize. *So when a team designs mobile first, the end result is an experience focused on the key tasks users want to accomplish without the extraneous detours and general interface debris that litter today's desktop-accessed Web sites. That's good user experience and good for business.*

"You have to prioritize." That's the key point—we should use small screens as a lens through which we view every aspect of our designs, including our navigation. And if we're collapsing or hiding something because it doesn't "fit," let's instead see that as an opportunity to stop and ask if there's a larger issue at play: that is, if we're hiding or removing an element because it doesn't have value on smaller screens, can we simplify the design and content of that element until it works on smaller screens? Or, alternately, maybe it doesn't have value for *any* screen?

And really, I think that's the primary reason it feels so difficult to work with responsive navigation systems: they're often designed from a desktop-first mindset, and we're left to make them "fit" on smaller screens. But if our users are opening our navigation drawers and finding all the junk we didn't want to sift through in our redesign, is that show/hide toggle really benefiting anyone?

To be clear: I think the ability to conditionally conceal parts of a design is incredibly useful, especially for navigation. Whisking away unnecessary information and features can reduce the cognitive load on our users, and make our sites more approachable. But that useful ability is also easily *mis*used. If we're truly designing mobile-first, we shouldn't use show/hide toggles to

sidestep the potentially difficult discussions about the real value of our content.

After all, if our audiences are becoming predominantly mobile, we should stop trying to make complex, widescreen-designed elements play nice on smaller screens—instead, we should consider the small-screen user's needs first.

## ALTERNATIVE PATTERNS

Of course, there's real value in using widely-adopted design patterns. If a symbol like the hamburger icon is familiar to your users because they've used it elsewhere, it can lower the barrier of entry, and make your navigation more intuitive to them. And that's not something we should devalue lightly: the familiarity of an element can be a powerful benefit, both to our sites and to our audiences.

However, we shouldn't evaluate the utility of design patterns on their ubiquity alone. Patterns are, after all, just patterns: they're not rules or defaults. In fact, there are some rather novel alternative navigation patterns out there. Let's take a look at a few of them.

### The progressive reveal

The visual state of our responsive navigation is often treated as somewhat binary: it's either entirely visible or completely hidden. But some sites are challenging that approach, and designing navigation systems that make the best possible use of the space available to them.

The BBC has been experimenting with responsive design in public for some time now, redesigning the m-dot site for BBC News to be completely responsive. While their responsive design was initially accessible only to their mobile audience, it eventually became the default experience for all their users—whether on mobile, tablet, desktop, or anything else (http://bkaprt.com/rdpp/02-21/; **FIG 2.25**).

The site has two levels of navigation: global navigation at the top for other sites in the BBC network, and then a menu below

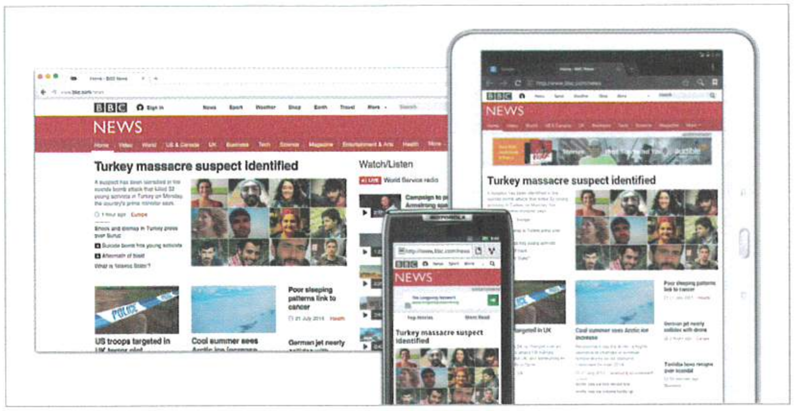


**FIG 2.25:** Flexible, fast, and resilient: the responsive site for BBC News began as a mobile-only site, and then became the default experience for all of their users.

it for BBC News-specific links (**FIG 2.26**). I think it's safe to say that neither menu is light on content, but rather than simply concealing their navigation, they've adopted a *progressive reveal* pattern for it. At the smallest level, each menu contains a single element that toggles the display of additional links: the topmost menu has a "More" link, while the BBC News navigation has a hamburger-enabled "Sections" link.

