



Moving to Responsive Web Design

Bring existing static sites into today's
multi-device world with responsive
web design

—

Inayaili de León

Apress®

www.allitebooks.com

Moving to Responsive Web Design

Bring Existing Static Sites into
Today's Multi-Device World with
Responsive Web Design



Inayaili de León

Apress®

Moving to Responsive Web Design: Bring Existing Static Sites into Today's Multi-Device World with Responsive Web Design

Inayaili de León
London, United Kingdom

ISBN-13 (pbk): 978-1-4842-1986-7
DOI 10.1007/978-1-4842-1987-4

ISBN-13 (electronic): 978-1-4842-1987-4

Library of Congress Control Number: 2016951450

Copyright © 2016 by Inayaili de León

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director: Welmoed Spahr

Acquisitions Editor: Ben Renow-Clarke

Development Editor: Chris Nelson

Technical Reviewer: Massimo Nardone

Editorial Board: Steve Anglin, Pramila Balen, Laura Berendson, Aaron Black, Louise Corrigan,

Jonathan Gennick, Celestin Suresh John, Nikhil Karkal, Robert Hutchinson, James Markham,

Matthew Moodie, Natalie Pao, Ben Renow-Clarke, Gwenan Spearing

Coordinating Editor: Nancy Chen

Copy Editor: Tiffany Taylor

Compositor: SPi Global

Indexer: SPi Global

Cover Image: Designed by freepik.com

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springer.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales-eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

Printed on acid-free paper

To my parents, for making me love to read.

Contents at a Glance

About the Author	xiii
About the Technical Reviewer	xv
Acknowledgments	xvii
Preface	xix
Introduction	xxi
■ Chapter 1: The Planning Stage	1
■ Chapter 2: The Content Development Stage	23
■ Chapter 3: The Design Stage	47
■ Chapter 4: The Build Stage	93
Conclusion	137
Appendix: Resources	139
Index.....	145

Contents

About the Author	xiii
About the Technical Reviewer	xv
Acknowledgments	xvii
Preface	xix
Introduction	xxi
■ Chapter 1: The Planning Stage	1
The Team	2
Involve All the Right People	2
Define a Project Leader	3
Everyone Participates	4
Low-Fidelity Planning	5
Gaining Perspective	5
Digital Record	6
Finding Time in Your Schedule	9
Understand Your Calendar	9
Deprioritize Other Projects	10
Assigning Tasks	10
Use Everyone at Once	11
Do Not Use Everyone at Once	12
Account for Downtime	12

Keep a Tight Scope.....	13
Start with a Wish List.....	13
Organize	14
Prioritize	14
Break Down the Tasks	14
The One-Hour Test.....	15
Define Stages	16
Determine What Is Out of Scope.....	16
Do Not Forget Testing	17
Set Deadlines	17
Phase 1	18
Phase 2, Phase 3 . . .	18
Rollout Strategies.....	18
Managing Site Updates	20
Keep a Record as You Work Through the Project	20
Summary.....	21
■ Chapter 2: The Content Development Stage	23
Designing with Real Content	23
Focus on Accessibility	24
Good Accessibility Helps Everyone.....	25
What You Can Do	26
Accessibility and Usability.....	27
Reviewing Your Existing Content.....	28
Content Inventories and Content Audits	28
Doing a Content Inventory	28
Doing a Content Audit.....	30
Quick Content Inventories and Content Audits	31

Ongoing Maintenance	32
Avoid Duplicated Work.....	32
Phasing Content Updates	32
Updating and Simplifying Existing Content	33
Quick Wins.....	33
Shorter Is Not Better.....	34
Less Content May Be Better Content.....	35
Reduce Cognitive Load Without Restricting Access to Content	37
Have a Plan	40
Defining a Style	41
Improving Your Content Management System	44
Remove Inline Styles	44
Remove Styling Options.....	45
Add Structure to Your Content	45
Another Option: Doing Nothing	45
Summary	46
■ Chapter 3: The Design Stage	47
Evolution, Not Revolution	47
Focus on Reusability	49
Focus on Accessibility	52
Performance First.....	54
No More Flats	54
Setting the Rules.....	55
Write It	55
Build It	58

- Determining Breakpoints 59**
 - Decide in the Browser 59
 - Tweakpoints and Breakpoints 59
 - What to Keep in Mind 60
 - When to Use Analytics 60
- Defining a Style Guide 60**
 - Screenshots, Screenshots, and More Screenshots 62
 - Responsively Rationalize 64
 - Build Your Style Guide..... 65
 - Clean Up Your Style Sheets..... 66
 - When You Already Have a Style Guide 66
 - Style Guide or Pattern Library?..... 66
- Standardize across Sites..... 67**
- Quick-and-Dirty UX..... 68**
 - Sticky-Note-Sized Wireframes 68
 - Super-Speedy Prototyping..... 68
 - Express Testing..... 70
 - Key Things to Test..... 71
- Grids and Type 72**
 - Convert Your Grid to Percentages 72
 - Explore Responsive Grids 73
 - Reorder Content..... 76
 - Adjust a Strict Typographic Scale 78
- Handling Your Images..... 79**
 - Make an Image Inventory 79
 - Mind Those Bytes 81
 - Consider SVG 81

Some Useful Responsive Web Design Patterns.....	82
Navigation Patterns	82
Tables	87
Getting the Most Out of Feedback and Reviews	91
Summary	91
■ Chapter 4: The Build Stage	93
Experiment on Smaller Projects	94
Focus on Accessibility	94
Focus on Performance	96
Work with a Performance Budget.....	96
Understand Perceived Performance.....	98
Large Screens Like Lean Sites, Too	99
Trim Down Your Web Fonts	100
Some Handy Tools	100
Fluid Grids and Type	103
Fast Track to a Fluid Grid	104
Improve your Markup	104
Remove Inline Styles	106
Get Inspired	107
Abandon Absolute Units.....	109
Scalable Type.....	110
Media Queries	110
Mobile First, and Enhance Progressively.....	111
Browser Support.....	114

Responsively Retrofitted Images.....	114
SVG Images	114
Picture and srcset.....	115
Image Caching.....	117
Compress Those Bitmaps	118
Optional Images.....	118
Testing.....	118
What to Test On and What to Fix.....	119
Build Your Device Lab	121
Emulators and Other Tools.....	122
Improving Your Process.....	125
Writing Code and Markup.....	126
Publishing Your Site.....	132
Measuring Success After Release.....	134
Summary.....	136
Conclusion.....	137
Appendix: Resources.....	139
Index.....	145

About the Author

Inayaili de León is Lead Web Designer at Canonical, the company behind Ubuntu, where she focuses on establishing and evangelizing the brand's visual direction online. She loves and lives the Web and her job and that she can learn something new every day. Inayaili has a degree in Communications Design and has been working as a web designer since 2003. Working on content-heavy web sites is a challenge and a pleasure that she happily takes on, transforming what could easily look like a mess into user-friendly designs. Inayaili is an author and a speaker, writing on her own blog, Web Designer Notebook (<http://webdesignernotebook.com>), as well as for popular online publications such as A List Apart, 24 Ways, Smashing Magazine, and .net Magazine. She is also a member of .net Magazine's and Smashing Magazine's Expert Panel. Inayaili is Panamanian-Portuguese, was born in the USSR, and has lived in London since 2008—her favorite city in the world. She loves cats and naps. Her portfolio, speaking schedule, and much more can be seen at <http://yaili.com/>.

She is the co-author of *Pro CSS for High-Traffic Websites* (Apress, 2011).

About the Technical Reviewer



Massimo Nardone has more than 22 years of experiences in security, web/mobile development, cloud and IT architecture. His true IT passions are security and Android. He has been programming and teaching how to program with Android, Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than 20 years.

He holds a Masters of Science degree in Computing Science from the University of Salerno, Italy.

He has worked as a project manager, software engineer, research engineer, chief security architect, information security manager, PCI/SCADA auditor, and senior lead IT security/cloud/SCADA architect for many years. His technical skills include security, Android, cloud, Java, MySQL, Drupal, Cobol, Perl, web and mobile development, MongoDB, D3, Joomla, Couchbase, C/C++, WebGL, Python, Pro Rails, Django CMS, Jekyll, Scratch, and so on.

He currently works as Chief Information Security Office (CISO) for Cargotec Oyj.

Massimo worked as a visiting lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University). He holds four international patents (PKI, SIP, SAML, and Proxy areas).

Massimo has reviewed more than 40 IT books for different publishing companies and is the coauthor of *Pro Android Games* (Apress, 2015).

Acknowledgments

Like any other significant undertaking, writing this book would have been nearly impossible without the help of a good deal of people.

First and foremost, thank you to Nicklas, who read all the chapters more than once and gave me immensely helpful and insightful feedback. He did all of this often after a full day's work, and having made dinner and put our son to bed. This book would not exist if it were not for him (or it would have taken an inordinate amount of time and made a lot less sense).

Thank you also to my friends and colleagues at Canonical, whose infinite experience and expertise they are kind enough to share with me every day. A special thanks to Alejandra Obregon, for suggesting that I write on the company's design blog about the process we went through when making our site responsive, and for crafting the structure and topics of the series of articles that followed. And thanks to Carla Berkers, for her excellent tips on how to best conduct user research.

Many, many thanks to the team at Apress, who have so professionally edited, revised, managed, and produced this book. Namely, thanks to Ben Renow-Clarke for asking me to write the book in the first place, to Nancy Chen for being so kind and polite even when I was delayed for more than a few weeks and definitely needed a nudge, and to Massimo Nardone and Chris Nelson for so patiently revising my drafts and contributing their expert insight to every chapter.

Last, but certainly not least, thank you to Rafael, my son. Luckily, I do not think he will remember me sitting in front of my computer during bath and bedtime for so many nights. Now we can go enjoy our weekends again! :)

Preface

Five years ago, when I finished writing my first book, I swore I would never do it again.

As fellow authors—and anyone who has ever had to write an essay, a dissertation, a thesis, a report—know, realizing that you have to write something that has to make some kind of sense can take over your life in a way you do not foresee when you first sign up to do it.

But just as you swear you will not go through the pain of labor again in the moments after you have a child, only to have a second and third, here I am again.

I do love writing, and I always encourage people to share their own experiences, the problems they are facing, and the solutions they came up with. As banal as they might sound to themselves, many times people do not want to read about how other teams are doing something wacky, different, or great. They want to hear about how the simple things they are doing are also being done by others, as a way to confirm that their ideas are not crazy and that their processes do make sense.

Our projects, teams, and processes are messy. They are not linear and neat. We get interrupted, we have to stop and start things later, we have to work with legacy code and designs, and we have to handle copy that might be too long or too short. But as web professionals, we know what kinds of things we would like to improve in our web sites, to make them better for users and for our businesses. This book is about carving space within your imperfect schedule, to make your site better by making it responsive, bit by bit, step by step—tackling one thing at a time until you get to something you can proudly put out there for others to see and use.

In this book I am offering you my experience. It will certainly be different than yours. Some of the things I recommend that you do might be commonplace for you. Others may be new. Try them—they just might work out.

Introduction

I think it is safe to say that most people who design and build web sites are by now aware of responsive web design. We know we should be building sites and applications that adapt to any screen size the user chooses to view them on, as opposed to building fixed-width sites as we did before, which makes them hard to use on devices smaller than a laptop.

Previously, we could assume that the flat mockups we designed were going to look fairly similar in size, layout, and positioning of elements across browsers and screen sizes; but with the emergence of responsive web design, we no longer have that fine-grained control. Our carefully crafted designs are now free to run wild across screens.

When the concept of responsive web design appeared, we took a huge step toward the maturity of our craft. But at the same time, we were left with millions of fixed-width sites that desperately needed a redesign.

This introduction takes a brief look—a reminder of sorts—at the evolution of web design, from its inception in the early 1990s, through the convoluted use of HTML and CSS in the late 90s and 2000s, to the revelation that was responsive web design at the beginning of this decade. Let's step back a bit.

The Web Used to Be Responsive

Back in the beginning of the Internet, every site was responsive. Sites were made up of documents with little more than text that ran across the full width of the viewport, and links that connected those documents together (see Figure I-1).

World Wide Web

The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#) , [Policy](#) , November's [W3 news](#) , [Frequently Asked Questions](#) .

[What's out there?](#)

Pointers to the world's online information, [subjects](#) , [W3 servers](#) , etc.

[Help](#)

on the browser you are using

[Software Products](#)

A list of W3 project components and their current state. (e.g. [Line Mode](#) , [X11 Viola](#) , [NeXTStep](#) , [Servers](#) , [Tools](#) , [Mail robot](#) , [Library](#))

[Technical](#)

Details of protocols, formats, program internals etc

[Bibliography](#)

Paper documentation on W3 and references.

[People](#)

A list of some people involved in the project.

[History](#)

A summary of the history of the project.

[How can I help ?](#)

If you would like to support the web..

[Getting code](#)

Getting the code by [anonymous FTP](#) , etc.

Figure I-1. *The 1991 web site for the European Organization for Nuclear Research (CERN)*

Those were simple times. But web site designers wanted more. We wanted to be able to truly design our web sites. We wanted them to be more than a vessel for words and hyperlinks, and our layouts and designs became more engaging but also more complicated.

The design tools we had in the beginning were primitive. At first, we had inline styles that lived in our markup. For every bit of text that we wanted to style, we needed to add that style directly to our HTML. It was cumbersome work.

CSS, when it was released, was basic. It did not easily re-create for print the types of layouts we were used to designing, so we started using tables to position elements on the screen. Tables were originally developed to display tabular data, but we used them to re-create the web layouts we were designing with tools and practices made for print design.

We wanted to replicate in this new medium the control we had when designing for a medium we knew well. We wanted something that resembled a page, with margins and edges—something familiar. And although we gained that control somewhat with the use of tables for layout (see [Figure I-2](#)), we lost the fluidity and adaptability of the early web sites.



Figure I-2. eBay's November 2000 web site, using tables (highlighted in the screenshot) for layout

We did eventually let go of using tables for layout—the result of the substantial effort of a few voices in the web community who called and campaigned for a focus on web standards and for separation of concerns (see Figure I-3). But despite the tremendous achievement that was the move from using presentational tables to following standards-based HTML and CSS, it still took us another few years to go back to that initial, flexible Web that was accessible on any browser.

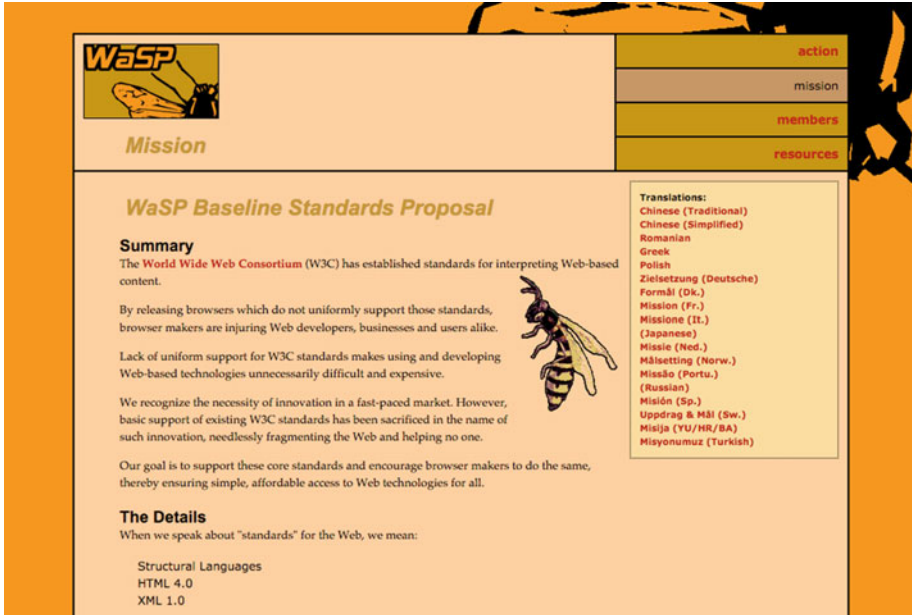


Figure I-3. *The Web Standards Project's mission*

Enter Responsive Web Design

It was 2010 when Ethan Marcotte published his “Responsive Web Design” article on A List Apart: <http://alistapart.com/article/responsive-web-design>, 23 years after the release of HTML, 19 years after the release of CSS, and amid an ever-increasing explosion of mobile devices and mobile web users. In his article, Ethan explained that responsive web design relies on the combination of three technical factors: fluid grids, flexible images, and media queries. But this new approach also assumes a new way of thinking about web sites: instead of creating compartmentalized versions of a site, we should be thinking about creating different aspects of the same unified design, which adapts to the context on which someone chooses to use it.

Responsive web design allows the elements within a site to reflow, move around, change size, and even change their appearance, to best fit the device that is being used. In a responsive web site, at certain sizes (breakpoints), either the entire design or certain elements adapt to their surroundings—the browser and the device. Ethan’s call was to design web sites that could adapt to any device, with any screen size and any combinations of capabilities.

We remembered that we had been there before, and now it was easier to express the place we needed to get to—we had a name for it. When reading Ethan’s article, every web designer was at the same time both nodding and terrified of the work that lay ahead. Just when we thought we had it all figured out, we were thrown a curveball, and we had to learn to adapt. Again. We thought about the time it would take us to convert existing fixed-width sites into fluid, responsive ones, and the task seemed insurmountable—nearly impossible.

We have come a long way since 2010. Some of the techniques that Ethan described in his original article on the use of media queries and responsive images are now more widely supported by browsers and are more widely understood and appreciated by designers and developers. HTML and CSS standards have been, or are in the process of being, updated to reflect the needs of responsive web sites. As web professionals, we have tried and improved new processes of designing and building sites that, just like the sites themselves, are more flexible and adaptable.

Key responsive redesigns of major web sites have also been released in the past five years (see Figure I-4), and we have all followed along. Businesses and clients now understand better than before the advantage of creating sites that people can see from any device they choose to use. And we can all grasp how much more effective it is to maintain one web site instead of several different ones.



Figure I-4. *The Boston Globe was one of the first major responsive redesigns, launched in January 2012*

We have learned many lessons and overcome many obstacles in the process, but there is still a lot of work to be done, and there are still a lot of sites to be updated—and one of those web sites just might be yours.

Time Is a Luxury

As web design professionals, I think by now we all agree that every project we start from scratch should be a responsive project. But not all projects start from a blank slate, and many teams are still managing and working on fixed-width sites and apps that they cannot simply get rid of to begin a fresh, new responsive project.

When my team and I started to plan the responsive retrofit of our fixed-width site, `ubuntu.com`, it became clear to us that most resources out there that focused on how to make a web site responsive assumed that you were beginning from scratch. This was certainly not our case, and it is not the case of many other teams. We had an existing site with hundreds of pages and millions of users, and not a lot of time to embark on what seemed like a massive effort. I lie—we had no time at all.

Our team is constantly and consistently receiving briefs to work on projects that support other teams across the company, and our backlog is immense. Finding time to work on something self-initiated, however small, is difficult, let alone working on what seemed like a huge project, with a fuzzy scope, that would take all of our resources away from other important projects. We were not going to be able to sit down for a few weeks or months and focus on this one project.

My team leader suggested that I start documenting the process we were following: the steps we were taking on this messy, wonderful journey of converting an existing site with lots of existing content and existing code into a responsive site, when we did not have all the time in the world.

If I had to guess, I would say that most designers and most teams do not have the luxury of starting a responsive design from scratch. Starting with a clean slate is an ideal-world scenario, and we would not be able to call ourselves problem-solvers if we were working in ideal scenarios, right?

Taming the Project Beast

It does not matter how much you want to convert your site, or constellation of sites, to being responsive. That desire will not turn itself into your team's and your company's priority. The most important thing your team will have to do will still be to keep your site updated with new content, new campaigns, and everything else required to keep a living, breathing site alive. You cannot simply alienate the business and ignore the other projects you need to work on. So it is important to remember that it does not matter how badly you want to make the transition—you must keep your priorities in check throughout the process. And above everything else, you and your team will need to approach the possibly lengthy project with generous amounts of patience, persistence, and enthusiasm.

This is why I consider Chapter 1 in this book so important. In it, I guide you through how to start thinking about and planning your responsive retrofit. I give you easy-to-replicate exercises that you can do with your team to move all the ideas from your heads to a tangible plan and convert them into a workable, realistic timeline. Chapters 2–4 are more focused on getting the project done, from content through design to building and releasing.

Chapter 2 looks at the most important part of your site: the content. I give you some tips on how to begin analyzing your existing content and what to do with those findings. You also see how to manage content updates and maintain an evergreen site, while at the same time you and your team are working on a new codebase and potentially a new content management system.

Chapter 3 looks at some of the most common design problems of responsive retrofit projects and at strategies for how to solve them when you are pressed for time. You explore the importance of style guides, reusability, and letting go of ideal user experience processes when time and budget do not allow for it.

Finally, Chapter 4 looks at time-saving, fear-squashing strategies for building and releasing your brand-new, responsive web site. I do not delve into coding, because that could easily be (and is) the topic of a separate book; but I cover this step of the process so that, as with the rest of the project, you can manage your resources and your time well while building a web site you can be happy with at the end.

Some Housekeeping

You will notice that I do not mention specific project-management methodologies in this book. Hopefully my advice, especially in Chapter 2, is agnostic enough that it can be applied to whichever methodology you and your team follow, be it agile, waterfall, or some kind of hybrid (which seems to me to be not at all uncommon).

Throughout this book, I refer to the *site* or *web site* interchangeably when I am talking about the project you want to responsively retrofit, but it might be a site, an application, or what have you.