But as both navigation menus gradually widen, things get interesting: instead of simply moving the trigger around, each navigation bar gradually reveals links from its hidden panel. Anything that's still hidden is accessible by tapping or clicking on the "More" link or the "Sections" button—and as the design gets wider still, more links are gradually, *progressively* revealed. So while the global navigation menu might show "News" and "Sport" links at a smaller breakpoint, a slightly wider viewport might then promote "Weather," "Shop," and "Earth" as well (**FIG 2.27**).

As you may have guessed, this is a JavaScript-driven solution. When the page loads, resizes, or gets reoriented in a handheld browser, the BBC measures the width of the browser's viewport and then, based on its width, shows or hides certain links in each menu. The panels that expand when you tap or click on

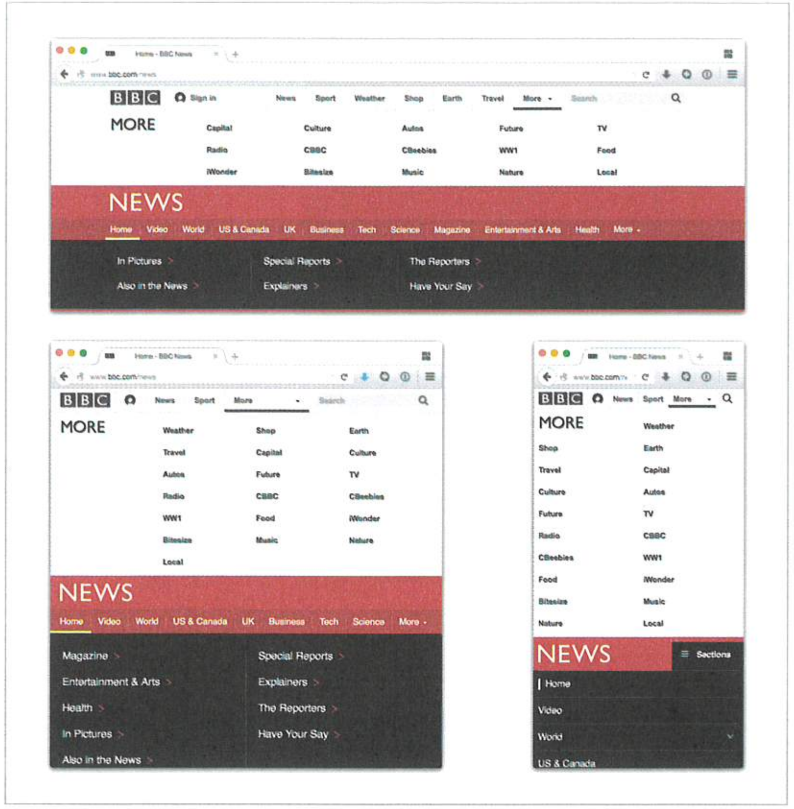


**FIG 2.26:** The two levels of navigation on the BBC News site, viewed at a few different breakpoints.

"More" or "Sections" are JavaScript-generated too, populated with any links that happen to be hidden in the parent menu at that breakpoint.

The *Guardian* uses a similar approach for their navigation. While their menu relies more on CSS than JavaScript, it embraces some of the same principles of progressively revealing links over time. In a post on their developer blog, product manager Chris Mulholland described the menu as a hybrid solution: one that combines the show/hide toggle seen on most

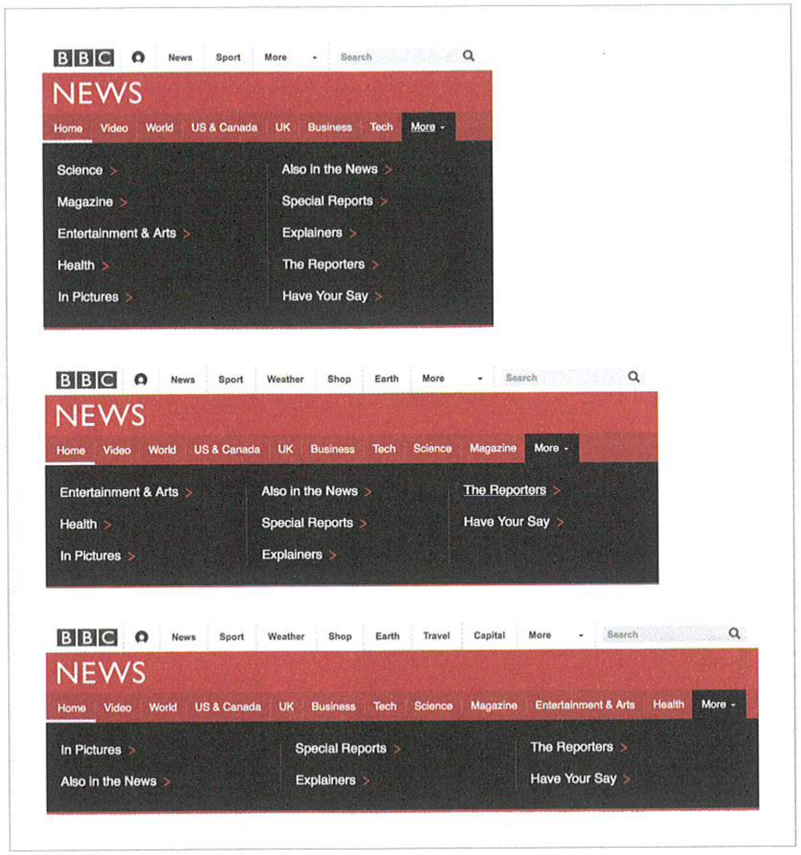


**FIG 2.27:** As with the global navigation, the BBC News-specific menu progressively reveals links over time—while ensuring the remainder are always accessible from the "More" dropdown.

responsive sites, while still keeping key elements visible at all times (http://bkaprt.com/rdpp/02-22/):

> *We have prioritised the signposting at smaller screen widths, but the side-scrolling allows you to access any sibling or top-level section.*

> *If side-scrolling isn't your thing, the All Sections is a familiar 'safety-net', giving you access across the site, starting with the section you are in.*
>
> *We have also tried to make it easier to navigate through the subsections, for example in Culture you are always shown the next section in the sequence. The links loop around almost like a carousel—making it easier to click through the sections.*

Rather than simply concealing the navigation entirely, the *Guardian* opted to present top-level categories in a scrollable container (**FIG 2.28**). As you might imagine, this scrollable area gets quite small on narrow viewports, but still allows the user to move through the carousel-like layout to find information most relevant to her. But if she doesn't find what she's looking for, regardless of viewport sizes, there's an "All Sections" link, which reveals a complex, multilevel navigation structure for the entire site (**FIG 2.29**).

Generally speaking, the navigation's structure doesn't change considerably as the design reshapes itself. You can view the site on the narrowest smartphone or the widest flatscreen display, and you're still left with the same scrollable region, with the expandable menu to the right. Given the other navigation patterns we've looked at in this chapter, the *Guardian*'s is almost novel in its consistency.

## Becoming more responsive

Putting the technical details aside, what I especially like about the *Guardian*'s navigation is the process of *how* they developed it. If we return to Chris Mulholland's overview of the navigation, he credits their menu design to three factors: a close analysis of their design goals; rapid iteration of a number of possible solutions; and—perhaps most important—involving their users as early as possible in the design process (http://bkaprt.com/rdpp/02-22/):

> *We know that as much as we can guess and assume what our readers want, there is nothing better than putting prototypes in front of them as early as possible. From the* Guardian *web-*

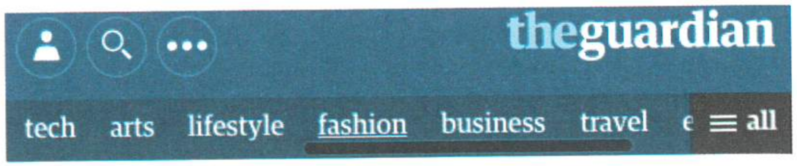


**FIG 2.28:** The *Guardian* has adopted a different kind of progressive reveal for their navigation.

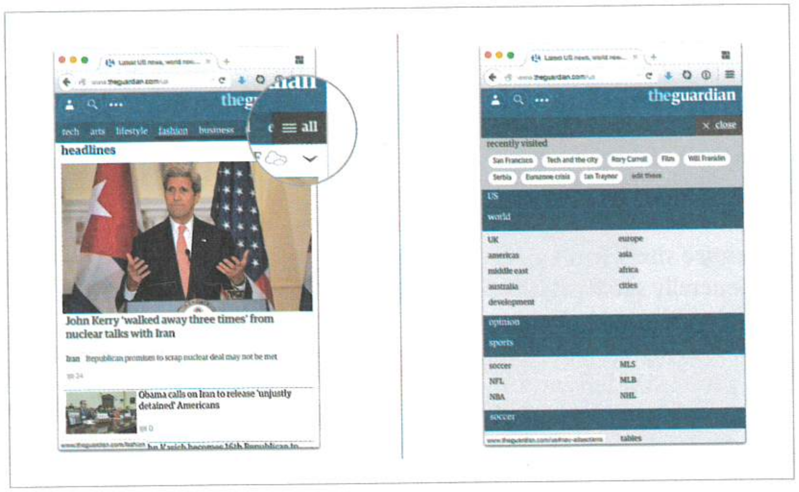


**FIG 2.29:** On viewports small or wide, the *Guardian*'s navigation features a toggle to display a complex map of the site's structure.