Summary

In this chapter, you have travelled back in time to recall the inception of the web and web design. You have seen how, although web sites began by being simple and flexible, the need for differentiation and design control quickly made them into stiff, convoluted things. You have also learned about how Ethan Marcotte’s “Responsive Web Design” article influenced designers and developers to go back to creating sites that were flexible and adaptable to any device.

However, some sites have not yet been able to transition from fixed-width to responsive. Overstretched teams with never-ending backlogs are common, and yours may be one of them.

With the help of this book, even if your team is super busy, you will be able to achieve the goal of making your existing site responsive. The first thing you need to do is plan your responsive retrofit project, as I discuss next.

CHAPTER 1



The Planning Stage

There is so much to do, so much you and your team want to fix and improve, that you probably want to dive straight into the responsive retrofit project. But it pays to spend a bit of time planning how your scarce time and resources will be used throughout the duration of the project and understanding that the transition will not happen overnight.

By the end of the planning stage, you should feel as though you have a better idea of what converting your site to responsive will entail, how long each task you want to accomplish may take, and when you plan to release the first iteration of your responsive site.

In this chapter, you learn:

- How to make sure you involve all the key people from the start of the project
- How to start planning your project using low-fidelity, low-cost tools
- How to find time in your schedule to work on your responsive retrofitting project
- How to maximize each person's time throughout the project
- How to define the scope of your project
- How to define milestones and deadlines so your team has concrete dates to work toward
- Which roll-out strategies are the most typical for responsive redesign projects
- How to factor parallel site updates into your plan
- The importance of keeping a record of decisions for the duration of the project

The Team

You need people to work on any project. Whether that means you will be part of a team of only two or three people, or a team of dozens, you should know who will be working on the project during its different stages.

Before you delve into any kind of planning or task assignment, it is a good idea to spend some time finding out which people will and should be involved in the project. This usually is not something that is handed to you on a plate. You need to make sure you have enough conversations and interviews to assess how the project might affect different people in different departments. Only after you conduct this fact-finding exercise will you have enough knowledge about the structure of the team and organization to know who will be part of the project at which stages.

Involve All the Right People

Based on your initial research, write down the names of all the people who you think will be involved in the project, from beginning to end. You may have to augment this list as you go through the planning stage, but try to be as thorough as possible now. Everyone you have listed should know that they are part of the list—part of the team.

As with any other project, you should be clear about what each person's role will be in the project, even if the description of their role will change throughout the process. So, for each person, write a brief description of how they will contribute to the project. Will they be mainly creating the front-end code? Will they be writing content? Are they just part of the sign-off process? Do they need to be informed about the current status of the project, but they will not actively work on or influence it?

I also find it useful to list in the same table what each person's skills are, what they are good at and have experience at, and also what their responsibilities are—what they will have the final say in. A user experience designer may have, for instance, the final word on the information architecture of the site or the navigation design. A front-end developer may be responsible for the coding standards, browser support, and testing.

You can start your table with the following column headings, listing one person per row:

- Name
- Title, Team
- Role in this project
- Role description
- Participation level
- Core skills

Common roles that are part of a responsive retrofit project are the same as for any other web project. Depending on the size of the project, your team may include the following:

- Researcher
- Copywriter

- Content strategist
- User experience designer
- Interaction designer
- Visual designer
- User interface designer
- Art director
- Front-end developer
- Back-end developer
- Project manager

You also want to make sure other people outside of the usual team are included in the list if they will have any participation or say in the outcome of the project:

- CEO
- Brand manager
- Quality assurance engineer
- Product manager
- SEO analyst
- Communication and sales team

Remember that some people who might participate in the project do not have to participate in any of the planning meetings. These include the following:

- Testers, whether they are external users who are formally recruited, or internal people with whom you may do some guerilla-type usability testing
- Consultants or anyone who may provide their experience and expertise on a given subject, but who will not effectively work on or have any say in the project

Now that you have a good overview of everyone who will be involved in the responsive project, you need to decide who will hold the project's key role.

Define a Project Leader

The next thing you need to do is make sure your project has a leader: someone who will own the project and be in charge, reminding everyone to keep on track. If you do not define a project leader, it becomes too easy to make up excuses as to why you have to delay the responsive project and prioritize other projects. Something to keep in mind is that the project leader does not have to be the team's project manager. Just as in many other design projects, the person who holds the lead role tends to be a senior designer or developer, quite often a user experience designer.

If I had to guess, I would say it is likely that, if you are reading this book and following the exercises, you may be the project leader. However, you may simply be facilitating the planning of the project, but need to delegate the responsibility of leadership to someone else.

The project leader is the person who will do the following:

- Make sure the project is completed
- Make sure the project is delivered on time and on budget
- Keep cheering the team on, even when it does not feel like you are making progress
- Tie the rest of the team together, because people may not be working all together at the same time

The project leader should be someone who will work on the project, not someone who is external to the team or does not participate in the day-to-day work. But just like the other members of the team, the project leader does not necessarily have to be working on the project at all times.

When I led the responsive retrofitting of `ubuntu.com`, there were several periods of time when I had to work on other projects, but these had been planned and scheduled (we look into these pauses later in this section). Nonetheless, the responsive project was always part of my responsibilities.

Even when the project leader is not doing deliverable-generating work on the responsive project, it is likely that other people are. So, the project leader still needs to manage and be on top of the team's work and what is being produced.

Everyone Participates

Everyone will have a different idea about exactly what type of effort a responsive retrofitting project involves and what it should achieve. In the planning stage, you want to make sure this subjectivity is eliminated and everyone is on the same page.

When everyone who will work on the responsive project participates in at least some of the planning meetings and workshops, they will be more invested in the project, and they will better understand why certain decisions were made. It is harder to challenge a decision in which you had a say than a decision you had nothing to do with or the context of which you do not understand.

By the end of this stage, it is likely that more than a few compromises will have been made and that not everyone's ideas and desires will have gone through. But you will have had the discussions necessary so that people can understand why certain decisions were made.

With limited time and resources, you have to make some tough decisions and compromises, so you do not want to run the risk of someone who joins the project at a later stage challenging every single decision that was made in the beginning—and probably made for good reason.

Keep in mind, though, that this does not mean your plan will not change at all. You may change your mind or discover that a solution you had not thought of is better than what you planned before. You should always allow for some flexibility when you find out your first idea is not the best for the project.

Another factor worth remembering is that when more senior executives are involved in the initial planning meetings and can listen to how the team is exploring the project—assuming your team is made up of smart, dependable professionals who create good work, of course—they tend to step away from the nitty-gritty, daily grind of the project and let everyone else do their jobs. Again, it is important that everyone, even C-level executives, invited and encouraged to participate in the scoping and scheduling of the project if they are to (or want to) have any type of involvement in the project at later stages.

I think I know what may be going through your mind right now: “I will not realistically be able to get all of these people together in a room for hours on end, to plan this project—didn’t you promise that this book offers a solution for teams with no time? What gives?”

I understand and know from my own experience how difficult it is to get lots of busy people to attend meetings that can span several hours. But I can promise you that making the effort in the beginning of the project will pay off tremendously throughout the development of your responsive site, because you will have time to worry about solving tricky design and development problems, rather than having to deal with politics and conflict inside and outside of your team.

Taking the time to ensure that all the right people are in the room when you kick-start your responsive retrofit project is a task in itself, one that may require a high degree of patience and waiting for the right moment when everyone is available. But if you want to get things done right from the get-go, that is what it takes.

Low-Fidelity Planning

I find that the simplest and easiest way to plan a large, complex project is to start with lots and lots of sticky notes. Working with sticky notes is a type of low-fidelity planning that allows you to move things around as many times as needed as your requirements and schedule change, and as you learn more about the project.

You can and should move the sticky notes as many times as you think you need to, until you feel you have arrived at a plan that you are confident will work for your team and for the needs of your site. It is easier and cheaper to move bits of paper around than to change your design and code later.

Gaining Perspective

When you are planning a project that has the potential to be extremely large and complex, involving several people throughout a long period of time, there will be a lot of small moving parts, small tasks and ideas, that you have to group, prioritize, and schedule. By having all the building blocks of your project easily visible in front of you, you will have a better perspective of the project’s dimensions and how much you may have to cull for each phase you want to plan.

This is a little like watching a crime show on TV. The detectives always spread all of their findings and evidence on a big wall, and they stare at the wall for hours on end, trying to find missed connections or clues. Being able to see the entire case in front of you makes it easier to find these missing links and to think of creative, out-of-the-box solutions.

So just like those detectives, you need lots of free wall space, because sticky notes (your main tool at this stage) are known to multiply at an exponential rate. Ideally, you can find some wall space where you can keep your sticky notes and other things up for the entire duration of the responsive project. If that is not possible, try to find ample wall space at least for the planning sessions, and make sure your findings are well documented at the end of each meeting.

You do not want to run out of sticky notes, so stock up. Do not be fooled by cheap brands, though. I recommend that you buy the real deal: 3M's Post-it® Notes. You want to ensure that your sticky notes—that is, your ideas and your plan—do not go flying off at the faintest breath. You want them to stick! Also get as many colors as possible, because you may want to differentiate things like types of tasks, assignees, project stages, iterations, and even different projects.

Digital Record

It is likely that you will eventually have to move all the planning into a digital format. But at least in the initial stages, when ideas are being generated and you are trying to fit a complex project in with many other projects, being able to change your plan without cost and effort is important.

Depending on how your team usually works, you may want to move at least key pieces of the plan onto a digital format—which can be more easily shared with everyone—as soon as possible. This is especially true for distributed teams, where not every team member works in the same location at all times. This digital record of your plan, tasks, and deadlines can be more or less informal, depending on what works for you.

There are several tools you can use to plan your responsive project, but some have a proven track record and are favored by many design teams. Most of these tools offer a basic set of features and a small number of user accounts for free, but you have to pay (usually monthly, per user) to take advantage of all the features and provide access to several users.

Basecamp (see Figure 1-1), currently in its third iteration, among other things, allows you to see a calendar view of all or some of your projects at once, with all their milestones and deadlines. You can also follow a project or a person's progress on a given project, or across all projects, which can be handy. Another useful feature added in Basecamp 3 is the ability to create automatic check-ins with the team, where you can ask anything you want on a recurring basis.

OPEN SOURCE TOOLS

The tool you choose to use to plan your project will greatly depend on how your team works, the functionality you need, and how much you can and are willing to pay. There are several open source project-management tools out there in addition to the ones I have outlined. If you want more control over your tool, you may want to investigate these further.¹

¹Robin Muilwijk, “Top 11 Project Management Tools for 2016,” March 28, 2016.
<https://opensource.com/business/16/3/top-project-management-tools-2016>.

The screenshot shows the Basecamp 3 website. At the top left is the Basecamp 3 logo. At the top right are links for Features, Pricing, Using Basecamp with Clients, About us, Support, Sign up, and Log in. A banner at the top center says "Version 3 is all new for 2016!". The main content area has a central illustration of a person with a speech bubble containing various project-related icons and questions: "DID WE GET THAT DONE YET?", "WAIT, WHO MADE THAT DECISION?", "WHEN IS THIS DUE?", "WHERE DO I PUT THAT?", "NO ONE TOLD ME THAT!", "WHO SENT THIS TO THE CLIENT?", "THIS IS EXHAUSTING! WHERE ARE THE DONUTS?". Below the illustration is the headline "Working with other people? Struggling to keep everyone on the same page?" and the sub-headline "It's time to try the Basecamp way." Two columns of text describe the benefits of Basecamp. On the right, a sign-up form is visible with a "Sign up now for free below" button, a testimonial "Just last week, 10,772 companies got started with Basecamp 3!", a "Sign up using Google" button, and a form for "Your Name" (Julie Appleseed), "Your Email" (julia@widgetco.com), and "Company/Organization" (Widget, Co.), with a "Sign up" button at the bottom.

Figure 1-1. Basecamp 3

Another tool that has been around for a while is Asana (see Figure 1-2). This task-management software gives you at-a-glance project status views, much like Basecamp, and also calendar and file views. With Asana, you can create guest accounts with limited visibility of the content. It integrates with popular services such as Campaign Monitor, Dropbox, Evernote, Google Drive, Harvest, HipChat, JIRA, Slack, and WordPress.

The screenshot shows the Asana website. At the top left is the Asana logo. At the top right are buttons for "Get Started for FREE" and "Log In". The main content area shows a project view for "Mission to the Moon" with a task "Design flag to place on moon" and a description "Flag size should be 4ft x 6ft and three colors." Below the task is a file attachment "unicorn.pdf". A list of comments follows: "Skylar J. 3 days ago Does anyone have ideas on what should be on the flag? Ryan O. maybe you have suggestions?", "Ryan O. 2 days ago Maybe a catstronaut or a unicorn?", and "Ryan O. attached 2 days ago unicorn.pdf". On the right, a blue sidebar contains the text "Track projects from start to finish" and "Responsibilities and next steps are clear, so you can shoot for the moon—and get there." with a "Learn More" button.

Figure 1-2. Asana

A popular tool to take quick notes and collect inspiration and research findings is Evernote. Evernote lets you digitize physical notes, too, and create notebooks for different projects. You can also annotate documents like PDFs and see previous versions of your notes. One of its key selling points is its comprehensive search, which searches everything in your notes, including inside images and attachments. It also provides offline access, which can be useful if you want to read through your notes anywhere, at any time.

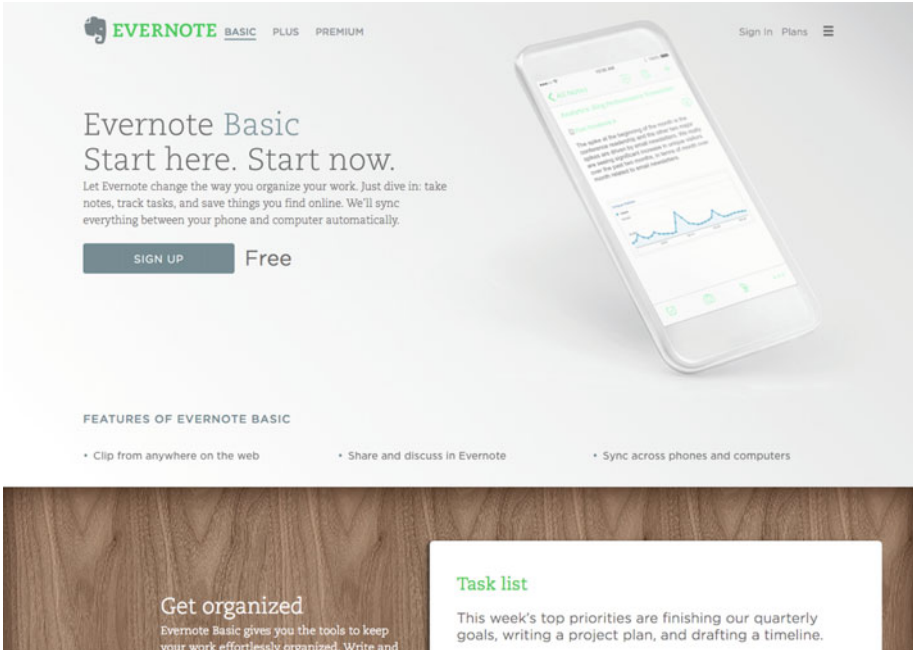


Figure 1-3. Evernote's free Basic version

A more recent addition to the project- and task-management tool pool is Trello (see Figure 1-4). Trello works with a model of cards in columns. Its super-fast and user-friendly interface has made it into an almost-instant success among design teams. You can create checklists within the cards and embed documents and comments. Trello supports Markdown syntax, and it integrates with services like Dropbox, Evernote, GitHub, Google Drive and Google Hangouts, MailChimp, Salesforce, Slack, and Twitter.



Simple on the surface, with more under the hood.

A Trello board is a list of lists, filled with cards, used by you and your team. It's a lot more than that, though. Trello has everything you need to organize projects of any size.

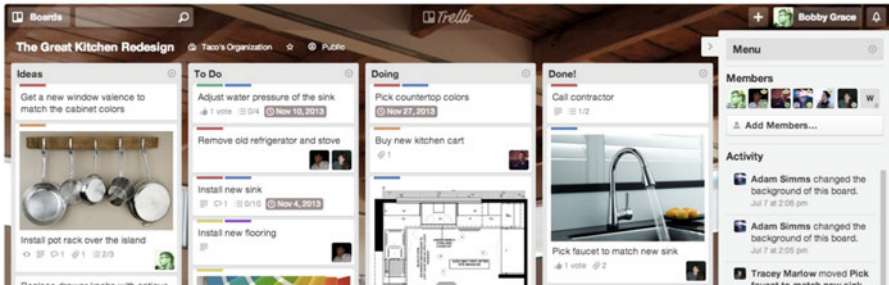


Figure 1-4. Trello

Finding Time in Your Schedule

If you are reading this book, you probably work on a team that is incredibly busy, managing (or juggling) several projects at the same time, trying to plan and fit in different briefs from clients or other parts of the company. You are also probably in charge of updating and maintaining an existing site, and you will keep having to do so even while you are working on a responsive retrofit of that very same site.

I find that the hardest part of responsively retrofitting a large site tends to be finding the time in your schedule to do so, rather than solving the several technical and design problems that will soon come your way. But there are some processes and tools you can put to good use that will help you go beyond that familiar feeling of being faced with a project that you want to work on but that seems to not fit in your schedule.

Understand Your Calendar

Before you start adding chunks of time and dates in your calendar to work on the responsive retrofit, you need to know what your other commitments are. Using sticky notes, create a grid of the next few months, with each month or week running across the top of the wall as column headings. Then add to the corresponding column all the deadlines when you must deliver other projects. You also need to add to the schedule the time you will be working on these projects, not just the deadlines (see Table 1-1).

Table 1-1. *A Grid View of Your Calendar*

	Jan 4–8	Jan 11–15	Jan 18–22	...
Name 1			Project B: release V1	
Name 2		Project A: content finished		
...				

To make this grid a bit more complicated, but also more useful, I normally list the names of the key team members or the key disciplines (content, design, development, and so on) down the left side, as row headers. This gives you a more granular view of the calendar and also highlights more free time within your team, because you have a person view, and not everyone works on every project at the same time.

When you are finished with this exercise, you will have a better idea of whether there are any free blocks of time when you can fit in the responsive project, or whether you will have to do some deprioritization of other projects so that you can work on making your site responsive.

Deprioritize Other Projects

Look at the calendar you came up with in the previous section, and identify any projects whose deadline is not set in stone or is still vague; that has an undefined or loose brief; or to which a client has not yet committed by paying or signing a contract. Also identify other self-initiated projects that you deem less important than converting your site to being responsive (which should be most of them).

At this stage you must be ruthless, or you will never be able to fit in a responsive retrofit. If you do not find it tough to deprioritize some projects, you are not being tough enough with your schedule and your priorities.

Once you have arrived at an updated prioritization list of projects and an adjusted timeline, speak to the people involved in the projects that have fallen down the priority list to explain that your team will be able to reconsider your timeline once the briefs become a little less unclear, or once other matters that make the projects unclear are resolved. Explain that you are trying to plan a responsive redesign project, which will take priority over any loosely defined brief—and make sure to explain the benefits that working on and delivering a responsive site will bring to the entire company.

I agree that all this sounds much easier said than done: now is the time to polish those diplomatic skills. On the flip side, by the end of this step, you should have more free blocks of time in your calendar, during which you can work on the responsive project.

Now you are ready to begin filling in those spaces with the work that needs to be done.

Assigning Tasks

The time you spend defining your project and your schedule will have been in vain if you are not able to then match each task to the right person, or group of people, at the right time. You want to make sure that each time one of your team members works on the project, their time and skills are being applied effectively.

For this to work, you need to develop a schedule that factors in not only each person's main skills, expertise, and availability, but also the number of people who are necessary for a task to be completed. Some tasks require more people than others in order to be done well.

When assigning tasks to the different people on your team, consider that there are three types of tasks:

- Tasks that are done by a group of people
- Tasks that can be done by just one or two people
- Tasks that can be done during downtime, when there are gaps in your schedule

Use Everyone at Once

Certain tasks become easier and are done much more quickly when you can all sit together and discuss the problem at hand. By and large, when you need to generate several ideas, having a brainstorming session with a handful of people will give you better results, more quickly, than sitting on your own and trying to come up with ideas over days or weeks. These idea-generation sessions do not have to be lengthy to be useful.

Let's say you want to think of a more flexible solution for designing the footer of your site, which may include several links to key sections of the site, social media profile links, terms and conditions and other legal links, and so on. In this situation, you can try to find a two-hour block during which a combination of three to five designers and developers sketch their own suggestions and everyone discusses the pros and cons of each proposed solution. By the end of this session, you will have two or three ideas that can be then prototyped, tested, and iterated on until you have crafted a responsive footer that will work for your site.

Here are some other tasks where it is useful and more productive to have three or more people working on at a time:

- Device and browser testing
- Generating ideas for a responsive navigation
- Generating solutions for a responsive grid
- Discussing solutions for dealing with responsive images
- Going through all the design patterns of your site and discussing which should be deleted or merged with other patterns
- Having a session where you name your site's design patterns
- Sketching a new information architecture for your site
- Other tasks where generating ideas is the key objective

Do Not Use Everyone at Once

When you defined the team to work on the responsive project, that did not mean everyone would be working on the project at all times. It is unlikely that there will be many moments when the entire team is sitting together working on the responsive project, and that is fine and expected. Several repetitive, time-consuming tasks can be done by just one person, while everyone else can be working on other tasks or on other projects.

Anything to do with making inventories of content, pages, images, or design patterns can be done by one person. Many development tasks, especially those related to cleaning up and refactoring code, can be done by a couple of developers. Further, much of the research into tools and solutions, and into how other people solved similar problems, can be initiated by one person, to be discussed with the wider team later.

Here are some more tasks that can be accomplished by one or two people at a time:

- Cataloging IA
- Inventorying content
- Inventorying design patterns
- Inventorying image usage
- Cleaning up markup
- Cleaning up and refactoring stylesheets
- Building small prototypes—for instance, to test a specific design pattern
- Proofreading copy
- Editing images
- Writing or editing copy for a new style guide, or updating an existing one

Account for Downtime

There will be times when no one on the team will be working on the project. That does not mean you can slack on making sure the project is running according to plan, though. As long as these periods of pause have been accounted for, and as long as everyone knows when they will happen, that should not be a problem.

During these pauses, the role of the project leader is important, because they must ensure that when the team returns to work on the responsive retrofit, the momentum and excitement of working on the project are replenished. It can be dispiriting to not see progress on the work you are doing and to not be able to check things off or have the end in sight as you do them. The project leader has to make sure the project is at the back of each team member's mind as something they will accomplish in its own time, and as something that is moving forward in the right direction.

Now that you have carved some time in your schedule to work on the responsive project, and you have a good overview of the time each person on your team has available during the next few weeks or months, it is time to delve into defining the scope of the project.

Keep a Tight Scope

You do not have the luxury of several large blocks of time to work on this project, so you need to focus on what you can achieve in the limited time you do have. If you follow these steps and complete the exercises I suggest, you and your team will have a good understanding of the expected outcome of your project, what you need to do to reach that outcome, and, later, how the project will fit alongside your other projects.

Before you start these exercises, the responsive project is just an idea in everyone's mind. Each person will have been thinking about it differently. Some people may think it does not involve much effort, whereas others may be terrified of its dimension (I suspect most of us may be in the latter group). Once everyone's ideas are outside of their brains and visible to the team, it will be easier to start defining the scope and timeline of the project.

As you go through the steps in this section, be sure to take photos, write down, or record in some way all the ideas that pop up. You may want to refer to the initial stages later, and if you only capture the plan that you come up with at the end, that will not be possible.

Start with a Wish List

Once you have established the high-level goals of the project, I find that the simplest way to begin defining a more granular scope is for everyone to write on sticky notes whatever comes to mind that they would like to improve on the site, regardless of whether those things are directly related to making the site responsive. They should write one thing per note.

This is also a good way to begin breaking down a large project into small chunks. We all know that smaller tasks seem less daunting and more attainable than a single big task.

You can ask people to prepare for this exercise in advance, but leave some time for them to think about it and write down their ideas during your planning meeting. If you want to give some instructions about what you want everyone to focus on when writing their wish list, you can say that all tasks listed should be able to be completed in a certain number of days, or in a one- or two-week sprint, for instance. Once everyone has finished this task, have them stand up and put all the sticky notes on the wall, anywhere they want.

The next step is for everyone to help move the sticky notes around on the wall so that they are roughly gathered by topic. Do not worry if some notes are moved back and forth a few times between topics, by different people—just make sure after a certain time that the exercise is finished and everyone stops moving the notes. You will be surprised at how what initially looked like dozens or hundreds of disparate ideas can in fact have many common threads.

Organize

Now it is time to organize the tasks and topics in a more structured way. Some of the tasks you define will likely be related to the project only marginally, so you can begin by moving these aside or assigning them to a different project or a following iteration of the responsive project.

You can also divide tasks by degree of difficulty or completion time. Perhaps you want to start with the quick wins in one list: the changes and updates you can do to your site that are relatively easy or not time consuming but that would represent a step in the right direction (the responsive direction!). Move larger tasks, which may be complicated and lengthy, or may involve different teams, onto another list.

At this stage, you may want to clean up the wall and create new notes that encapsulate the main idea from each group of notes; otherwise, you will have to move a few dozen notes every time you want to move one task or idea. Make sure, though, that you do not chuck aside ideas that sound similar to others but that are in fact distinct and should therefore be their own topic or task.

Prioritize

The next step is to begin ordering the notes by importance or, if necessary, in the chronological order in which tasks must occur. For instance, you may need to inventory all of your current design patterns before you do anything else. Or you may want to inventory all of your content, or clean up your HTML. You may feel that the most important thing to accomplish is to design and build a fully responsive navigation, or to update all of your content to be more small-screen-friendly. Whatever you and your team feel are the main priorities should be moved to the top.

Some things absolutely have to occur before others tasks can be done, so make sure to capture these, too, and place them in order of priority. As for the rest, their importance is subjective, so you should expect a lot of talking and discussions during this exercise.

When someone is arguing for one task to the detriment of another, make sure their arguments are grounded on user and/or business goals. It is hard to argue against, or for, someone who has a different opinion when their reasoning is merely based on personal preference.

For example, if a developer says, “I think we should work on implementing jQuery Mobile because I don’t like other JavaScript libraries,” ask how that will benefit the users of your site, the company, or the project. It is great to work on something exciting and fun that you are passionate and happy about, but you need to be aware when that only benefits your own sense of pleasure and accomplishment and adds no value to the overall project. As I have previously established, it is important to balance your team’s desire to make a site responsive with the other goals of the business; so it is important to be able to justify the time you will spend working on the project with sound business and user benefits, not just subjective personal likes and dislikes.

Break Down the Tasks

Some of the items you arrive at may be too large to be completed in a week or two. Some may even have to be completed in two different stages of the project. If someone listed, for example, “Rewrite our copy to be more mobile-friendly,” you may want to divide that

into two or more notes, such as “Rewrite copy of top section pages,” “Rewrite copy of home page,” “Rewrite help pages,” and so on. If you only have small chunks of time in between working on other projects to work on your responsive redesign, having these smaller tasks to fit in your schedule will be easier.

You should also, as a group, think about any tasks that will have to be done in relation to another note but that have not been listed. Still using the previous example, before you can rewrite any copy, you may have to hire a copywriter or define guidelines for copy and tone of voice.

The One-Hour Test

A good way to start breaking down the larger tasks in your prioritized list is to try to answer the following question:

If you only had one hour, what would you do next to move the project forward?

When you have only one hour to improve something, you focus on the task that is not only the most important, but also achievable in one hour—the task that will give you more bang for your buck, as they say. If you want to, or have to, you can even plan the entire project with this question in mind. This approach means you can more easily forego nice-to-haves, embellishments, and tasks that do not add real value to the final goal, which is to make your site work beautifully across any device.

You can increase the time limit to two hours, if that works best. But I do not recommend going over two hours, because then it is harder to focus on the next-most-important task that will get you closer to having a responsive site. The scope of what you can do in two hours or more is much broader and therefore less useful when your main difficulty is deciding what to do next.

You can propose this question before you start writing your wish list, so that the tasks the team comes up with are already broken down into smaller chunks. What if you make it 30 minutes? Can they think of anything that can be accomplished in that short time?

Here are some ideas of tasks you can try to complete in one hour:

- Look at your site’s analytics data for specific information, such as common devices or most-visited pages.
- Make an inventory of the types of images used on your site.
- Assess the quality of the content in one section of your site. Is it too long to read on a small screen? Are the images adding anything to the content?
- Measure the size of key landing pages.
- Perform accessibility tests, such as color contrast and testing font size.
- Test your site with one user.

Bear in mind, though, that there is a certain type of work that does not lend itself to being accomplished well when it is done in small blocks of time. Both designers and developers work better when they are allowed to focus on a certain task for long enough that they achieve a certain flow—or “get in the zone.” Usually this is desirable when designers need to create designs, when developers need to write code, or when copywriters need to, you know, write. These are the types of tasks that require a certain degree of creativity and inspiration, and sometimes of serendipity; and to create above-average results, it is important to schedule longer blocks of time that enable moments of uninterrupted work.

Define Stages

I find it a good idea to plan in some detail at least the first two releases of a responsive project during this planning stage. As with any other project, time frames and scopes will probably be tweaked, changed, and adapted to unforeseen circumstances. If you and your team know which priority tasks must be completed in stage 1, you will feel more confident about leaving some tasks for the second and further releases. You will not have time straight away to work on some of the things you would like to see fixed, but you will know that those are planned for the future and will not be forgotten.

Defining the second stage, even if loosely, and moving some of the tasks on the wall to a Phase 2 section, makes it easier to discuss which tasks should be done first. Instead of telling someone, “We will not do this,” you can say, “We will do this after these other things are completed.”

Determine What Is Out of Scope

One of the most important things to do when planning this responsive project is to list all the things you and your team will not be able to do, or fix, or even think about. It is also very important that everyone who is working on the project or will work on it in the future knows about this list—this is another example of why it is so important for everyone who will participate to take part in the project’s initial planning stages.

I think by now we have established that you do not have the luxury of fixing everything you think is wrong with your site. The things listed under “will not fix” should be documented and shared with everyone.

Some of the things other teams have left out of the first iteration of their responsive retrofit projects include these:

- Updates to the information architecture
- Updates to copy
- Responsive images
- Visual design
- Large screen layouts
- HTML updates
- Improvements to touch interaction

I am not saying that these examples should definitely be left for later stages of your project—that depends on the project’s limitations in terms of time and resources, and the needs of your site.

If you do not have a list of what you will not be working on, you may have to steer your team back in the right direction and toward the right focus. Designers and developers are opinionated folks and will have their own ideas about what should be done first. But you have spent a long time prioritizing the tasks that are most important for making your site responsive and that can realistically be accomplished in the limited time you have—no one should be running wild with their subjective idea of what should be improved when time is of essence.

You should also be aware of scope creep, which may lurk around the corner once things are set in motion. It is common to come across things that turn out to be essential to do, but that were not part of the initial plan. Many more times, things pop up that are not deal breakers and that can be parked until further releases. Before anything is added to the scope of the project, make sure to weigh its importance against all the other tasks that may have to be pushed out of the current iteration in order for these unexpected tasks to be completed. If you do not do this, the scope of your project can easily balloon.

Do Not Forget Testing

If you have not yet worked on a large responsive web design project, you may underestimate the amount of time that testing across devices and operating systems will take. The best way to incorporate testing in your plan is to embed it every step of the way. So if you are creating a prototype of responsive navigation, allow extra time for testing. If you are rewriting the copy on your site to be easier to read and follow on small screens, allow some time to test it on real devices before you consider that task completed.

The goal of converting your fixed-width site into a responsive one is to create an experience that is of consistently high quality across any device. That is why it is so important to incorporate testing right from the start.

At all costs, avoid leaving device testing to the end, before release. You are likely to be faced with unexpected bugs that you will have to fix at the very last minute, and they may take much longer to solve and retest than expected.

If you have a dedicated testing team, you should still make sure testing is done throughout the project, not just at the end of each release. It is important that you define the project plan alongside the testing team and align both schedules.

By the end of the steps outlined in this section, you and your team should have a much clearer, shared idea of the effort required to reach the outcome you have defined for your responsive project and how to fit the work into your schedule. You will have defined what success will look like at the end of the project, and the steps you need to take to get there.

Set Deadlines

All projects should have a deadline. You should be able to picture the world when the first iteration of your responsively retrofitted web site is out there. When you do not define dates for milestones and deadlines, and you rely only on your desire to see this project to fruition, you soon start to drag your heels, and other projects quickly move up your priority list.

If you usually plan frequent, small releases, you may want to try to define larger milestones in between the smaller ones, where key updates to the site are completed, such as “Release 3: All copy small screen-friendly” or “Release 5: New IA.” Whichever is your case, make sure these milestones and release dates are defined in your schedule and in everyone’s calendars.

Phase 1

With a better perspective on all the other commitments you have and how much time you have to work on the responsive project and when, it should be easier to decide on a date when you think it will be possible to release the first version of your responsive site. Even if the date you plan for release is several months in the future, it is important to define one and be sure everyone on the team is aware of it.

The date you settle on should be relatively flexible, without this fact being publicized outside of your team. However, you should move it only if there is a real emergency that needs to take priority, not if someone just deems something “urgent.” For example, if you find out that part of your web site is inaccessible on a common mobile device because of a poorly tested JavaScript-led navigation, you may have to fix this straight way and thus fall a little behind on the responsive retrofit. If the CEO does not like the text color you have used on links across your sites, you may have to politely present your schedule for the next few weeks and find a time to consider the request at a later stage.

There should be a little urgency about completing this project and having something released, so do not give yourself too much leeway or pad the schedule. You want to make sure whatever you release for the world to see looks and works great, but you should not give yourself and your team more time than you normally would for any other project.

Phase 2, Phase 3 ...

Ideally, at this point you should also define a date for the release of a second phase of the project, during which you can work on less pressing, but still important, tasks. This date will not be as set in stone as the first release date—as we all know, the further in the future a project is planned for, the more imaginative its corresponding scope and deadline can be. But having this second phase (and possibly following phases) defined will make it less difficult to forego some of the cool things that you want to do on your project but that are not fundamental for the first phase.

Rollout Strategies

A rollout strategy is very much dependent on your particular circumstances and the way you have planned your responsive project. Different types of rollout strategies include the following:

- Releasing a sandboxed experimental site that lives in parallel to the existing one and that is updated often
- Releasing the responsive site section by section, so old fixed-width sections coexist with new responsive ones
- A complete release of a redesign of the full site

The strategy that is most suited to you will depend on the type of site you have and how your users, well, use it.

My bank's web site has been undergoing a redesign for a few years now. Online banking is serious business, so you need to make sure your users do not make mistakes just because you have implemented a new design and things look different and have moved around. The bank's rollout strategy started with a call to action to any customers who wanted to try the beta version of the new site. It took a few months until all users were switched to the new design by default, and a few more months until that move was compulsory—at first, you could switch back to the old design, which I must confess I did. This is also a strategy followed by many other large web sites that have millions of users every month.

The second approach, releasing the responsive site section by section, is also adopted by some large organizations. When the release is broken into sections, with each release it feels like progress is being made at good speed; this can also make it easier to plan the responsive project, because each phase can be shorter. The downside of this approach is that it may confuse visitors who go between sections; but ideally, old and new versions of the site living side by side will not happen for a prolonged period of time.

If you decide to follow the third approach—a single, full release—you will have more work to do at first, but in following iterations you can focus on improving your new responsive site without having to go back to the old design or having to maintain two versions of the site at once.

The rollout strategy that is appropriate for your project will start to become evident as you begin planning your redesign, examining your schedule, and defining the work that needs to be done and the length of time the project will span. The choice you make will also be impacted by the way your team already works. If you are used to releasing early and often, you may want to follow the same strategy for your responsive retrofit.

■ **Note** If you have ever listened to Karen McGrane and Ethan Marcotte's "Responsive Web Design" podcast, you have heard Karen consistently ask her guests what their rollout strategy was. This makes for some very interesting listening, and you can also read about it in her book, *Going Responsive (A Book Apart, 2015)*.

Regardless of which rollout strategy you plan to follow, I recommend that you put in place a way for users to provide feedback to the design and development team. Taking the time to send feedback about a web site can be time-consuming, so make sure it is easy and quick for someone who has something to say to say it. Avoid asking the user to sign up for something. Ideally, you should only need an e-mail address (in case the user does not mind you following up on their comment) and the feedback itself. And remember to put in place a process that guarantees the feedback received is in fact read and acted on.

Managing Site Updates

It is very unlikely that you won't be required to keep your existing site up to date at the same time your team is working on responsively retrofitting it. In some cases, it may be possible to pause all updates to the site during the responsive retrofit work, but this does not happen often. The key thing to bear in mind and to avoid is the duplication of work. You are too busy to do the same thing twice more than the absolute minimum number of times, so even if you do not completely block any updates, you should limit them to only time-sensitive, high-priority ones and/or block the development of new features. If you have the benefit of not needing to update your HTML or content management system at the same time you responsively retrofit your site, that is a good way to decrease the number of places you have to keep things in sync.

The amount of effort involved in managing site updates will be directly related to your rollout strategy and whether you are building a new content management system. I would advise you to not be too precious about keeping your demos and prototype versions of the site totally in sync with your live site. Otherwise you will spend more time trying to keep these various versions in sync with each other than working on making your site responsive. The next chapter explores the management of content updates and editing.

Keep a Record as You Work Through the Project

The responsive retrofit of your site will probably go on for at least a couple of months, perhaps even a year or more. During this time, people will have meetings, make decisions, and change their minds many, many times, no matter how watertight you think your plan and scope are. You should make sure a summary of these conversations, any decisions that are made, and the reasoning behind those decisions are recorded in some way.

People can be absent from discussions for several reasons: they may be on holiday, sick, working from home, or just late to work. You need to be sure the content of any discussion that produces actionable tasks and decisions is accessible to them.

Several tools are available that can make this recording easy, keep comments in a central location, notify team members about new discussions, and make sure everything is searchable. Whichever tool you decide to use, make sure you do use it and that you and your team are thorough about keeping good meeting notes. One role of the project leader throughout the duration of the project will be to remind and nudge the rest of the team to keep good records of decisions and findings.

A popular tool used by many design teams these days is the ubiquitous messaging app Slack (see Figure 1-5). Slack offers the following:

- The ability to search across your entire message archive, including in attached files like PDFs and Google Docs
- Fine-grained notification customization
- The promise of no more e-mail
- Integration with Google Drive, Twitter, Dropbox, Google Hangouts, MailChimp, Twitter, Dribbble, GitHub, Heroku, Trello, Asana, Pivotal Tracker, Google Calendar, JIRA, and more

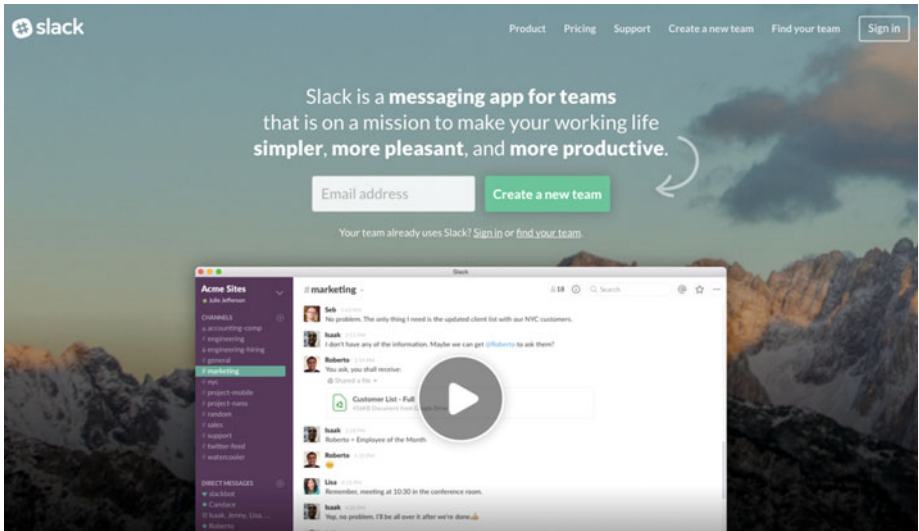


Figure 1-5. Slack, the popular messaging app

Another popular tool used by many companies large and small is the project-management app Basecamp, mentioned earlier in this chapter.

If you are already using a tool that works for you, stick with it. If you are choosing a new tool, it is a good idea to talk to your team and see if there are any tools they have used successfully (or not) that would more easily integrate with their workflow. Whichever tool you choose, its main attribute should be that it fits the way you work and will be used by the entire team consistently.

Summary

At the beginning of a responsive retrofitting project, the amount of work ahead can seem insurmountable. By following the steps and exercises that I have outlined in this chapter, you can tackle the project planning and quickly start breaking down and making sense of the long list of things you and your team need to do. By the end of this planning stage, you should

- Know who will be working on the project, and when
- Have defined a project leader
- Have defined a deadline for release
- Know how long the responsive retrofit project will take to complete
- Have a defined scope for the project

- Know how to best assign different types of tasks to a different number of people on your team
- Have agreed on what you will not do at each stage of the project
- Have a good understanding of the difficulty of each task you need to work on

Once you have a better idea of how your responsive retrofitting project will play out, it is time to get cracking with the work. The first element of your site that you should consider is the one that brings people to your site in the first place: the content.

CHAPTER 2



The Content Development Stage

We all know that content is the life and soul of the Internet. It is the reason people visit our sites, and the reason they come back over and over again (or at least we hope they do).

In the process of making sites responsive, we tend to simplify the design and remove unnecessary decorative elements, and our layouts tend to become cleaner. Because of this, the content increases in importance as it is propelled to the forefront.

Eileen Webb's words clearly capture the way in which content becomes even more important than it was before when we move toward responsive web design:

When you strip the pomp and circumstance out of a beautiful desktop view, you're usually left with a very straightforward, single-column, small-screen design that puts content front and center.¹

A lot can be said about the creation and management of content in a responsive web site, but most of those things can take an impressive amount of time to be brought to fruition. This chapter focuses on how you can improve your existing content when you do not have a lot of time on your hands, and when you have to choose very carefully what you focus that time on.

I assume that you do not have enough time and resources to work on a complete overhaul of your content strategy and content management system. This chapter provides some exercises and tools that can bring your site closer to giving your visitors, users, and readers a great reading experience when they access your site from any platform and any screen size.

Designing with Real Content

When working on a responsive redesign, you will hopefully be testing your site with real prototypes as soon as possible and, as much as possible, on real devices and emulators. It is important to understand that the best way to test the prototypes you will create

¹Eileen Webb, "Content Choreography in RWD," *Smashing Book 5: Real-Life Responsive Web Design—Part 1* (Smashing Magazine, 2015).

throughout this responsive project is by using real content, or at least content that is very close to the final content, rather than the typical “Lorem Ipsum” dummy text (“Bacon Ipsum”² is equally bad, by the way).