> *site, we invited daily visitors to join a panel of users to help us create the new website and provide feedback on ideas. Sending through low-fi prototypes to that panel of users was really valuable...[Next] we developed a prototype that would work with content and was completely realistic...User testing gave us some surprising results and took us down some other paths, but in the end we had much more confidence that regular and new users could navigate to both the most obvious, and the most hidden sections.*

In describing his work on the design of Virgin America's new responsive site, Joe Stewart, partner at the design agency Work & Co, said something similar—namely, that prototyping didn't just shape the design process, it *was* the design process. What's more, the client was never shown a static mockup of a web page (http://bkaprt.com/rdpp/02-23/):

> *Prototyping is basically our number one tool. So our philosophy on how to go about a project like [Virgin America] is to race to a prototype as quickly as possible. We actually never really made a presentation ever once, but we did constantly work on making prototypes. So, even the very first time we got to meet Dean and the Virgin America team, we showed them a responsive prototype.*

In recent years, there's been considerable discussion about whether or not we should start "designing in the browser" by leaping right into HTML and CSS—not just to prototype layouts, but to actually begin our creative work directly in the browser. And there are real benefits to that: the browser is, after all, a completely flexible canvas, and no desktop design application currently exists that can match its inherent responsiveness.

I agree with these statements—up to a point. After all, I think the approach has to be paired with the designer. If you're more comfortable thinking in HTML and CSS, great! But if you happen to work more quickly in a traditional design application, there's no reason to abandon a tried and trusted app for your nearest code editor. Instead, it's more important to acknowledge that each tool has strengths and weaknesses, and whichever path gets you to a responsive design first is the one you should take. Joe Stewart said as much during the Virgin America interview:

> *I still use Photoshop just because it's what I'm fastest in. I know a lot of people are switching to things like Sketch, which seems great, but for me it's just not the fastest. My design partner, Felipe, uses Illustrator for everything because that's what he's fastest in. I think it doesn't really matter how you get there. If you can get something that you can put in a person's hand*

*and get feedback, that's the goal. However you get there is how you get there.*

As much discussion as there is around "designing in the browser," we're not talking about the end of comps as a design tool. As the Virgin America redesign showed, applications like Photoshop and Sketch are still invaluable for sketching, for thinking about layout, for refining and discussing aesthetics. Instead, I think we're seeing the lessening importance of comps as an *end point:* as a client-facing design document or the definitive deliverable. Digital agencies and design teams still use Photoshop or Illustrator mockups to discuss aesthetics or composition options—but our industry still lacks a design tool that reflects the instability of the networks we design for, the various interaction modes available on our users' devices, and the flexibility of the web's canvas.

Personally, I share my colleague Dan Mall's take—we shouldn't necessarily be concerned with designing in the browser, but *deciding* in the browser (http://bkaprt.com/rdpp/02-24/). If you're comfortable sketching in Illustrator or prototyping interactions in Keynote, you should continue to do so. But as the *Guardian*'s Mulholland said, the more quickly you can get your designs into devices and browsers that you, your clients and stakeholders, and your users can hold and interact with, the better:

*We know that as much as we can guess and assume what our readers want, there is nothing better than putting prototypes in front of them as early as possible.*

Truer words were never hamburgered—I mean, *spoken*. Whenever possible, we should prioritize prototypes over Photoshop documents. As valuable as comps can be—and they are valuable—there's no substitute for reviewing prototypes "live" in as many browsers and devices as possible. They'll help you vet your design assumptions, and verify you're on the right path.