If getting hold of the final content before you start prototyping is not an option, you can design your responsive site using example content: content that captures the key message and expected length of each piece of text, but that is still likely to be changed later. This way, your designs and prototypes will be similar to the final content, even if you know they will suffer various rounds of changes when stakeholders provide feedback at different stages (probably later in the process than you would hope). And there are always last-minute changes that come up when you look at the final prototype that you have to account for anyway. By using example content, you will not be tempted to quickly remove a couple of words of your “Lorem Ipsum” text block to fit your designs better (not that we have ever done that, of course).

By looking at text and images that are close to the final thing, you may see ways to improve the flow or length of your copy to make for a better reading experience. In some cases, for instance, you may notice that it would be fine to explain certain aspects of your product in more words, without the pages becoming too long. And, more often than not, you can locate areas that need improvement because they are too lengthy, are too repetitive, and/or provide too little useful information for your users.

The way you go about collecting or creating example content can vary. In some cases, content production, user experience, and visual design are worked on by completely separate teams that do not even share an office. If this is your situation, then, assuming that you are planning on repurposing your existing content, you may want to carve out some time to suggest improvements or *small tweaks* to the content.

You will have to handle this exercise carefully, because you do not want to imply that other people have done their jobs badly. Let people focus on the new, responsive paradigm and on how your existing content can be greatly improved for it if you apply a few small changes. And make sure you tell everyone involved that this exercise will be part of the plan from the outset—few situations are more aggravating than showing up at a meeting to see someone present an “improved” version of your own work, without you knowing what is coming.

If you are closer to the people who are creating content, or if that is already part of your or your team’s role, suggesting example content will come more easily. But you should still inform the broader team and stakeholders that they should expect some changes.

Regardless of who is creating the content, if it is not you or your team, and you are handed wireframes that only show dummy text, politely ask for example or draft content, and explain how using it will improve your prototypes and designs.

Focus on Accessibility

Sometimes I feel that, even today, *accessibility* is a word that teams shy away from—one that scares many good professionals across all disciplines in our industry. We tend to avoid things we are afraid of, and we tend to be afraid of things we do not understand. A common reason for this fear is the misconception that making a site accessible means extra work that takes time and does not bring visible results. Another reason is the absence of a vision of how focusing on accessibility can deliver an improved user experience for everyone and higher profits for the company. On top of these reasons is a lack of understanding of exactly what can be done to make our sites more accessible.

²<http://baconipsum.com/>.

This kind of thinking leads to teams who only think of improving their site’s accessibility on an ad hoc basis or toward the end of a project, when all or most of the content, design, and development decisions have already been made and implemented. Accessibility should not be something you add to your site at the end of the project. Rather, it should be a part of the way you create your content, design your site, and build it. It should permeate the entire process from the start, and content strategy and content creation are not an exception.

In some jurisdictions, making sure a site is accessible is written in the law. But even in those cases, accessibility considerations have a tendency to be relegated to the final stages of the process.

Good Accessibility Helps Everyone

Considering users who have accessibility issues does not simply mean catering to users with permanent or long-term physical impairments, such as visually impaired and colorblind users, or people with reduced mobility. Accessibility issues do not just translate into the user’s physical or mental disability; they can relate to a temporary factor in the user’s circumstances. Consider the following scenarios, where the user’s context means they are having a harder time than usual accessing content or completing a task:

- Someone in a hotel on a business trip trying to do some work on a very slow Wi-Fi connection
- Someone holding a baby with one arm and trying to reply to an e-mail with the other
- Someone whose first language is Greek, who is trying to read a web site in English
- Someone in a wheelchair who is trying to fill out a form, only to discover halfway through that they need a document that is on the first floor of their house, but they are home alone
- Someone with dyslexia trying to find something on a site where the navigation is inconsistent across different pages

Obviously this is just a tiny fraction of the kinds of situations that can cause any typically “abled-bodied” user to have an accessibility issue.

Anne Gibson has created an alphabet of accessibility issues on The Pastry Box Project’s web site. She includes 26 examples of how real-life situations can hamper a person’s accessibility (see Figure 2-1).³ The list is well worth reading, sharing with your team, and keeping in mind the next time you are editing or creating a new piece of content.

³Anne Gibson, “An Alphabet of Accessibility Issues,” July 31, 2014, <https://the-pastry-box-project.net/anne-gibson/2014-july-31>.



Anne Gibson

Thursday, 31 July 2014

An Alphabet of Accessibility Issues

A is blind, and has been since birth. He's always used a screen reader, and always used a computer. He's a programmer, and he's better prepared to use the web than most of the others on this list.

B fell down a hill while running to close his car windows in the rain, and fractured multiple fingers. He's trying to surf the web with his left hand and the keyboard.

C has a blood cancer. She's been on chemo for a few months and, despite being an MD, is finding it harder and harder to remember things, read, or have a conversation. It's called chemo brain. She's frustrated because she's becoming more and more reliant on her smart phone for taking notes and keeping track of things at the same time that it's getting harder and harder for her to use.

D is color blind. Most websites think of him, but most people making PowerPoint presentations or charts and graphs at work do not.

E has Cystic Fibrosis, which causes him to spend two to three hours a day wrapped in respiratory therapy equipment that vibrates his chest and makes him cough. As an extension, it makes his arms and legs shake, so he sometimes prefers to use the keyboard or wait to do tasks that require a steady touch with a mouse. He also prefers his tablet over his laptop because he can take it anywhere more conveniently, and it's easier to clean germs off of.

F has been a programmer since junior high. She just had surgery for gamer's thumb in her non-dominant hand, and will have it in her dominant hand in a few weeks. She's not sure yet how it will affect her typing or using a touchpad on her laptop.

G was diagnosed with diabetes at an early age. Because of his early and ongoing treatment, most

Figure 2-1. Anne Gibson's alphabet of accessibility issues

Think of how your site or your pages would perform under these more strenuous circumstances. You can probably already think of a few ways in which you can improve your site for these users, right?

What You Can Do

It is very unlikely that you will have time to go through all of your content and correct any accessibility mistakes that may have slipped through. But you could perhaps begin by selecting some of your key pages or passages and improving the accessibility of your text.

Here are a few things that can get you started:

- Avoid long sentences and paragraphs, because some people have difficulty focusing on an idea for too long.
- Break up long blocks of text into smaller chunks by using headings and bulleted lists.
- Avoid complicated words that are not necessary.
- Make things easy to understand for people for whom your language is not their first language.
- Consider including a table of contents at the start of a long page.
- Use the active voice instead of passive voice if possible, and avoid double negatives.
- Provide text alternatives for media formats such as video and audio.
- Make sure images and other graphics have alt text or a detailed caption.
- Provide a glossary for technical terms or abbreviations.
- As much as possible, keep your site’s navigation and key interactions in the same location throughout.

The list could go on—you can find more design- and development-specific accessibility points in the final two chapters of this book.

If you do not have enough time to edit existing content, you may want to consider proposing a plan to review sections in the future, as they go through updates as part of the normal maintenance cycle.

Good accessibility should be the responsibility of everyone on the team, and it should happen instinctively as part of the design process, but we all know that this is easier said than done. Ideally, someone on your team (and people on other teams that are involved in the production of any type of content for your web site) should be nominated to be accessibility champion or accessibility advocate. Focusing the team on accessibility throughout every stage of your projects should be part of their job description. It may be easier to start this on your own team, where you have more power to influence the process; but hopefully, as other teams see how you put this into practice, it will be easier to establish accessibility advocates throughout the organization.

Accessibility and Usability

I think it is worth clarifying that *accessibility* and *usability* are not the same thing. However, they do overlap.

To make a long story short, usability relates to how easy your site is to use, and accessibility relates to how easy it is for users to “perceive, understand, navigate, and interact with” your site’s content.⁴

⁴World Wide Web Consortium, “Introduction to Web Accessibility,” September 2005, <https://www.w3.org/WAI/intro/accessibility.php>.

When I speak about improving a site’s usability, I mean you should make sure users know what to do to accomplish their tasks and do not feel lost on the site. They probably should not have to read long, complicated instructions, and they should be able to undo or correct their errors easily.

When we say a site must be accessible, we mean any user, using any type of device or any type of technology, and with any type of physical or mental disability, should be able to access the site and its content.

These two concepts work together to create an ideal user experience. You should give thought to both and actively improve your site with both in mind.

Reviewing Your Existing Content

The first unavoidable step when improving your content is to carefully catalogue and examine it. Yes, I am referring to doing a content inventory, followed by a content audit. Before you can make any decisions regarding your content, you need to know exactly what you have to deal with: how much content your site holds, and the state of each piece of content.

Content Inventories and Content Audits

Some people use the terms *content audit* and *content inventory* interchangeably. But if you want to differentiate the two, you can say that a content inventory does not necessarily imply an evaluation of the content—just the inventory itself—whereas a content audit does. A content inventory assesses the content in a quantitative manner, and a content audit measures it in a qualitative manner. You can audit the content after you do the inventory; but they can both be done in the same document and even at the same time, so you do not have to physically separate them in two documents.

A content inventory and a content audit are useful tools to start estimating how much time you will have to spend working on the site to make it responsive. These estimations likely will not be accurate at first; but creating these documents is a good way to get a bird’s-eye view of your content and the effort involved early in the project, which may help you plan later stages. The content audit can become a document that is used throughout the responsive redesign to understand the complexity and amount of work ahead, even in projects where the content will not be rewritten or tackled in the initial phases.

Many people find this type of exercise tedious, because it involves long spreadsheets and mechanical work—it does not have the glamour of designing and building cool-looking responsive prototypes. But I find it exhilarating, because the product of a content audit can (and should) provide an important overview of all the parts of your site—even the ones that tend to be forgotten. Now, open a new spreadsheet, and let’s get cracking!

Doing a Content Inventory

You may already have created a content inventory for your site at some point. If you did, you may want to start from there rather than from a blank file. But you still need to go through all the content of your site, page by page or screen by screen, because it is very likely that your content inventory is not up to date.

Make sure you take your time and do not rush through the inventory. This is invaluable time that you will spend learning about and understanding your content. You may want to put on your headphones, get some music rolling, and become completely immersed in your content.

The information you include in your content inventory should be adapted to what you want to know about your content, and you can change this as you go along and as your project evolves. You can begin with the following column headings:

- *Title*: The title of the page.
- *Location*: The page's section in the IA. For example, "Products ► Blinds & Curtains ► Roman Blinds."
- *URL*: If there is a clickable link that takes you to the section, it should be listed here.
- *Copy*: A brief description of the main content.
- *Images*: Include all the images used in the section. You can also specify types of images and call out any image that may pose a problem in a responsive redesign. You may have something like this: "5 product photographs, 1 diagram—will need to be redone in HTML & CSS." Also note whether there is alt text or a caption for each image, or whether this needs to be created.
- *Other media*: Just as for images, list media assets such as videos and audio, and highlight any potential risks and any assets that will have to be re-created. If you prefer, you can have a different column for each type of asset.
- *Owner*: Who owns the content, and whose responsibility it is to edit it.
- *Author*: The author may be different from the owner, so it may be a good idea to list them separately.
- *Last update*: If you can find out, indicate the date when the page was last updated.
- *Notes*: Any other remarks you would like to add.

In a more thorough content inventory, you should also include things like e-mail newsletters, printed materials, and anything else that holds your content. Because you are creating the inventory to aid you in a responsive retrofit project, you may have to leave those types of content for a later stage; but keep them in mind for the future.

There is no fixed way to perform a content inventory, so you should feel free to change the process to whatever suits the needs of your site. You may want to include more information or structure your document in a different way. And remember that your content inventory can be edited as you go along; it is a living document that will be useful for your team as a guide to the work that will be done during your responsive retrofitting project.

Doing a Content Audit

The main goal of a content audit is to evaluate your existing content. You can do the audit at the same time you catalogue your content, or as a separate step.

An easy way to start evaluating your content is by using the ROT method, a common content strategy exercise. ROT stands for *redundant, outdated, and trivial*. The goal of this exercise is to identify content that falls into any of these categories.

Here are a few examples of ROT content:

- Broken links and missing pages
- Content that is repeated on different pages or in different sections on the same page
- Events that are in the past but are presented as being in the future
- Inaccurate contact details
- Old features portrayed as new features
- Pages that no one ever visits

As you identify and record your site's ROT content, you can also make a note about the effort needed to improve it. In certain cases, it will be easy to fix the problem, such as incorrect contact information or broken links. In other cases, a longer, more complex process for updating or curtailing the content may have to be set in motion—and perhaps may be out of scope for your responsive retrofitting project.

You can also do a more in-depth analysis of your content that registers in more detail what the state of each piece of content is and the risks associated with it that are relevant to making your site responsive.

Taking your content inventory as the starting point, create a spreadsheet that includes information such as the following:

- Summary of the content
- Type of content
- Risks it may bring to a responsive site (for example, it is too long, it is a complicated table, it is in a format that is not readable on smartphones, and so on)
- Does it need editing? If so, should it be edited before the responsive project is launched? At which stage?
- Estimate of how long, or how much effort, it may take to be improved (or taken care of in some other way). You can separate quick fixes from more complex ones in different columns for easier scanning.

Some of these questions may be hard to answer at first—that is why you should take your time with your content to truly understand its ins and outs. Just like the content inventory, the content audit is a living document that you can keep adding to. And much like the content inventory, the content audit should be relevant to the needs of your particular project.

Bear in mind that this audit does not just relate to copy. You should make sure you include other types of content in your document, such as images, PDFs, videos, audio clips, and so on.

There is no right way of going about your content audit. Starting with the ROT methodology is a quick way of getting stuck in the audit, but you can assess your content in other ways.

Remember that you should take advantage of the time you are going to spend with your content. Take it all in, read it carefully, and take thorough notes. You may not be able to act on many of your findings, but the audit will be an invaluable document that you can use to make the case for and plan further improvements to your content.

Quick Content Inventories and Content Audits

If you simply do not have time to do an exhaustive, full inventory of your existing content (and you really must have an excellent excuse not to), you may want to consider listing portions of your site, instead. These can be

- Sections
- Modules
- Screens
- Passages
- Anything else that works for your site

It may be tempting to employ some sort of automated script that collects your inventory data for you, but it is important that you get your hands dirty and immerse yourself in your content, learning it inside and out. So, yes, you also need a spreadsheet for this type of inventory—trust me, you will not be able to avoid working with spreadsheets when dealing with content!

In her book *Content Strategy for Mobile* (A Book Apart, 2012), Karen McGrane (p. 100) suggests a way in which you can do a speedier content audit, if time is lacking. In this case, you audit only a subset of your full content inventory. Karen lists three ways in which you can determine what content to audit:

- *Breadth of content*: Can you focus on only one section?
- *Depth of content*: Do you need to audit all levels of your site, or only down to a certain level?
- *Content types*: Can you analyze the different content types instead of every unique page?

Even this simpler type of content inventory and content audit will provide you with documents that you can use to estimate the amount of effort needed to bring your content to a good state and, later, to check things off as you work through them.

Ongoing Maintenance

One of the biggest issues to overcome when working on a responsive retrofit project is maintaining an existing, evergreen site while developing a new site.

Avoid Duplicated Work

In order to keep duplication of work at arm's length or reduce it to a minimum, you should plan what types of changes and updates you will be carrying out and when, during the time you are maintaining two code bases. The amount of duplicated work that you will perform will be closely linked to the following:

- Whether you are developing a new content management system.
- The type of site you run. E-commerce sites and publishers are unlikely to be able to stop updates to the site.
- The type of roll-out strategy you will set in place.

If you decide to stop or cut back in any way on the number of site updates your team will work on, make sure every relevant person and team is aware of your plans. You do not want people to be caught off guard when you tell them you are not going to work on an update that may not seem important to you, but that they have been working on and planning for a long time. Good, up-front communication is paramount in order for a project of the scale of a responsive retrofit to work, not just when it concerns content, but with regard to every aspect of the project.

Phasing Content Updates

A relatively straightforward way to balance ongoing maintenance on the old site with content changes for the new responsive site is to plan content updates based on existing deadlines for other projects. Let's imagine you are working on your responsive project over six months and that you know that in month 2, one of the sections of your site (let's call it section A) will be updated (independently of your responsive project) to reflect changes in the organization. And in month 8 (two months after your responsive project has been made live), another section of your site (let's call this one section B) will undergo content updates to reflect a seasonal campaign.

With this information, you can prioritize content updates to section A. You can make the decision to, alongside the content updates related to the scheduled campaigns, also improve the content so that it is user-friendly for any kind of device, big or small, improving not only copy but images and other media formats. And you can decide that you will not make any changes to section B as part of the responsive project, leaving those to be made as part of the seasonal campaign updates. Neat, huh?

As I mentioned earlier (and as is worth remembering), you need to make sure all the different people involved in content creation and in updating your site are aligned and working toward the same goals and the same deadlines. By prioritizing content updates of sections that are already scheduled to be changed or improved, and working on those sections as part of, or in parallel to, the responsive retrofit project, you can more easily

deprioritize the rest of your site. If a section is undergoing updates for a specific go-live date, it should be safe to assume that people are hoping that section will see increased traffic soon after release, so it makes sense to improve these sections first.

Updating and Simplifying Existing Content

Once you have made the decision to update certain content, you need to know exactly what you are going to do to it: what changes can be made that will benefit your readers but that are also possible given your limited time and resources. The types of updates and improvements you can make to your content are wide ranging, from quick, less contentious ones to more involved ones that may imply a shift in your content strategy. I cover both types here, but my main focus (and this book's, too) is on quick wins.

The way you approach updating your content will be closely linked to the roll-out strategy you will be following. If you are going to launch your new responsive site section by section, it makes sense to have the copy updates follow the same schedule and the same priority. If you are going to release everything at once, you may want to focus on smaller improvements across the site and larger improvements only to key pages or key user journeys.

Quick Wins

You may be feeling overwhelmed by the number of things you want to improve in your web site's content. Fear not. The trick is to start small and persevere. There are some things you can do to begin editing large amounts of copy and other types of content without requiring a huge amount of time or many people.

Erin Kissane points out this very same approach in her book *The Elements of Content Strategy* (A Book Apart, 2010):

[Q]uick fix remedies (fix broken pages, correct misspellings and inaccurate facts, switch inaccessible files to accessible versions) that can be applied to the current site without slowing the rest of the project down

If you conducted a content audit, you have highlighted some of the smaller, achievable tasks that you can perform on your content and that will instantly improve it without stopping the rest of the project from moving forward.

Some of the most common things you should be able to do easily are as follows:

- Correcting typos
- Correcting inaccurate facts and details
- Fixing broken links
- Removing redundant sentences
- Removing unnecessary adverbs and adjectives

- Making sure the <title> tag of the page matches its content
- Correcting inaccurate link descriptions (does it say Download but not start a file download?)
- Correctly labelling links that do not lead to other pages but rather start downloads, such as PDFs, videos, or ZIP files
- Improving link descriptions (“Read the full Acme Corp case study” instead of “Read more”)

If you want to go a little deeper and can spend a bit more time crafting your content, try to follow some of the following best practices:

- Keep the message clear and focused on each page and section of the site.
- Avoid repetition.
- Be consistent with call-to-action links, headings, and so on.
- Link to the same content using the same label across your site.

Not many projects have the luxury of infinite time and resources, but in the case of a quick responsive retrofit, that is especially true. This type of task—something that takes little effort or that is not contentious—should be the one that you and your team focus on delivering, at least in the initial release of your updated responsive web site. It is a little like doing the dishes and making the bed. These chores are quick and easy to do, but if you do all the other time-consuming spring cleaning tasks without them, your house will still look untidy.

Shorter Is Not Better

Rewriting content so that it is more user-friendly for someone who is reading it on a small screen does not necessarily mean you need to make the content shorter. Only you will know whether your web site’s content is in need of a trim. You would not cut down a long-form investigative journalistic piece because it is too long to be read on an iPhone. You also would not cut down bits of a user manual because it is too long.

Each piece of content needs to be evaluated not only in the context of the entire site, but also on its own merits. Your web site may contain short blog posts and long articles, or it may contain product information pages with lots of images, bulleted lists, and user manuals. If your only goal is to make your content shorter, you run the risk of removing important information that helps explain your product or that makes the story you are telling easier to understand and engage with.

To be able to make this type of decision, it is also important to have concrete findings from testing. Your users know best. More often than not, the people who are building a web site and creating and maintaining its content are not the web site’s target users. Just because you would like your pages to be snappier and have less jargon does not mean that’s what your users expect or need.

You can improve existing content, while keeping its length, by making sure you capture the reader’s attention immediately:

- Make sure your titles and headings include the words that your visitors are looking for and expect.
- Start with the main idea of the page, section, or paragraph, and only go into more detail further down the page.
- When possible, break up content into bullet lists.
- Be consistent when referring to key elements in your content. If you say “You should buy our personal insurance” in the beginning, do not say “Read more about our personal protection” later—readers will wonder whether those are the same thing.

You may be thinking that these tips are not easy to pull off on a large site without investing a lot of time—which I know you do not have. Just like many suggestions in this chapter, these may have to be planned for future projects or later stages of the responsive retrofit project. It is good to have an idea of what can be accomplished if you have a little more time to spend. As you go through your content audit and content inventory, you can note improvements that can be done to each piece of content but that will not necessarily be done straight away.

Less Content May Be Better Content

In which room do you think it will be easier to find something: a room where the surfaces are clear and the shelves and cupboards contain things that are well organized and that you can easily glance at? Or a room that has piles and piles of stuff all over the surfaces and the floor, and where the shelves and cupboards are overflowing with even more things?

That was an easy question. But for some reason, this idea is harder to grasp when it comes to your site’s content.

Erin Kissane explained this idea aptly in *The Elements of Content Strategy*:

So what does it matter if we have too much content? For one thing, more content makes everything more difficult to find. For another, spreading finite resources ever more thinly results in a decline in quality.