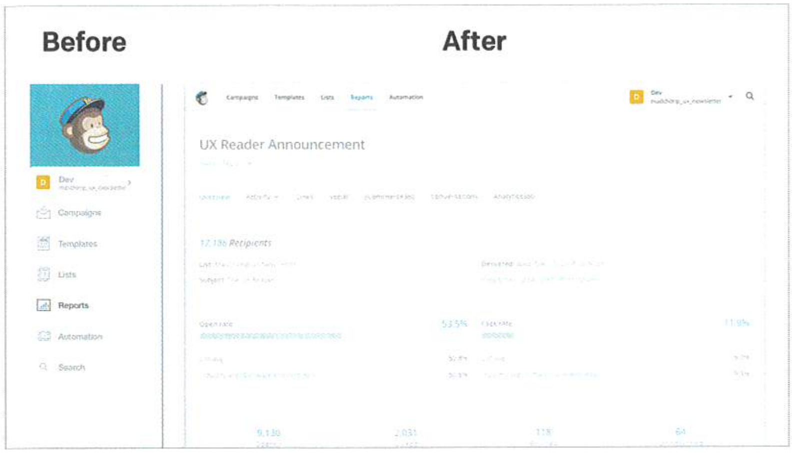
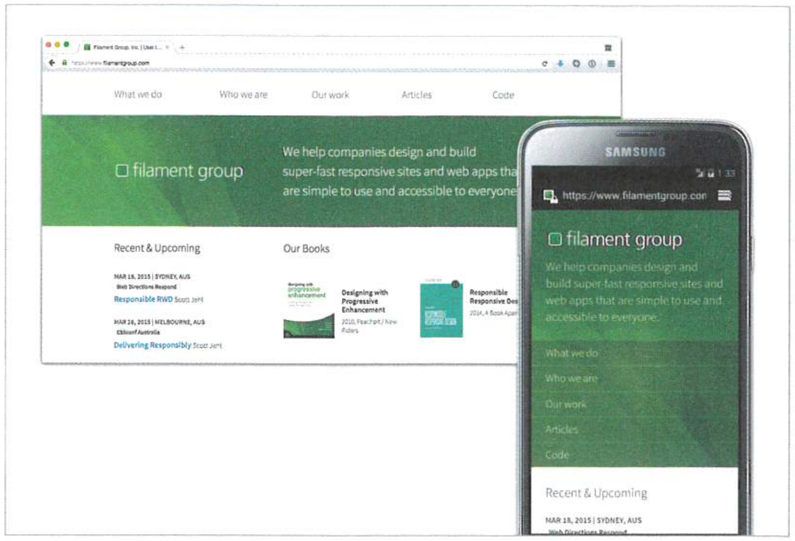


**FIG 2.30:** The navigation for MailChimp's application, before and after its simplification (http://bkaprt.com/rdpp/02-25/).

## Adapt the layout

Even with the best intentions, plans, and processes, our design assumptions don't always play out. When that happens, we often need to revisit parts of our work—and this is especially true of responsive navigation systems. The *Guardian* team landed on their current approach after a considerable amount of iteration. (And I'm sure they'll continue to refine it, too.) Email provider MailChimp found that their web app's responsive navigation, featuring a fixed toolbar, often obscured other interface elements (http://bkaprt.com/rdpp/02-25/). Simplifying the layout didn't just fix those issues—it dramatically improved the menu's usability (**FIG 2.30**).

Perhaps more important, MailChimp's work suggests an alternative to the navigation patterns we've reviewed thus far: rather than taking a complex approach, maybe we should look for opportunities to do less. In fact, we don't always need to



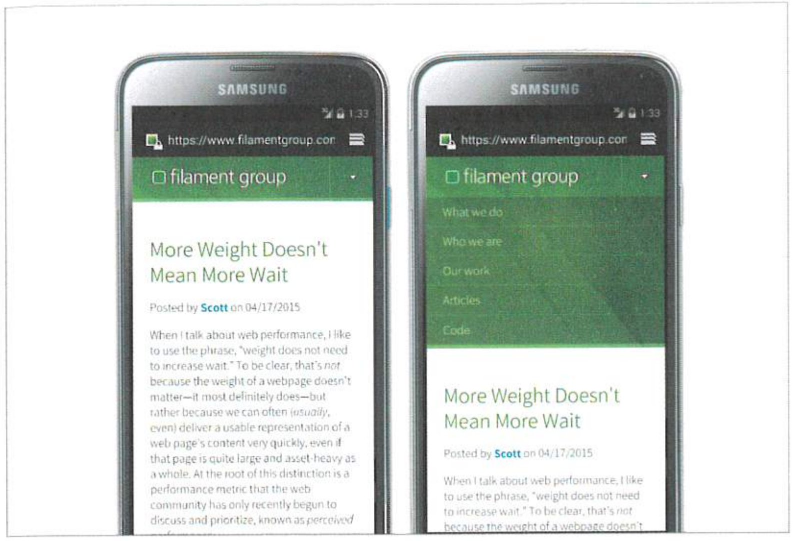
We help companies design and build super-fast responsive sites and web apps that are simple to use and accessible to everyone.

**FIG 2.31:** Filament Group's navigation isn't just attractive—it's never hidden from view on the homepage (http://bkaprt.com/rdpp/02-26/).

hide or conceal our navigation, when simply changing its layout can be incredibly effective.

Filament Group's responsive site does this admirably. On the homepage, their navigation's never hidden. On wider screens, it's located at the top of the page—but on narrower viewports, the menu's links are stacked directly underneath the company's logo and tagline (**FIG 2.31**). Given how focused their navigation is, this feels like a natural choice: the links can provide valuable signposting, so they're treated as first-class citizens.

As you'll notice, that's *just* on the homepage. On internal pages, Filament adopted a show/hide toggle on smaller screens, allowing them to conserve some much-needed space (**FIG 2.32**). Once again, I think this is a fine choice: since the inner pages' content should be the primary focus, moving the navigation behind a collapsible element makes sense. More important, it neatly demonstrates that patterns don't have to be universally applied throughout a site. Instead, we can be selective and



More Weight Doesn't Mean More Wait

Posted by Scott on 04/17/2015

When I talk about web performance, I like to use the phrase, "weight does not need to increase wait." To be clear, that's *not* because the weight of a webpage doesn't matter—it most definitely does—but rather because we can often (*usually*, even) deliver a usable representation of a web page's content very quickly, even if that page is quite large and asset-heavy as a whole. At the root of this distinction is a performance metric that the web community has only recently begun to discuss and prioritize, known as *perceived*

**FIG 2.32:** On internal pages, Filament Group uses an expandable menu link to conceal their navigation on smaller screens, allowing the content to take center stage.

nuanced in deciding how, where, and why we use those patterns in our work.

Frank Chimero's responsive design is one of my favorites—because, well, *look at it*—but I especially love how he's adopted a similarly reserved approach for his navigation (**FIG 2.33**). No matter how massive or microscopic your screen might be, the navigation's never hidden from view. What's more, Frank spent a considerable amount of time ensuring the navigation doesn't just fit; it feels at home.

Now by most menus' standards, both of these sites' navigations are fairly lightweight—just a handful of links, really. But that's perhaps where they gain a bit of flexibility. A more comprehensive set of links might need a heavier touch, and require adopting a complex design pattern—even (gasp!) a *hamburger!* But instead, I think the lighter touch they've adopted comes not out of their clever design, but out of their focused, distilled content.

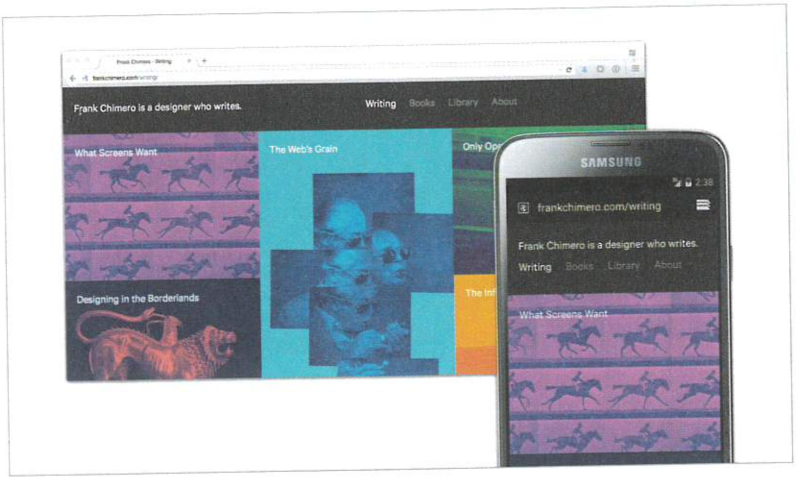


**FIG 2.33:** Frank Chimero's stunning responsive site, featuring a nav that never quits—or hides (http://bkaprt.com/rdpp/02-27/).

That's not to say responsive navigation systems can't be elegant *and* complex, as the BBC and the *Guardian* have shown. But I suspect that with all the challenges we face on the web, we should constantly search for opportunities to simplify our interfaces. If our responsive navigation can do that, we'll be in a better position to show our users the way.