Make your users’ life easier by giving them less stuff to sift through to get to what they’re looking for.

One solution you may want to consider when trying to reduce the amount of content on your site is the creation of an archive or a resource center, where dated content can be moved. This is exactly what we did with Ubuntu Insights (see Figure 2-2).

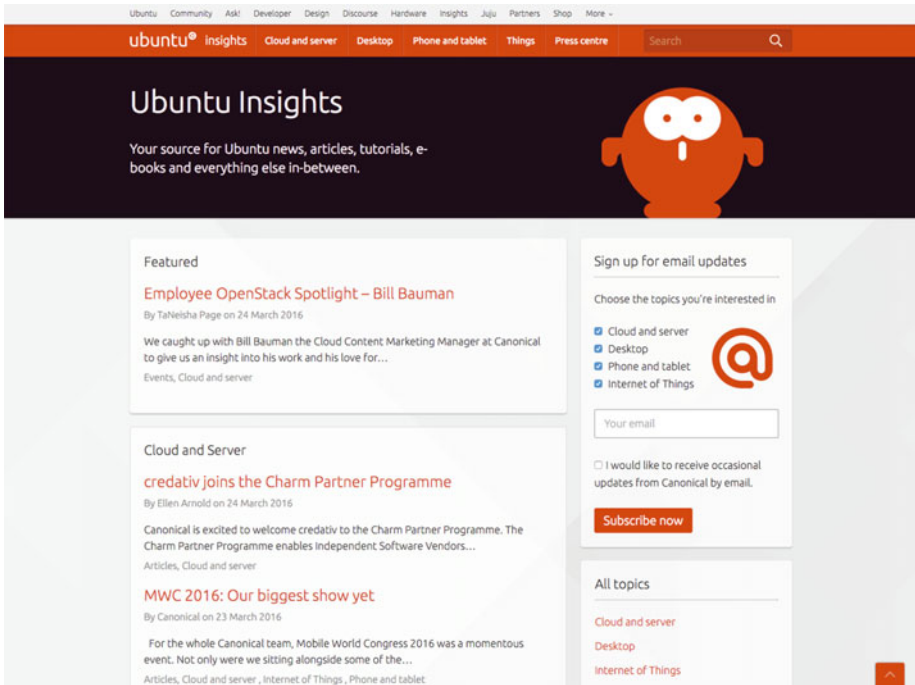


Figure 2-2. Ubuntu created a separate resource center to hold dated content, in order to simplify the main site

CASE STUDY: UBUNTU INSIGHTS

When working on the responsive redesign of ubuntu.com, the large amount of various types of dated content was an obstacle to simplifying user journeys, pages, and the site’s IA and navigation. The web site included not only product pages, download pages, and contact forms, but also lots of news articles, press releases, events, webinars, case studies, tutorials, and other help pages.

The solution the web team proposed to this problem was the creation of a separate site that served as a repository of the dated content that was bloating the marketing site. By moving this content to a resource center, the main site could be simplified, and it would also become easier to inject product description pages with fresh, new content on a regular basis; this content would come directly from the new resources site.

This had an immense impact on the IA of the site, because it massively cut down the number of pages that lived under the main site’s umbrella. It also made it easier to populate ubuntu.com with fresh information in a more regular, structured way. And

it made it simpler for content creators to, well, create content, because the new web site's admin panel was more user-friendly. We ended up with fewer pages and, in many instances, shorter ones, too.

The creation of a new project and another web site to maintain is, of course, not an option for many teams. But thinking about your content and structuring it into different categories that can be cross-pollinated throughout your pages may be useful when trying to simplify long, convoluted passages.

DO NOT BREAK THE WEB

If you do move and/or remove pages or sections from your site, make sure those URLs do not become broken. You do not want people to arrive at your site and immediately be faced with an error or a dreaded 404 page. You should make sure the old URL redirects automatically to the new location, if you moved content around, or to a page where the user can find similar information if you deleted the page. An alternative to deleting an out-of-date page is to include a message at the top of the page that clearly explains that the content is now out of date. Whichever path you choose to take, avoid at all costs creating broken links.

Reduce Cognitive Load Without Restricting Access to Content

The nature of working on a responsive redesign of your site implies that you will not be separating desktop users from mobile users with a View Full Site or View Desktop Site link. This pattern indicates to the user that the site they are looking at is in some way incomplete, missing some of the information or features of the “full site.”

I do not think this is what you set out to accomplish when you and your team decided to embark on your responsive retrofit. You want to make sure your content is accessible from any device, whatever its screen size or capabilities. This may happen in different ways between devices; but whatever you do, be careful not to remove the ability for users to access content when they view your site on a small-screen device.

It is risky to assume which content people will not want to see in certain situations. No amount of data that you can gather to understand how people are using your site will cover all user scenarios.

I keep going to back to Erin Kissane's *The Elements of Content Strategy* for her sharp insights. And I am about to do it once more. Regarding the practice of hiding some content for mobile users, Erin says:

[A]ssumptions about reader context ... will never be perfect. Always give readers the option of seeing more information if they wish to do so.

Notice how Erin does not say you should expose all your content at once. Rather, you should provide the *option* to access all content.

You can have a simplified and more linearized layout for small screens, but that does not mean less content overall. There are design patterns you can use to make small-screen layouts less complex without removing access to content. Running the risk of entering design turf, it is worth noting three common patterns that can be used to reduce the cognitive load on small screens.

Show and Hide Content with an Accordion

The first of those patterns is the *accordion* (see Figure 2-3). An accordion can be useful when the content it holds is optional for the user or secondary to the main content, such as when you need to show full technical specifications on a product page.

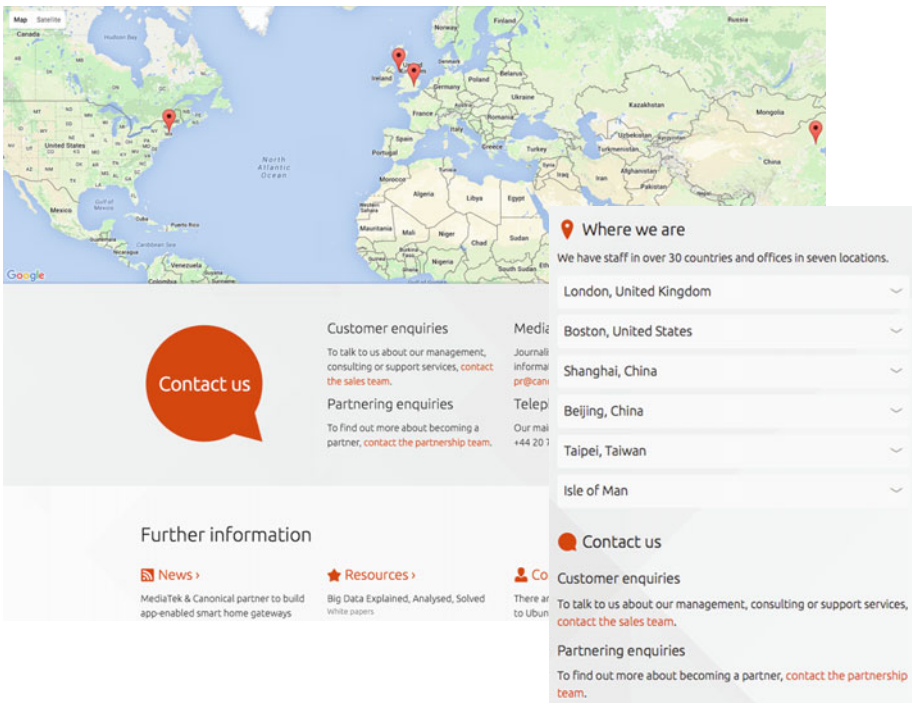


Figure 2-3. Canonical.com displays an interactive map with office locations on large screens and an accordion on small screens

In theory, an accordion is composed of at least two sections that can be closed, where only the titles or headings of those sections are showing. If a user wants to see the content on one of the sections, the user clicks its heading, and the section opens.

The classic pattern also dictates that when one section of the accordion is open, all other sections are closed. But if you do not want to be too strict, you can give users the ability to open more than one section at once.

Accordions are commonly used for frequently asked questions, even in large-screen designs, because they let users skim through the questions without having to scroll past answers they are not interested in.

Before you use an accordion or any other pattern that hides content, think carefully about whether it would be easier for the user to scan the content if you left it visible at all times instead.

Divide Content With Tabs

The second useful pattern to keep in mind is *tabs* (see Figure 2-4). Tabs are in a way similar to accordions in that they divide the content into smaller chunks and hide from view content on unselected tabs.

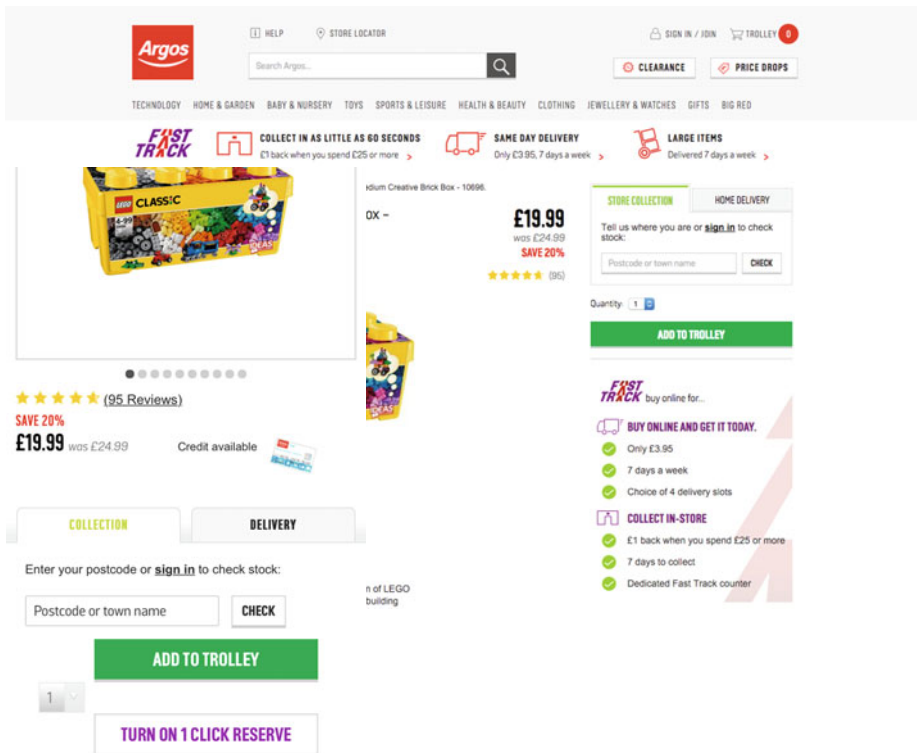


Figure 2-4. Argos uses tabs on both the large- and small-screen views to differentiate between the collection and delivery options

Tab headers are usually laid out horizontally (especially on small screens) but can also be laid out vertically. The content of one of the tabs is in full view, and when a user clicks one of the headers, the content below that tab comes to the foreground; the other tabs are hidden behind it.

Unlike accordions, tabs more commonly exist in the same format (as tabs) in large-screen designs; but they can also appear only on smaller screens, when showing all the content at once would make the design too crowded. It is worth noting that, in some cases, tabs in a wide layout are converted into an accordion in a narrow layout.

Tabs are popular on product pages to divide different types of content, such as product details, technical details, customer reviews, and so on.

Remember that, although *simplification* is the operative word when responsively redesigning a large-screen design, you do not have to simplify at the expense of the content available to your users. You may want to reduce the cognitive load of your pages on small screens, but be careful not to do it by making your mobile view less useful.

Expand Content with Read More

Finally, the third design pattern to consider, and perhaps the most common of the three, is *Read More*. This pattern is mostly used to cut off a block of text that may be too long to display on a small screen and that the user may not necessarily want to read in full.

A typical place where the pattern is used is in lists of comments or reviews, which can be very long. When a comment is longer than a certain number of words or lines, the text is cut off at that point; if the user is interested in reading the rest of the comment, they can tap the Read More link, and the text block expands to show the entire content. In some cases, the Read More link is replaced with a Read Less link that returns the text block to its original state.

When using this pattern, be sure the Read More link is not just hiding a couple of words, but rather a substantial amount of content. Otherwise, the user will feel as though the link is adding an unnecessary obstacle between them and your content.

Have a Plan

Good content strategy is not defined quickly in a couple of hours or a day or two. It involves a substantial amount of time dedicated to research and analysis, and it requires the ability to guide what can sometimes be a considerable number of people to unify toward a common goal in the way they create, edit, and publish content.

WHAT IS CONTENT STRATEGY?

The discipline of content strategy covers the planning, creation, and maintenance of any of type of content, be it text, audio, video, or other formats. The strategy should be developed according to the business and user goals of the particular product or company; it relates not only to the actual management of time and resources but also, and probably most importantly, to the definition of the actual approach to and objectives for the content.

Content strategy does not have to be done or implemented by someone who holds the Content Strategist job title—and in many cases it is not. It can be part of the responsibilities of roles as wide as copywriters, user experience designers, project managers, and product designers, to name a few.

It is unlikely that you will be able to completely rethink (or even create) your content strategy within the scope of this project. If there is currently more than one instance of your site—such as different desktop and mobile versions—you may even have different teams working separately on the production and maintenance of your content.

The transition to responsive may have a large impact on your content strategy and the way you produce and maintain your content. It may involve phasing out one or more sites to create a unified site where all your content lives. If different sites currently hold different versions of the same content, at some point you will want to define what stays and what goes. And that exercise will also involve defining new processes, and perhaps new teams and new roles. This is not an easy job.

You do not want to embark on this mission lightly, and you should not make assumptions about what a perfect content strategy looks like for your company without considering all aspects of the editorial process. Erin Kissane agrees:

Content recommendations made without consideration of available resources are unlikely to result in success.

I swear this is the last time I will quote Erin's *The Elements of Content Strategy* in this book. (Or is it? You will have to read on to find out.)

The definition of a sound content strategy for a substantially sized web site is a topic that can fill a very thick book. But the goal of this book is to give you a jumpstart on your long-overdue responsive retrofitted site, while assuming you do not have a lot of time and resources to design and implement massive changes. If budget allows, you may want to consider hiring a consulting content strategist to help you define the best approach for the future, or to at least get you started.

Defining a Style

It may come as a surprise to you, but many large, sprawling web sites do not have a copy style guide or a voice and tone style guide. Having a style guide is not something that only newspapers should worry about. A style guide provides writing consistency across different authors without removing their ability to write in their own voice.

In many cases, the move to a responsive site is an opportunity to introduce new habits and processes within your team and the larger organization that focus on best practices not only in terms of user experience design but also in terms of copywriting and coding. The creation and adoption of a style guide for your site if you did not have one before may sound like too much effort to put in on something that is not completely related to your goal, which is to have a beautiful, usable, responsive web site. But as much as you want visual consistency through a design pattern library (which I go into more detail about in Chapter 3), that consistency should start in your actual content.

If you cannot spend a lot of time creating your own style guide from scratch, you may want to use MailChimp’s Content Style Guide as a starting point (see Figure 2-5). MailChimp’s style guide is released under a Creative Commons license and is publicly available on GitHub. Although a lot of its content probably will not be appropriate for your specific needs, it provides a structure and can give you lots of ideas about what to include in your own guide.

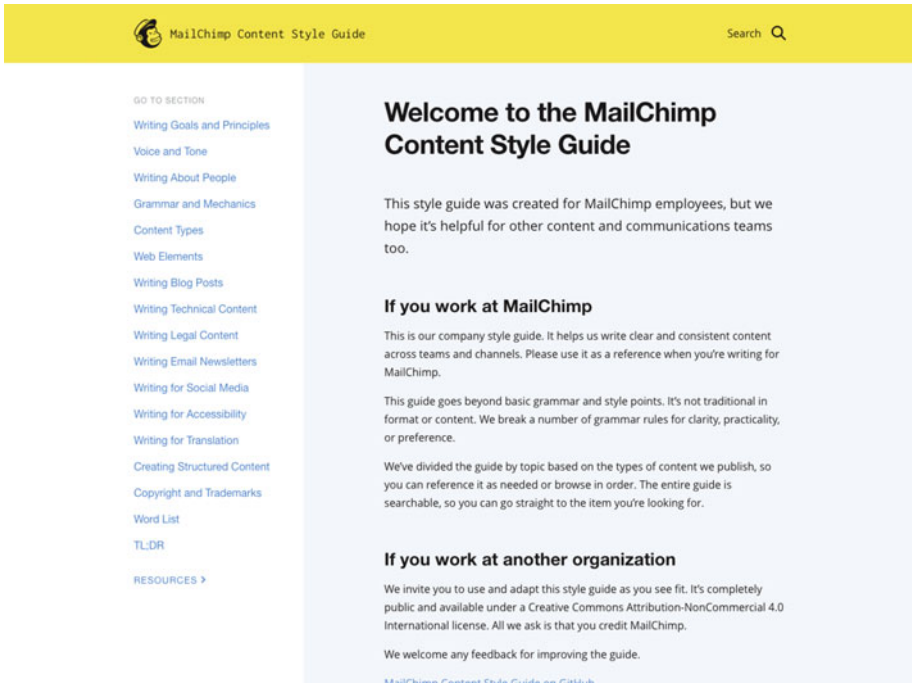


Figure 2-5. MailChimp’s acclaimed Content Style Guide

Another option, if you do not have a lot of time to spare, is to look into existing style guides, such as *The Chicago Manual of Style* (see Figure 2-6), the *Guardian and Observer Style Guide* (see Figure 2-7), and the *New Oxford Style Manual*. Many organizations follow existing authoritative style manuals from publishers or schools, because these manuals tend to be thorough and prescriptive and there is little else to add to them. But do not let an existing style guide overrule what your particular case needs. You can use it as a starting point, but you will probably have to adapt some rules to your circumstances.

The Chicago Manual of Style Online

HOME | THE CHICAGO MANUAL OF STYLE | PREFACE | CHICAGO STYLE BOOK | TOOLS | HELP

16th edition | 15th edition

Account Administration | Log In | Shopping Cart

Table of Contents

List of Figures

List of Tables

Index

Go to 1

Go to 12th Ed.

Contents

Preface

Acknowledgments

Part One: The Publishing Process

1 Books and Journals

2 Manuscript Preparation, Manuscript Editing, and Proofreading

3 Illustrations and Tables

4 Rights, Permissions, and Copyright Administration by William S. Strong

Part Two: Style and Usage

5 Grammar and Usage by Bryan A. Garner

6 Punctuation

7 Spelling, Distinctive Treatment of Words, and Compounds

8 Names and Terms

9 Numbers

10 Abbreviations

11 Foreign Languages

12 Mathematics in Type

13 Quotations and Dialogue

Part Three: Documentation

14 Documentation I: Notes and Bibliography

15 Documentation II: Author-Date References

16 Indexes

Appendix A: Production and Digital Technology

Appendix B: Glossary

Bibliography

Index

About The Chicago Manual of Style | About the University of Chicago Press | Terms of Use | Privacy Policy | Site Map

The Chicago Manual of Style (16th edition text) © 1949, 1989, 2003 by The University of Chicago. The Chicago Manual of Style (16th edition text) © 2014 by The University of Chicago. The Chicago Manual of Style Online © 2006, 2007, 2012 by The University of Chicago. The Chicago Manual of Style is a registered trademark of The University of Chicago.

Figure 2-6. The Chicago Manual of Style: you can browse the table of contents, but you need a paid account to access the full contents

free become a member sign in subscribe search jobs dating more - UK edition -

the guardian
words of the year

UK world politics sport football opinion culture business lifestyle fashion environment tech travel browse all sections

home

The Guardian and Observer style guide

Guardian and Observer style guide: A

'Style to be good must be clear. Clearness is secured by using words that are current and ordinary.' Aristotle

Follow the style guide on Twitter: @guardianstyle

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Wednesday 23 December 2015
14:51 GMT

75 Shares

Save for later

A

all mouth and trousers

A - style guide Illustrations: Photograph: Jakob Heineke

a or an before H?

Use an before a silent H: an heir, an hour, an honest politician, an honorary consul; use a before an aspirated H: a hero, a hotel, a historian (but don't change a direct quote if the speaker says, for example, "an historic"). With abbreviations, be guided by pronunciation: eg an LSE student

A*

Most popular

Neurosurgeon: Blackwell fight 'should have stopped three rounds earlier'

Can you solve it? The logic question almost everyone gets wrong

Did you solve it? The logic question almost everyone gets wrong

Welcome to Dragon School - the lair of the British acting elite

David Cameron accused of racial profiling in London mayoral letter

Figure 2-7. The Guardian and Observer Style Guide, available online

At the very least, if your content creators are writing in English, you should define which dialect your company should follow. Just by saying you are writing in British English, a lot of rules will be defined, not only in terms of spelling but also in terms of punctuation.

You should also consider things like capitalization. Should titles and headings be sentence case or title case? Should conjunctions and articles be capitalized in title-case headings? Are you going to capitalize job titles? What about your product names?

If you have to use technical terms, you may want to start a glossary for your authors to use as a reference, which can be expanded. A glossary can also be used to clarify spelling of words and terms that can have variations, like *e-mail* or *email*.

Just like a design pattern library, a writing style guide should not become stale. You can start by defining a few things and expand the manual as questions arise and clarification and guidance are needed. Even if you have a thorough style guide that covers most eventualities, your company may create a new product or service that needs to be added to the style guide, so be sure to keep it up to date.

Most important, though, your guide should be easily accessible and shareable among everyone who needs to use it and consult it. It probably will not be possible to update all of your content to match the style guide rules if you did not have one in place before, but you can make the decision that any new content, or any pages or sections that are updated in the future, should also be updated to follow the agreed style.

Improving Your Content Management System

Every responsive retrofit project is different, and priorities vary immensely. However, completely updating your CMS is probably a stretch too far if you are trying to cut down the scope of your project as much as possible.

Depending on how or even whether your content strategy and organizational structure will change when moving to responsive, you may have to create a CMS from scratch or change functionality extensively in your existing one. The following suggestions are likely to be out of scope for your responsive retrofit project, but they are worth considering if you can squeeze them in. They are not dramatic but rather add small improvements that can change for the better the way your content is produced.

Remove Inline Styles

Some CMSs allow authors to include inline styles (HTML and CSS) in the main content area. We have all heard horror stories of content authors and web site owners who have singlehandedly defaced a beautiful web site by formatting text as they please (think of a bold, italic, red, justified, Comic Sans text block inside three nested tables). Switching off this ability in your CMS may not be too complicated, and if so, it is something worth doing. If damage can be done by adding inline styles to a large screen design, imagine what can happen when your web site is being viewed across a wide range of devices and screen sizes!

Even if adding this constraint to your CMS is not an option, you may want to consider going through your CMS entries and delete inline styles that are already there. They have a tendency to break responsive design layouts particularly badly.

Remove Styling Options

Following from the previous point, some CMSs that use What You See Is What You Get (WYSIWYG) text entry to make it easy for authors to format and style text have the ability to turn off some of those very same styling options at the flick of a switch. If this is your case, you may want to consider doing this and limiting the amount of control authors have over layout.

Remember: the author's job is to create and edit good content. If they get stuck on perfecting the layout of the content, making it look just right on their own screen, they may well be counteracting the work that has gone into making the site responsive in the first place.

Add Structure to Your Content

Are your content authors faced with an input field for the title and then a large text-input field where they can put everything else? Karen McGrane calls this big input field the “giant blob” in her book *Content Strategy for Mobile*. Think about how you can start breaking this big blob apart and dividing your content into smaller chunks.

Let's take a look at a common content type, an event, to show how breaking the content into its constituent parts can be achieved. If authors decide themselves where in the blob to include information such as location, start and end dates, venue, and pricing information, even if they try to be consistent, there is always room for error, and your event's metadata will be buried deep within a block of text.

Consider dividing this information into separate entry fields. In this case, you could have the following:

- Event name
- Venue
- Location (city, country, state)
- Start date
- End date
- Price, if not free
- Whether it is free (this could be a check box that translates into a neat little label in the layout)

You should also consider adding better labels to your CMS that explain clearly what each piece of content will be used for and that include examples of what good content looks like for that particular section. This will help authors know what is expected without having to guess, and it aids consistency across your site's content.

Another Option: Doing Nothing

Ideally, as part of a responsive retrofitting project, you will have enough time and resources to make key changes to your content and your content strategy so that they are brought into a more responsive world. But in certain cases, you will have to consider the option of not updating the existing content for the initial release of your retrofitted responsive site.

Making updates to the content will likely involve several people who may not be at your disposal throughout the weeks and months during which you and your team work on the responsive project. For some projects, updating the content to improve the reading experience across all screen sizes and to make the content more adaptable to any platform are the prime actions that need to be accomplished with a responsive retrofit. In other cases, updating the content is not the highest priority, and other things, like improving navigation, are more pressing.

I am not trying to diminish the importance of content over things like visual design or front-end code. If you already have in place an effective content strategy and your web site is populated with considerate, useful, up-to-date content that is accessible to everyone, fixing your content may not be your top priority for this project. Starting from scratch and trying to come up with new solutions will not be the most effective use of time for a team that is already thinly spread across other projects.

What I want to accomplish in this section is just to remind you that there *is* the option of leaving content updates for later. Also remember that, even if you have no time or resources to work on updating your content in the first stages, you will still be looking at and analyzing your entire site as you work through the project. You will surely come across content that you want to improve, so make sure you flag anything that you want to do in the future that relates to improving your content.

And by all means, go ahead and fix that typo you find on your home page.

Summary

In this chapter, you have learned how you can improve your content for your responsively retrofitted web site. Depending on how much time you have, you can perform tasks that are less or more involved.

You now know the following:

- That you should focus on designing with real content instead of “Lorem Ipsum” blocks
- That accessibility starts in your content, not your design or your code
- How to perform an inventory and audit on your content
- How to approach ongoing content maintenance while you develop a new site
- Some ways in which you can quickly improve and simplify existing copy
- That you should consider creating or reworking your content strategy
- That you should have a style guide or manual in place, even if it’s small
- Small ways in which you can improve your existing content management system
- That doing nothing is also an option for sites that have good or excellent content already

The next chapter looks at how you can adapt your site’s design to better suit a new responsive reality.

CHAPTER 3



The Design Stage

It is becoming harder and harder to define exactly what it is that we are talking about when we discuss content, design, and development, because these disciplines overlap constantly and consistently in every project. And so they should.

A good content specialist has at least a basic level of understanding of design and how sites are built. A good visual designer understands how content and design are interlinked and how the designs they create will be brought to life. And a good developer knows how content and visual design affect their work, too.

Design encompasses many things; it is not just choosing fonts and colors, or even laying out elements on a screen. It influences, and is influenced by, everything from content to code.

In this chapter, I talk about visual design in its purest sense; but at certain points I also go back to talking about content and look ahead at how design decisions influence development. Neither of these parts of a project lives in a vacuum, separated from the others.

Keeping within the premise of this book, I assume that you and your team do not have the time for a complete redesign of your site or family of sites. So, your goal is to use the available time that you do have to create the maximum positive impact on how your users experience your site on any device and on any screen size.

In most cases, this constraint means the existing large-screen, fixed-width, desktop-first design will be reused if not in its entirety, at least partially. So it is this constraint that I will keep in mind when going through the steps in this chapter.

Evolution, Not Revolution

The concept of evolution is the key idea that should drive the conversion of an existing web site into responsive when you are not allowed the luxury of a lot of time and endless resources. Responsively redesigning an existing site is the opposite of starting from a clean slate. You have to deal with legacy content, design, code, and processes, and these, for the most part, are likely to have to stay the same—at least, in the initial versions of the project.

You will not have time to fix everything you think needs to be fixed, so you must be very clever about which things you leave as they are, broken and untouched; which things you reuse and recycle; and which things you can change and improve. You will have to leave perfection aside for a little while. The likelihood is great that you and your team will have to release the first iteration of your newly responsively retrofitted web site with a few (or more than a few) edges left to smooth.

And you are not alone in this respect. Consider some of the larger and most successful responsive redesigns of the last few years, and you will easily find examples where alpha and beta versions were released as quickly as possible, with many usability issues, for users to experiment with and provide feedback on (see Figure 3-1).

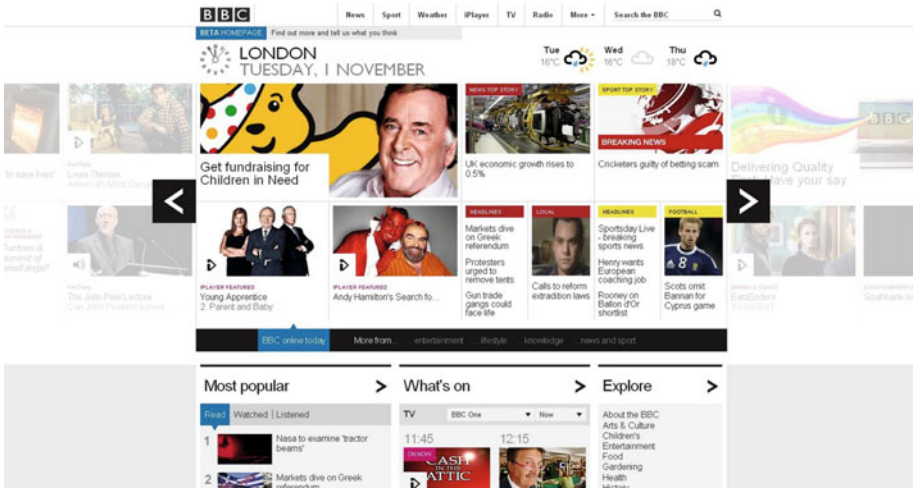


Figure 3-1. The BBC’s home page redesign was released in beta, and users had the opportunity to provide feedback

Many of these examples were successful, not *in spite* of having being released with faults for everyone to see, but *because* of that. Although this may not work for every type of site, allowing real users to try out your early prototypes is more often than not the best approach for testing. If you multiply a few users by hundreds or even thousands, you may have a lot more feedback to sort through, but you will also have the benefit of your prototypes being tested in myriads of real situations, instead of just in the testing lab.

The big redesign unveil is out of date. These days, it is common practice to reveal at least part of the design process to the world at large.

This type of roll-out strategy is perfectly in sync with the idea of evolution and iteration. You and your team should accept that your site will not be launched perfectly, whether you launch it sooner or later. You will prioritize the most important tasks that will get you the most bang for your buck—in this case, the best responsive experience for your users that you can achieve within your busy schedule. And you know that if you need to, you will keep on iterating and evolving your site based on real users’ feedback and testing.

Remember that evolution in the natural world happens gradually; you can even say it happens slowly. But the result is something that is perfectly adapted to its context and its constraints. Evolution means learning with the process, course-correcting when necessary, and keeping and refining the existing good parts for the next generations.

This is the way you should face the challenge of a responsive retrofit when you know that part or most of your existing design will have to be reused. Slowly but surely, you and your team will achieve the design goals you will set out when you envision your responsive site. But that will not mean your web site will have stopped evolving and getting even better.

Focus on Reusability

If the design of your site is going to evolve and not radically change, you need to be able to pinpoint what works well in the way you work right now, and which are the key areas to improve. When time is of the essence, it is vital that you do not spend it redoing the things that are good enough to begin with—at least, in the initial stages of the project.

By using existing solutions, you will save time to focus on solving the trickier design problems and the issues that are most aggravating to your users. You want to spend your time fixing the issues that will have the biggest positive impact once they are solved.

This means reusing not only things *you* have done before, but also things other people have done. You should understand that it is okay to copy and “steal” other people’s solutions—provided that adequate licenses are purchased and that credit is given where credit is due, of course!

One of the great aspects of working on the Web is how so many clever people share their experiences, their tools, their creations, and their process for others to inspect and even reuse for their own projects (see Figure 3-2). Make sure you take full advantage of this, without forgetting to give back whenever possible.



Figure 3-2. The HTML5 Boilerplate framework (<https://html5boilerplate.com>) was initially created by developer Paul Irish as a starting point for all of his front-end projects and is now one of the most popular frameworks available

Let me stop being vague: many things can be reused, copied, or recycled. Here are some items you can consider reusing from within your own team and organization:

- Brand
- Visual direction
- Font families
- Color palettes
- Large-screen designs
- Content
- CSS
- Style guides
- Patterns and components
- Information architecture

And here are some more things that you are likely to find in the wild, and that you can use on your own project:

- Common responsive web design patterns
- Scripts and other libraries and bits of code
- Color palettes
- Front-end frameworks
- Responsive grid generators
- Typographic scale generators

You should also remember that you can get inspiration from something that already exists, but use it in your project in different ways than the most obvious ones. For instance, you may want to carefully look into Bootstrap (<http://getbootstrap.com>), the popular front-end framework, not for its actual contents or to use it to build your site, but for the way its documentation is structured (see Figure 3-3).

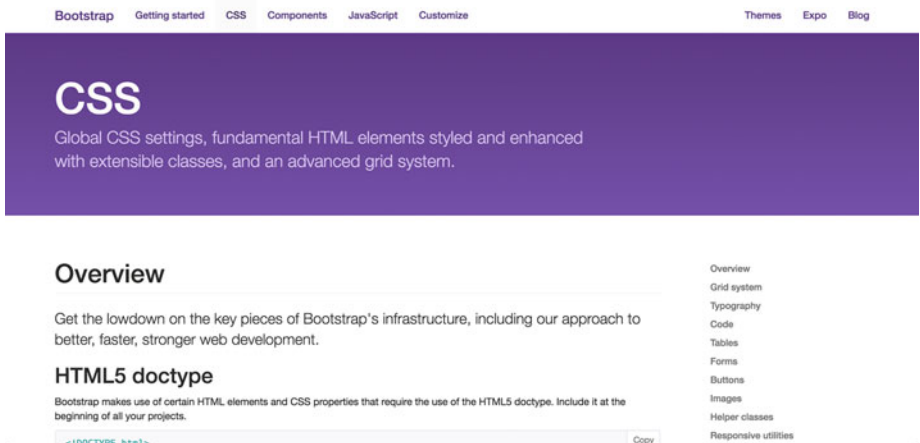


Figure 3-3. Bootstrap, the popular front-end framework

You may want to analyze existing grids of mobile operating systems and understand what you can learn and take away from them when developing your responsive grid, rather than simply analyzing the grid of responsive sites that you admire (see Figure 3-4).

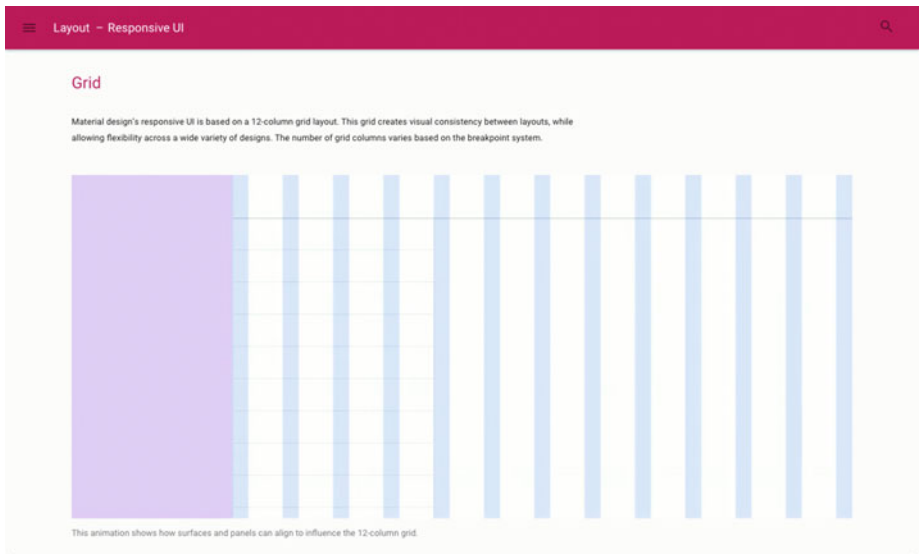


Figure 3-4. Google's Material Design spec can yield some inspiration when it comes to responsive grids¹

¹Google design guidelines, “Layout – Responsive UI,” www.google.com/design/spec/layout/responsive-ui.html#responsive-ui-patterns.

Make sure, however, that whatever it is you are using for your project is adequate. Just because someone spent time creating a framework or building a generator does not mean the product or its work is right for you.

Equally, just because something is prebuilt and somewhat generic does not mean your project's quality will be compromised if you use it—provided you select wisely. If your team is made up of clever people who are competent professionals, I am sure you will be able to reuse existing tools and patterns artfully and in such a way that the result translates into something that is your own.

Focus on Accessibility

Following on the ideas from the previous chapter, of how accessibility considerations should begin in the way you produce content, they should also permeate your process through design. If accessibility is something you tend to see as an add-on or one of the final steps in your testing process, your responsive retrofit project may be a good opportunity to start changing your ways.

As you have seen before, saying that someone has accessibility issues does not just mean they suffer from a permanent disability. Someone accessing your site on a slow hotel Wi-Fi connection has accessibility issues. Someone who has broken an arm has accessibility issues. Someone in a highly stressful situation with a piercing headache has accessibility issues.

Making life easier for people with a temporary or circumstantial disability will in turn make life easier for everyone else: for the person with a super-fast broadband connection, for the two-armed person, and for the relaxed person. Everyone benefits.²

Do not think that because you are actively making sure your site is accessible to everyone, you must create designs that are too simple or plain, and this constraint will limit your creativity. With that concern in mind, I recommend that you read Sara Hendren's "All Technology Is Assistive," which exposes how accessibility considerations for a few can spawn incredibly innovative and interesting solutions for all:

[D]isability concerns are in fact an overlooked source of rich aesthetic ideas, with relevance and impact for design far beyond their immediate starting point.³

So, what things can you do in your designs and design process to ensure that accessibility is front and center? The short answer is, loads of things—but that is not very helpful. Without getting into too much detail about such a broad and fascinating subject, here are a few simple things you can do to get started:

- Test all of your foreground and background color combinations on contrast checkers, such as Lea Verou's Contrast Ratio, shown in Figure 3-5 (<http://leaverou.github.io/contrast-ratio>).

²"Inclusive: A Microsoft Design Toolkit," www.microsoft.com/en-us/design/practice.

³Sara Hendren, "All Technology is Assistive," Backchannel, October 16, 2014, <https://backchannel.com/all-technology-is-assistive-ac9f7183c8cd>.



Figure 3-5. Lea Verou's Contrast Ratio tool

- Install a color-blindness simulator that allows you to test your designs against common and less common types of color blindness, such as Color Oracle (<http://colororacle.org/>).
- Make sure target areas for links and other user interface elements are large enough and do not require fine precision.
- Minimize the number of clicks and taps where possible.
- Be consistent with the placement of navigational elements, and keep them in the same place across your site as much as possible.
- Break up your content with headings and lists, making sure its hierarchy is well defined.
- Avoid using large images where they are not needed.
- Provide low-quality images as defaults instead of higher-pixel-density versions.
- Do not hijack scrolling.

As I am sure you know, there are many other things you can do in your web site to make it more accessible to more people, not just through its design but also in the front- and back-end development. Make it your mission to instill in your team the idea that accessibility is part of the foundation of any good web site and that the effort and time spent on improving it will pay many times over in a more positive experience for all your users.

Performance First

Discussions about performance are usually tied in with discussions about code and development. However, it is truly in the design stages of the project cycle that thinking about performance should begin.

The notion of setting a performance budget early in the project keeps growing in popularity, and for good reason. We are now designing and building web sites that are viewed on a dizzying array of devices with completely opposing capabilities and on inconsistent connection speeds. But designers tend to be working on super-fast Internet connections on the latest monitors, and more and more of them are using Retina displays. This is not a good representation of the reality of most users.

We have been hearing forever about how sites like Amazon work hard at making the site as fast as possible, to the last millisecond.⁴ These efforts are backed up by testing and, for companies as large as Amazon, are reflected heavily in their profits. When your competitor is just the click of a button away, making sure you do not leave your customers waiting and frustrated is critical.

By defining a performance budget early on, designs can adapt to this constraint. In his article “How to Make a Performance Budget,”⁵ Dan Mall goes through a simple process to define a performance budget for the less technical of us.

He starts with comparing three key indicators—Start Render, Document Complete, and Fully Loaded—for his own site and competitor sites or sites he admires, and plotting those numbers in a spreadsheet. He then sets a goal to reduce these indicators by 20%—a number that comes from research that states users perceive a task as faster or slower if there is a difference of at least 20% in speed.⁶

After some calculations (I recommend you read Dan’s article attentively to understand the math), Dan reaches a kilobyte weight target, which he can then use to make decisions such as whether he can afford to use web fonts, or whether he can do away with images and JavaScript and allow for a larger “spend” on a web font.

This is not the type of constraint that designers are used to considering. But, as Dan says, “designers do their best work within constraints.” And I cannot think of a tighter (and perhaps more fun) constraint to work under.

No More Flats

On many teams, even the ones that work in a more iterative and agile way (whatever that means), the typical design process involves several iterations of Photoshop or Sketch mockups, which are exported and shared with stakeholders. Once feedback is gathered from brand managers, art directors, product owners, the CEO, and so on, the mockups are updated, and another round of “flats” is shared. And so on and so forth, until all the design decisions are made and the ultimate mockup reflects all these considerations.

⁴Kit Eaton, “How One Second Could Cost Amazon \$1.6 Billion In Sales,” Fast Company, March 12, 2012, www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales.

⁵Dan Mall, “How to Make a Performance Budget,” December 4, 2014, <http://danielmall.com/articles/how-to-make-a-performance-budget>.

⁶Tim Kadlec, “Fast Enough,” January 14, 2014, <https://timkadlec.com/2014/01/fast-enough>.

Even if these mockups are created in various sizes to represent different types of screens and devices, nothing will ever come close in terms of representing the final product as an interactive prototype that you can test on different, real, devices—that you can stretch and squeeze to your heart’s content.

There is nothing wrong with exploring the initial concept and flow of a screen or a page in a tool like Sketch. It is important to sometimes have that time and freedom to just think about how colors, type, and space will work together to create a design that is beautiful and balanced. But do not think that this type of exploration can only be done when it is detached from code. The serendipity that classic design tools can sometimes bring to a project can also happen in a prototype. And the color, type, and space that you want to get just right render in a completely different manner when you are dealing with HTML, CSS, and a browser rendering engine.

You can try dozens of different font family and size combinations in Photoshop, but when you try the same combinations in an HTML page on a small Android device, some of the options that looked beautiful before simply will not provide for comfortable reading. Furthermore, making most of your design decisions in the browser is essential when you are trying to save time by cutting down on unnecessary steps in your process.

It is important that you and your team get your hands dirty as quickly as possible with the responsive retrofit project and start working with prototypes. Making sure flat mockups are always up to date with the latest decisions and feedback wastes time that could be more cleverly used by iterating on a prototype. And if you are really, really pressed for time, there is even a way in which you do not have to open Photoshop and can jump straight into prototyping mode.

Setting the Rules

An easy way to delve into the prototyping stage is to start by creating a document in which you list the responsive rules that should be applied to your web site. This document consists of just words—no mockups and no diagrams should be needed—and it will be used by the developers to build the first responsive prototype of your site. That’s right, it is that easy!

These initial rules will have lived mainly in the heads of the people on your team. Writing down what you think a responsive version of your site may entail is a good exercise in taking that vision out of your head and making it into something more real that you can share and have discussions around.

Write It

The best way to gather ideas for the document is to get people in a room and start writing down all of their ideas on sticky notes. In my experience, this exercise works well if the group is not too large: maybe three to five people, making sure you include at least a couple of visual and user experience designers. You may want to come into the session with a list of topics you would like to cover, so the brainstorming can happen in a more structured way and people have a better understanding of what you want to accomplish.

You will probably come up with topics that are specific to your site, but make sure you cover the following topics as well:

- Grid
- Breakpoints
- Typography
- Images
- Tables
- Forms
- Spacing
- Navigation
- Footers
- Components
- JavaScript behavior

It may also be easier if you suggest a couple of rules so that people understand what you mean by *rule*. And you should make it clear that these rules are not unchangeable. They will be used for the *first* responsive experiment, and you will iterate on them as you go along.

Once the first brain dump is finished, the next step is to sort these ideas within the topics and eliminate duplicates. If most people suggested a specific rule, it is likely that you will want to add it to your document; but you need to carve out time to discuss why some rules may be too complicated to try in this first test. Do not discard anything just because you think it will not work, though—it may well do so. The goal of the sorting step is for the group to choose which rules you will include in the first version of your document and which ones you will put in the pipeline to try later, if needed.

When you start working on the document, begin with common responsive patterns, but do not forget to be opinionated where you can. After all, you probably do not want to create a basic responsive web site, but something that reflects your brand's and your product's personality.

Some examples of how to phrase common design patterns as rules that can get you started are as follows:

- At the medium breakpoint, when there are four boxes in a row, the grid should make the row have two boxes instead.
- At the smallest breakpoint, all content should be linearized into a single column.
- The existing typographic scale should scale down proportionately by 80% (medium breakpoint) and 70% (small breakpoint).
- At the smallest breakpoint, images that are floated within text should have their floats removed and take up the full width of the viewport.

- At the smallest breakpoint, all decorative images should be removed.
- In smaller screens, the navigation should be replaced by a “hamburger” menu.
- When clicking the menu, there should be displayed a list of the first-level navigation links, and links with sublevels should display a + icon to their right, which opens the list of sublevels below the top-level link.
- Tabs should be converted into accordions, where the tab titles are stacked on top of each other. Upon clicking the title, its corresponding section is revealed. The user should be able to open all tabs at once.

If you are unsure whether you should follow one rule over another, it may be necessary to provide two (or more) alternatives so that the developers can try them both without interrupting the flow of work.

There may also be instances where you will not have any defined idea of how things should work, and summarizing a vague concept would be too hard. In these cases, it is okay to state in the document that you want this particular element type or pattern unchanged so you can resolve it and explore different solutions in other ways later. You can do this for smaller elements, such as buttons or other form elements, and for bigger, more complex elements, such as navigation or images.

Leaving some things unchanged and giving them the opportunity to just “live” for a moment in a responsive space can lead to ideas that you would otherwise not have if all you did was stare at that particular design element on your large-screen design, thinking of how to shrink it. If you are lucky, some of the elements you leave unchanged may not need any further work, at least in the first iterations of the responsive retrofit project.

Something else to consider, and that you may want to suggest, is that, while creating the prototype, the developer or developers also refactor the existing CSS to be mobile first. This is not a small task by any means, but it is a great moment in the project’s timeline, when the team is in the mood for experimentation, to have a better understanding of how much work would be involved in refactoring the existing stylesheets to be mobile first, and whether that should be a step in the process or something to forget about.

Remember: these are the very first ideas you will try when making your site work responsively. They will certainly change and evolve as you go along. Some of these initial ideas will prove good or just good enough for a start, and some you will quickly discard in favor of some other solution.

After this meeting, it is easier if only one person is in charge of writing the document and sharing it with others to confirm that everything agreed on was indeed included. The beauty of this exercise is that you can quickly experiment with some of the most common responsive web design patterns and assumptions, and test them straight away, to see whether they are the best solution for your site. Designers do not have to spend much time coming up with rules, because they do not have to create sketches or comps; and developers have some flexibility in interpreting the rules and do not have to worry about replicating mockups perfectly.

Build It

Once you are happy with your initial document, the developers (or whoever is going to build the prototype) can take some time to go through the existing stylesheets and “add some responsiveness.” I find it more effective if this is done as a group exercise, where two or three developers sit in a room together and can discuss any issues or ideas as they work through the rules. This can make for an intense and fun week that produces a tangible, useful deliverable by the end. If questions arise during development that need to be answered or brainstormed with someone else, the developers can call that person into the room and have discussions about the pressing matter. The goal is to move fast and get a prototype as quickly as possible.

If it is too cumbersome to create a prototype of your entire site following the rules from the document, choose a key section or a selection of key pages that you would like to start exploring. But do not choose the easier ones. Choose sections that seem more challenging from the start—you want to dive straight into the deep end!

The result of this process will probably (or even most likely) be a bit rough at first, but do not let that discourage you (see Figure 3-6). It is a great way to take that first step in making a responsive prototype and having something you can test on real devices.

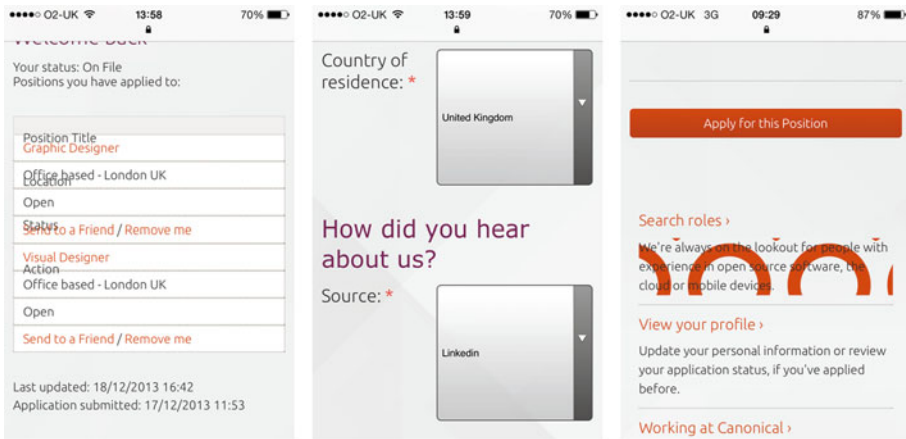


Figure 3-6. Canonical.com’s first responsive test: not everything looks perfect on a first-stage prototype

You can use whichever format works best for your team to create your document: Google Docs, a Word file in Dropbox, a Basecamp text document, and so on. Just make sure whatever you use can be easily shared, edited, and commented on.

Determining Breakpoints

Breakpoints are at the heart of responsive web design: they define the widths at which your layout will adapt to better fit and make use of the space available in the viewport. There are a few different approaches to determining breakpoints on a responsive site. The consensus these days seems to be geared toward a more content-based approach, where the breakpoint is determined at the size where content starts to break, where it becomes harder to read, and where certain components do not have enough space to completely fit and need to be adjusted or moved.

Another popular approach (and possibly more prevalent) is to choose a few breakpoints, determined by the most common devices either on the market or used by the site's visitors. At these breakpoints, the layout and order of content may change, and there may be significant differences in the way the grid works.

Both approaches are valid, but the latter one may mean the chosen breakpoints will soon become outdated by technology. When you link your designs with whatever devices are available now, once a new type of device comes along, you will have to revisit your decisions.

Decide in the Browser

In your responsive project, you are probably working from a large-screen design, which you will leave mainly untouched, down to smaller sizes. This is the case for most responsive retrofitting projects that do not have scope for a complete redesign. In this case, the exercise I outlined in the previous section, of creating a fast responsive prototype based on a document with simple responsive rules, can be very helpful for making decisions about how and where to rearrange content and reflow design patterns. You can see where your content starts to break by looking at it on real devices, and you can make decisions based on that tangible experience.

Another interesting exercise that you can do quickly in-browser is to remove all floating, widths, and positioning rules from your CSS and see how your content behaves when it is stacked linearly. Doing this, and then resizing the browser window, will very quickly give you a good idea of how many breakpoints your content may need, and you will see how your content responds to very wide and very narrow screen sizes.

Tweakpoints and Breakpoints

A technique that I particularly like, and that is also content-based, is when you have a few breakpoints at which there are dramatic changes to the design and layout, but there are also breakpoints at which there are only a few adjustments to some components and style. Jeremy Keith calls these *tweakpoints*.⁷ This approach respects the content and the components of your site; it only changes them when they need to be changed, not just because everything else has changed due to a big breakpoint.

⁷Jeremy Keith, "Tweakpoints," February 15, 2013, <https://adactio.com/journal/6044>.

What to Keep in Mind

Regardless of which approach you decide will work best in your case, there are certain elements you should analyze carefully when you are determining where to add a breakpoint or tweakpoint. Mark Boulton recommends that you direct your attention to a few key elements that may need adjusting when you are studying the way in which your content adapts to a resized viewport:⁸

- Type size and leading
- Micro and macro whitespace, such as margins and paddings
- Vertical space
- Flow
- Number of words
- Source order

When to Use Analytics

Looking at your analytics will not answer the question of the points at which your content breaks when resized. For that, you need to look at your site. But getting an idea of the sheer diversity of screen sizes on which users visit your site is certainly something you should do.

By looking at your analytics data to check what common screen sizes your users are on, you will see that the gulf between the larger and smaller screen sizes can be immense. You will also see that there is not one true mobile, tablet, or desktop size: the variety is never-ending, with countless tiny differences between them. This will open your eyes to the idea that your web site should truly be fluid and adaptable to any size of device.

Defining a Style Guide

Creating a style guide is not an easy or a quick step, but the benefits that it will bring to your responsive project will be collected in heaps. I really believe this is not a step that you can skip if you are serious about making the move to responsive and you do not already have a style guide in place.

Your style guide can be more or less detailed to begin with, but it will create a framework for the work you will be doing, making it easier for people within and outside your team to communicate about design. Figures 3-7 and 3-8 show style guides with different levels of complexity, though both do the job well. A style guide will help you and your team focus on reusability—an indispensable component of making your site responsive when time and resources are scarce.

⁸Mark Boulton, “The In-Between,” February 14, 2013, www.markboulton.co.uk/journal/theinbetween.

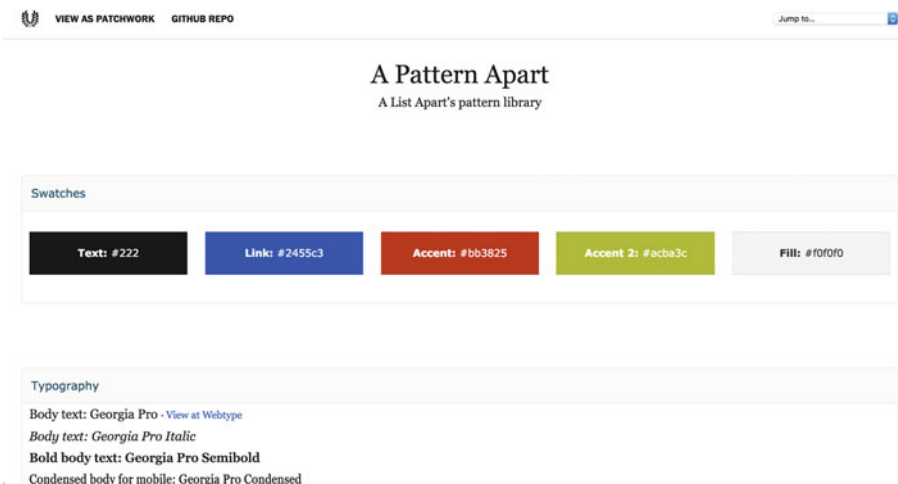


Figure 3-7. A List Apart's pattern library fits on a single page (<http://patterns.alistapart.com/>)

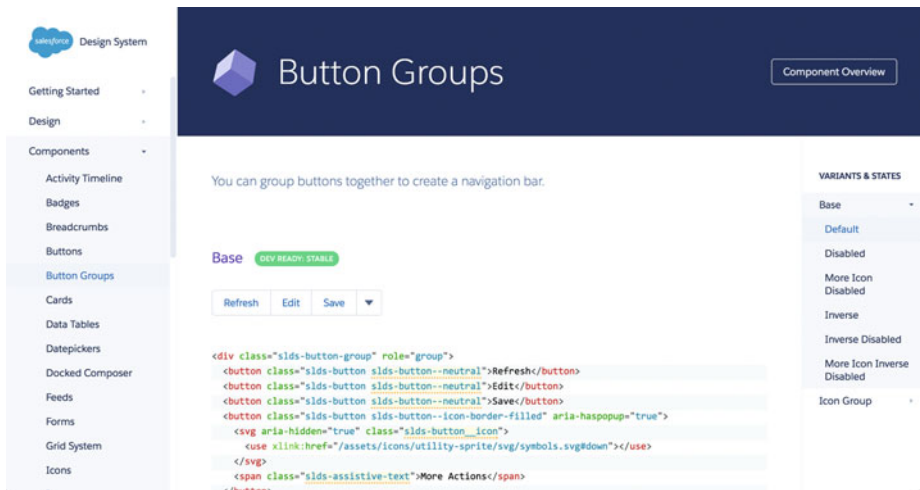


Figure 3-8. Salesforce's Lightning Design System has several layers of navigation and headings (www.lightningdesignsystem.com/)

A style guide will save you and your team time because it creates a consistent style that is easier to manage and to understand. When something needs to be designed, you can refer to your style guide to see what rules you have already defined for that particular type of element. When someone asks for a variation of your existing patterns, you have a document to back you up when you ask them about the reason for the deviation. The point of defining a style guide is not to redesign your entire site in the process, but to examine your existing design, extrapolate design patterns that are used consistently, and define how those elements should be used across your site.

You may prefer to run the creation of your style guide as a separate project from your responsive redesign. If you do so, make sure you complete the first version of your style guide before you delve into your responsive retrofit, because the style guide will serve as a solid foundation from which you will make important design decisions. As you make your site responsive, your style guide will be updated accordingly.

Although the creation of a style guide from the ground up does not have to be a task that takes up a large amount of time and resources, it is important that you and your team are committed to creating it and that key people in the organization are on board and supportive of your efforts. Just as you need a person who will champion the responsive retrofit project as a whole, you also need someone on your team who will drive and evangelize the creation of a style guide. Even when the entire team is in agreement that making a style guide is important, having someone whose responsibility it is to see the project through is imperative.

As with many other topics in this book, the subject of style guides is broad enough to call for an entire series of books dedicated to it. I will provide you with the steps you can take within your responsive retrofit project that will help you get started with your first style guide.

Screenshots, Screenshots, and More Screenshots

The first step you need to take when creating your style guide is to make an inventory of all your existing design patterns and components. The easiest way to do this is to go through your existing site, take screenshots of all the components, and save them into different folders (see Figure 3-9).



Figure 3-9. Component inventory folders

Ideally, the screenshots would be taken by a few different people. But if it is not easy to find the time to do so, you can assign the task to one or two people.

You will probably end up with folders for the following:

- Colors
- Typography
- Quotes
- Lists
- Tables
- Forms
- Links
- Buttons
- Icons
- Navigation
- Footer
- Tabs
- Tooltips
- Code
- Search
- Images
- Boxes
- ... and so on

Organizing the screenshots this way, where you can quickly look at them together, makes it easy to see variations that should be rationalized, similar patterns that should be merged into a single one, and patterns that need to be removed.

Take screenshots of small elements (like a single button or an input field) but also of large elements (like the header of your site or a form fieldset with all of its components). Do not take screenshots of every instance of the same component, but rather of every variation of a component. If you spot a button that looks exactly the same on different pages, take only one screenshot of it. If you spot another button that varies by 1 or 2 pixels in border radius, take a screenshot of it. If you spot another button that has slightly larger padding than the first one, take a screenshot of it. I am sure you get the drift by now.

You will be surprised to encounter slight deviations from an initial design pattern that you thought was consistent across your site. And this may happen frequently, so be prepared to spend a fair amount of time with your team defining rules for how the patterns should work.

Responsively Rationalize

The next step in the process is to, together with your team, go through the inventory with a fine-tooth comb. Whether you had to take the screenshots yourself or assigned another team member to do it, this step should be done as a group. You need to have a discussion about each of the elements you have identified and its variations, to decide which patterns can be merged with others, which patterns are redundant and not needed, and which variations should be made into a separate pattern (see Figure 3-10).



Figure 3-10. A component inventory uncovered several variations on the same pattern on *ubuntu.com*

As important as visual consistency is, you also want to achieve naming consistency. Keep your ears open, and write down the names that different people give to different components. Do most people call the navigation “main nav,” or is it “header nav” or something else? Make sure at the end of this process that you have agreed on every name in your style guide. Several cups of coffee or tea may be needed!

There are a few different schools of thought when it comes to organizing a style guide. Do you call patterns atoms, molecules, and organisms, according to Brad Frost’s Atomic Design?⁹ Or style, layout, and components, like Google’s Material Design?¹⁰ As usual, you must make the decision based on what works for your team.

As part of these discussions, you should also consider how each element will behave in a responsive world. If you are creating a style guide for a fixed-width site (and if you are reading this book, you are likely to be), this is an excellent opportunity to think about

⁹Brad Frost, “Atomic Design,” June 10, 2013, <http://bradfrost.com/blog/post/atomic-web-design>.

¹⁰Google design guidelines, “Material design,” www.google.com/design/spec/material-design/introduction.html.

how each individual part of your site will stretch and squeeze depending on the size of the viewport. By decoupling your design patterns from the pages where they sit, you can start thinking about them as repeatable objects in a system of templates and pages. You will focus on their reusability across the entire system, rather than what they look like on a particular page or screen. As usual, remember that all the decisions made during these discussions should be well documented, shareable, and searchable.

Build Your Style Guide

I will not go into too much detail about how to build your style guide so I do not run the risk of taking up the rest of the book on this subject. There are several tools you can use to create your own document; one of the most popular is Pattern Lab (<http://patternlab.io/>), by Brad Frost and Dave Olsen (see Figure 3-11).

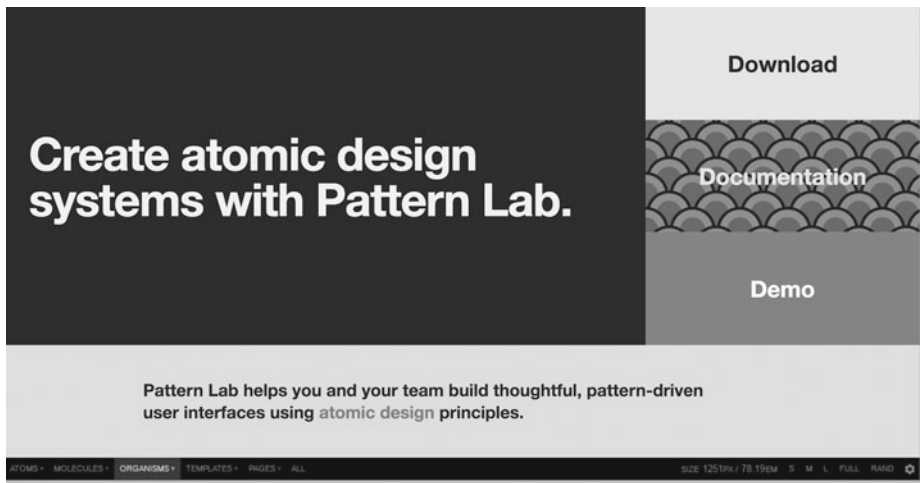


Figure 3-11. Pattern Lab is Brad Frost’s and Dave Olsen’s tool to create atomic design systems

In all honesty, a simple one-page HTML file that lists all of your patterns and their names, and that is linked to your latest CSS, is enough to start with (refer to the A List Apart example in Figure 3-8). This is what I recommend when you do not have much time.

Think about what information should be contained in your style guide and the level of detail you need to make sure your document is useful for its target audience. You will probably want to include some or all of the following elements:

- Name of the pattern
- Instance of the pattern, not just an image of it
- Description of how and where it should be used

- Description of how and where it should not be used
- Other patterns that can be used alongside it
- HTML and CSS code needed to use it
- Link to the location in your CSS files

Make sure you talk to a variety of people who will use the style guide before you decide what information to include. Some may prefer to just be able to grab some code, whereas others may find it more useful to be able to read about correct usage.

Clean Up Your Style Sheets

After creating a style guide, and as part of the same process, you should make sure your CSS reflects the same thinking and follows the same rules you and your team have determined. Otherwise, the style guide will have been made in vain—a static PDF file that shows all your design patterns will not suffice.

Duplicated components should be removed, but you should also remove redundancy from your CSS files, making them more manageable and easy to understand. This step may sound like a no-brainer, but it is easy to forget after you have spent so much time rationalizing all your design patterns.

When You Already Have a Style Guide

If you have an up-to-date, non-responsive style guide, then a lot of the work is already done for you. You will still have to examine your style guide and consider how your patterns may change in a responsive world. But if your style guide is not up to date, then you need to go back to the first step and make an inventory of your existing patterns, which you can then compare with your out-of-date style guide.

Style Guide or Pattern Library?

Running the risk of arguing semantics, it is worth touching on the subject of whether your site needs a style guide or a pattern library. Different people have a different understanding of what each of these terms means, and many people consider them one and the same thing. So it is good to clarify this distinction on your team and align expectations.

A definition that I particularly like is the one that Jeremy Keith offered on the *Style Guides* podcast:

[A] pattern library (...) can be part of a style guide, and a style guide could contain other things. It could contain tone of voice; it could contain here are the colors, here are the font choices, here are [brand style guides].¹¹

¹¹Jeremy Keith, *Style Guides* podcast, Episode 41, <http://styleguides.io/podcast/jeremy-keith>.

You can follow Jeremy's interpretation; choose to use them interchangeably, as I have done throughout this chapter and this book; or propose an entirely different definition. The key factor is that everyone on your team has the same expectation of what the style guide will do, what it will contain, and how it will fit within other types of documentation you may produce or work with, such as brand style guides or tone of voice guides.

Standardize across Sites

It may be the case that your team oversees not just one site, but an entire family of sites that sit under a common umbrella, partially or fully sharing the parent's brand and visual direction. In this situation, the definition of a style guide or pattern library is paramount for making sure the brand is applied consistently across the different properties, especially when considering a move to responsive.

This is where the standardization process that you go through when defining a style guide will really pay off. Each element and component that you define not only will save time for one team, but also is likely to save time for several teams. A shared style guide will allow each team to consult a single source of truth for your several components. They will not have to try to find who to speak to, to know whether a design for a certain pattern exists; they will not create duplicated work for a pattern that is already available and usable; and they will not ask teams further up the organization for resources to create designs that are already in place.

Flexibility and simplicity are key for this to work well. When you create a style guide that will be shared across different web sites, you need to think further in terms of how prescriptive your design patterns should be, and which options you should give designers and developers when applying the components on their own site. Should you provide less complex patterns that they can build on based on their particular needs? Or is it easier to provide a larger assortment of more detailed components from which they can pick and choose?

These are some of the considerations you should discuss when going through your pattern inventory and making decisions about which patterns to keep and which to adapt. It is worth noting that the inventory itself will be bigger because it must include as many sites as you think necessary, so you can compare the different usage of the same pattern.

Remember that standardization does not translate into sameness. There is always a place for new ideas and for thinking outside the box, even when you are working with a solid style guide. Think about how chaotic and difficult to manage it would be if every single element of every single page of your site was designed differently. (I am certain there are use cases for this scenario, but that is not applicable to most sites.)

You should explore this idea more deeply than I can cover in this book, including how to technically implement a style guide that trickles down onto different sites that may have different needs. The solution will depend on your current development stack and your processes, and also on who owns each different site and how easily you can reach those owners.

Quick-and-Dirty UX

I think I have established by now that this book's main focus is on using your team's time cleverly while you convert your fixed-width site to responsive. But what about research and user experience? How can you get things right with so little time on hand?

We all know that to truly understand the goals of a project and, most important, the goals of your users, research is not something you can skip. If you want to improve the experience of your users, you need to be able to put yourself in their shoes and know what they want to get from your site, and how.

I assume that the main goal (or one of them) of your responsive retrofit project is to provide your users with a great experience when they visit your site on any device, be it a massive iMac or a tiny Sony Android. You want them to be able to easily read your content, navigate your pages, and perform any tasks they want to or need to, whether they are at their desk or on the train.

On the other hand, you already have a site, which your team probably put a lot of effort into creating and improving, even though it is not yet responsive. You probably have at least some knowledge about your users and some data on which you can base decisions.

You will have to improvise and understand that it is not the end of the world when you do not have the time and money to follow a rigorous UX process. Sometimes you have to go with something faster and cheaper, making sure you use cleverly all the skills and expertise of the members of your team. There are a few things you can do to make smart decisions—using the information you know and your team's experience—that will keep moving your project forward.

Sticky-Note-Sized Wireframes

It can be a bit daunting to have to visualize an existing fixed-width, wide design in a small viewport when you have not done so before, or when the content is so complex that you do not know where to start. I find that if you make little wireframes or quick sketches on sticky notes that represent one small part of the page, such as a single row, it makes it easier to begin exploring small-screen solutions for that particular element.

Not everyone is a fan of printing out web pages, but a simple exercise that you can do to start generating ideas of how your pages may reflow on smaller screens is to print a few of them and quickly sketch on sticky notes how each part of the page could be constructed in a small viewport. You can stick the notes alongside your printout to make a tool that can kick-start interesting conversations around your design and content flow within your team and with stakeholders.

Super-Speedy Prototyping

Let's clarify one thing: prototyping does not necessarily involve code. And it probably *should not* involve code in its first stages. The first thing that should pop into your mind when thinking about prototyping, particularly when you are moving at the speed of light, is paper (see Figure 3-12).

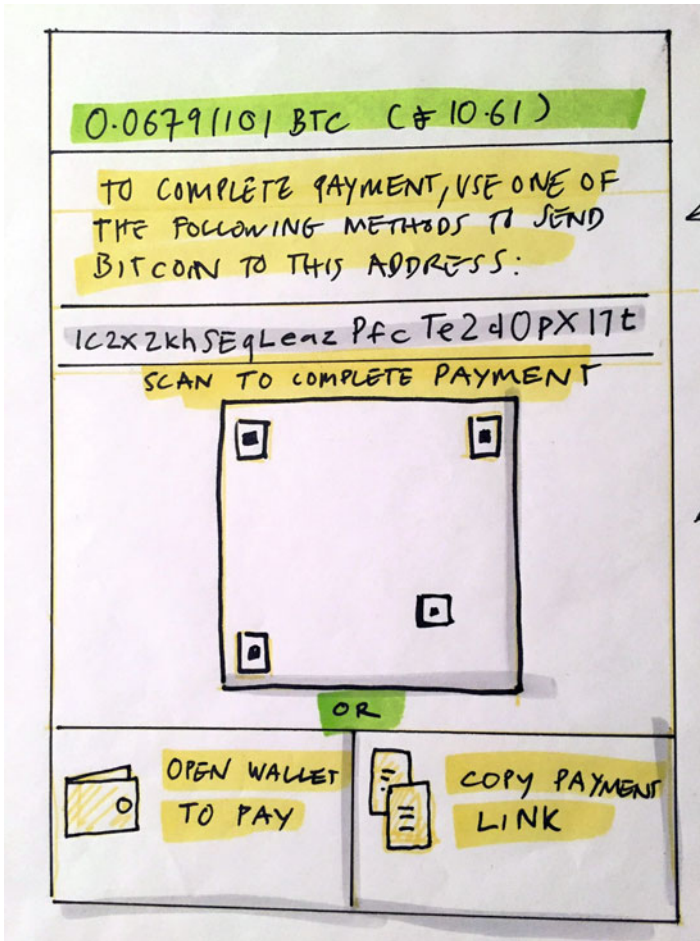


Figure 3-12. A paper prototype that can be tested with users

Despite the immediacy of paper, getting to a point where you have a prototype that you are happy to share and test may take time. But it does not have to be that way. Let me show you an example.

Let's say you want to design a new navigation for your site that works well across different screen sizes. A good way to kick things off is to gather a few members of your team from varied backgrounds for a brainstorming and sketching session. Do not only invite designers. You can mix things up by having visual, UX, and interaction designers, developers, and copywriters, for example.

You should set aside at least a couple of hours for this session, perhaps even an entire morning or afternoon, with a break or two in the middle. Start the session by looking at other sites' navigation patterns and discussing how those examples could potentially work, or not, in your case. This is an easy way to get the creative juices flowing and break

that initial awkwardness that is so common at the start of many workshop-type sessions. You can ask people to think of one or two examples beforehand, or you can propose the examples yourself at the session.

After this, grab big sheets of paper and markers and sketch the ideas that have come up during the initial discussion. Use as many sheets as you need, and draw as many ideas as you can in the allotted time—you do not want to discard any ideas at this stage; you will get to that soon enough.

As you draw and discuss the sketches, you and your colleagues will find issues with some of the ideas, and you will begin to sift through the options. At the end of this session, you may end up with a couple or a handful of ideas that you think might work for your site.

The next step, building a prototype, will likely be completed only once the session is over. You must decide, based on how much development time you think you can afford, whether you will build all the ideas you think may work, or if you need to whittle away some options and only test one or two alternatives.

Once the first iteration of your prototype is built, you should reassemble and test the prototype on a few devices. Having first-hand experience with how it feels to use your design is paramount for being able to create a great mobile experience. You will know where the prototype fails and where it succeeds, and you can gather feedback from the other team members and iterate based on that until you reach a solution you are happy with.

Express Testing

If you have time, you can and should test the prototype with people who are not directly involved with the responsive project. You can test either the original paper version or, if you prefer, a digital version of it, which can be reproduced in a tool such as Axure (www.axure.com/) or InVision (www.invisionapp.com/). Even informal testing can help you to quickly test a new design pattern or a certain device and to validate assumptions.

Testing does not have to take a lot of time and cost a lot of money. Finding participants and hiring testing labs can be an expensive endeavor in itself (albeit worth it), let alone the time it can take to analyze the results and produce digestible findings. The solution is *guerrilla testing*.

If you have not heard of this concept before, guerrilla testing is all about getting feedback from users quickly and cheaply. To run a guerrilla testing expedition, take the following steps:

1. Choose a location where your target audience may hang out (choose somewhere they would have 10 minutes to spare, not somewhere they would be in a hurry).
2. Set a goal for the test. What do you want to know? What questions do you want answered? Be specific. Knowing whether people love the design is not very useful, but you may want to know whether people can find more content to read after they have read one of your article pages, or if they get stuck.

3. Define what tasks the users can perform that would answer the questions. You do not want them browsing aimlessly until they find what you want them to find. Here is an example: “A friend just posted this link on Twitter. You click the link and look at the article. You like the site and want to read more. Where would you go from here? Do you know what this site is about?”
4. Gather all the necessary gear, and head outside. Ideally, you would be a team of at least two or three, making sure there is one designer and one developer present. Someone should be conducting the interviews, and another person should be taking notes, or recording the sessions if the participants have given written permission to do so.
5. Select between five and ten people with whom to test your prototype, and go through the testing protocol.
6. Ask people to think out loud as they try to complete the tasks. Some users will keep quiet unless you keep prodding them to think out loud. Do not just ask them what they think about what you are showing them; be specific. Where would you click now? What do you expect to find once you click there? Did it match your expectations?
7. Avoid asking leading questions, and do not explain too much of what they are looking at.
8. If testing on paper prototypes, keep non-relevant screens out of sight from the user, so as not to distract them, and only reveal the screens when appropriate.

You can also test around the office or with people you know, if you are looking for answers that do not require a very specific type of user. But be aware that these people may be tempted to be positive in their comments because they know you and/or are close to the web site.

Once the testing is complete, remember to catalogue your findings, even if you did not do formal testing. It is a good idea to summarize the key findings in a digestible format, highlighting the most important actions that should follow and things that can wait. After you have improved your prototype, go do some more testing!

Key Things to Test

If you feel overwhelmed by the idea of doing usability testing with users, because you want to find out whether many parts of your site are working well, start with the following topics. They greatly influence the experience of the entire site:

- Can they navigate your site? Are the labels in the navigation self-explanatory?
- Can they find further content once they have read a certain article or page?

- Can they find help easily?
- Is the search functionality helpful? Are the results produced useful?
- Does the product information page include all the details they want to see?
- Is the checkout process easy to go through? What are the main obstacles when trying to pay?
- For all of these examples, how well can they be performed on small and medium-sized screens?

Grids and Type

Web sites are mostly words aligned within grids, inside rows and boxes. This is certainly a simplistic way of describing a site's design; but based on these very simple elements, we create a multitude of different designs that are starkly different from each other, each with a different personality and look. That is why spending time considering your type and grid will provide a sound foundation for your refactored responsive site.

The reality of an existing large-screen design that you have to work with will not make it easy to create a solution that diverges much from the common responsive solutions. But you can still create something that works well for your design and for your content, and that works beautifully across all screen sizes.

Convert Your Grid to Percentages

If you are feeling a bit overwhelmed by the amount of work ahead of you, a quick exercise you can do is to convert all the units used in your existing CSS grid from absolute units (such as pixels) to percentage units, while keeping the site's main container (or containers) using absolute units. You do not even need to touch your HTML in this case.

This exercise will give you the ability to—by removing the fixed-width container—test how well your existing grid and its underlying CSS will react in a responsive world. You can test a very rough fluid prototype on small-screen devices and determine how much work is required to come up with a robust, flexible, responsive grid.

This does not have to be a time-consuming exercise, especially if the product of the experiment is not going to be made live for users. However, if you do choose to make it live, remember that your users will not notice the difference anyway, because your site will still be in a fixed-width container. The changes will all have been made beneath the surface.

As with many other steps in the process of converting your site to being responsive, this will be a collaboration between developers and designers, and a step that not everyone on the team needs to be involved in at the same time. One developer can make the CSS changes for the rest of the team to test later.

Explore Responsive Grids

Understanding your site's underlying grid and how it will change and behave in a responsive world is an important part of the foundation of a good responsive design. Things do not have to be complicated. Some simple rules may help you to visualize how your grid works and how it could behave depending on the size of the viewport.

If you have followed my suggestion of creating a document with some initial responsive rules to build your first prototype, you have already written down some basic principles for how your grid could scale up and down. Let's start with a very simple example:

A 12-column grid in large screens, scales down to 6 columns at its medium breakpoint, and then down to 3 columns at its smaller breakpoint.

In this case, the 12 columns will translate into a combination of different-sized blocks that contain different components (see Figure 3-13). These components can be text, images, or more complex elements such as forms, widgets, and so on. What is important is that you abstract that detail and think about the overall layout that is being produced by the grid.

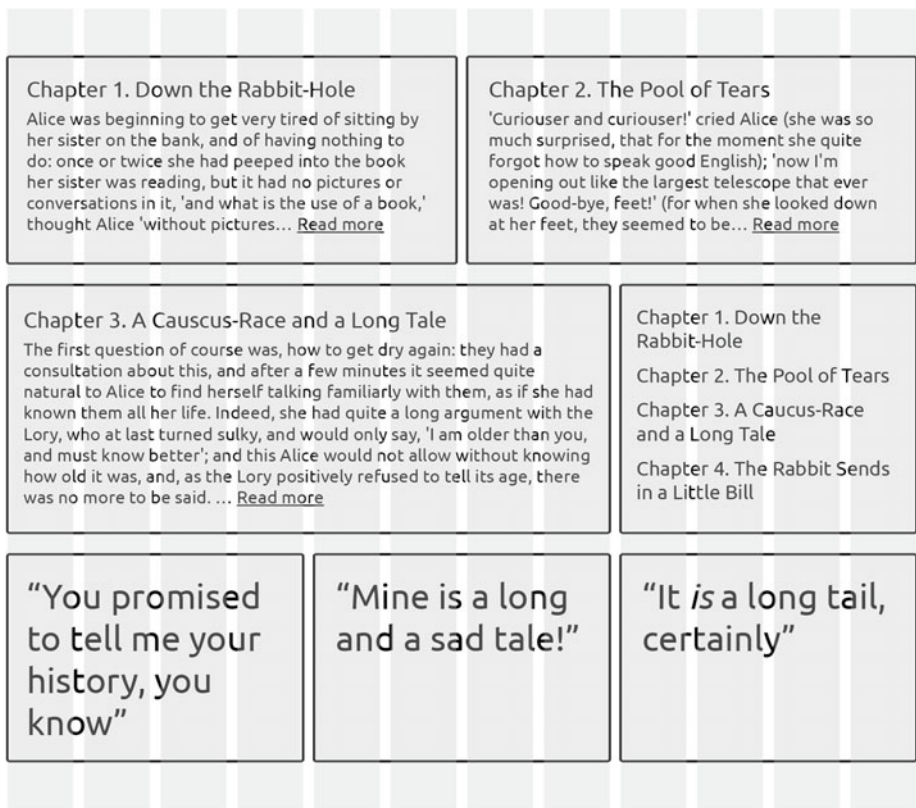


Figure 3-13. Content boxes in a 12-column grid layout

Now, if you consider how the same content would fit on a screen size one-third smaller, you will notice that not everything can stay the same as before (see Figure 3-14). You must make decisions about what happens when parts of the content become harder to read because they are squished too narrowly.

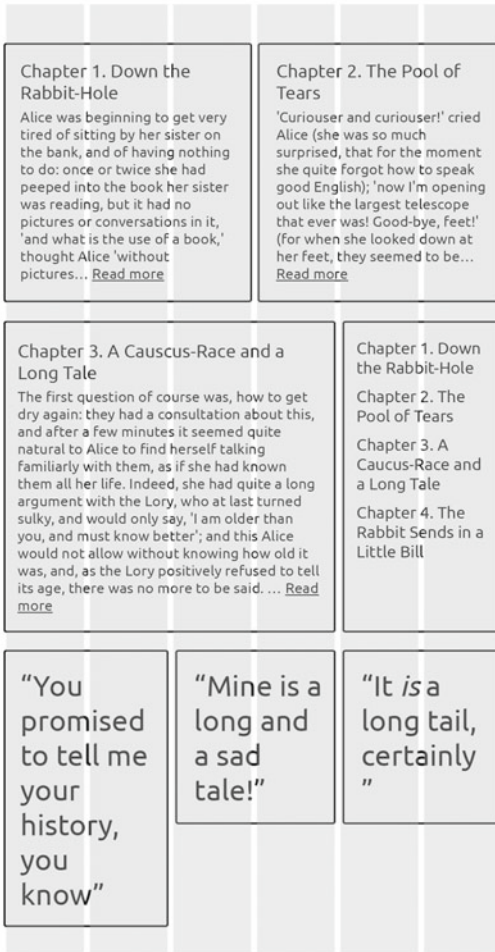


Figure 3-14. The same content boxes squeezed in a screen one-third smaller than before

In this case, the first row can still be divided into two boxes that take up 50% of the width. But if the contents on the second row keep their proportions, the smaller box becomes too narrow. So maybe you need a rule that says at this breakpoint, in rows that contain two boxes, those boxes should all take up 50% of the width, even if they didn't initially.

And what about the third row, with three boxes? It is really up to you. You could continue from the previous rule and state that in this case, the boxes in the third box should fall down to their own row (see Figure 3-15).

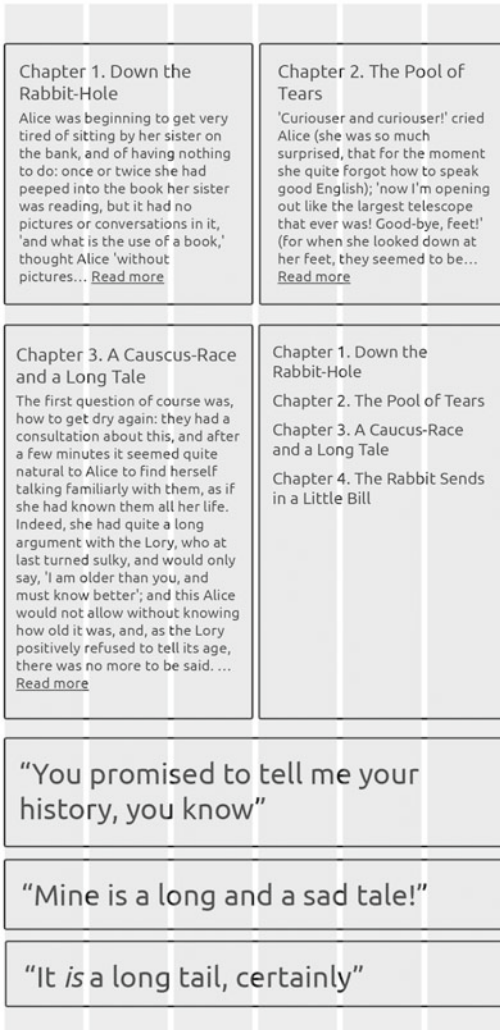


Figure 3-15. Small adjustments to the narrower viewport that adapt to the content

This is a simple exercise that will get you started on a conversation about your grid and how it affects your layouts. By the end of it, you should have simple sketches and notes about how your grid should behave in a fluid, responsive context (see Figure 3-16).

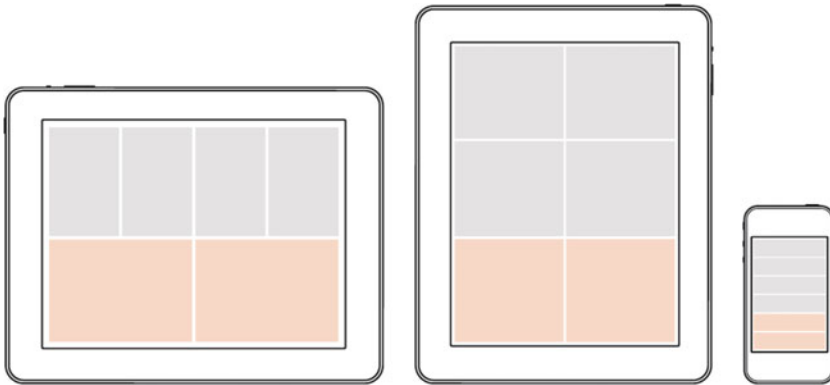


Figure 3-16. *A simple responsive layout*

Reorder Content

The order in which elements have been laid out in your HTML can dictate how it will reflow on small screens. Depending on your existing designs, it may well be the case that less relevant content comes first in the markup. In many retrofitting projects, touching the HTML is not an option, but there are a few tricks you can put to good use to make sure the most important content is seen first.

One very simple way to lessen the importance of a block of content is to show it only partially. You do not want to completely remove parts of your content from small screens. In this scenario, you make it so that only the title of that section is visible; to see the full content, the user has to click or tap the title.

Wikipedia applies this trick on its mobile site for all the sections of an entry except the introduction (see Figure 3-17). Rather than having to scroll through what can be extremely lengthy pages, the user can have a quick glimpse of all the headings and select only the one they are interested in.

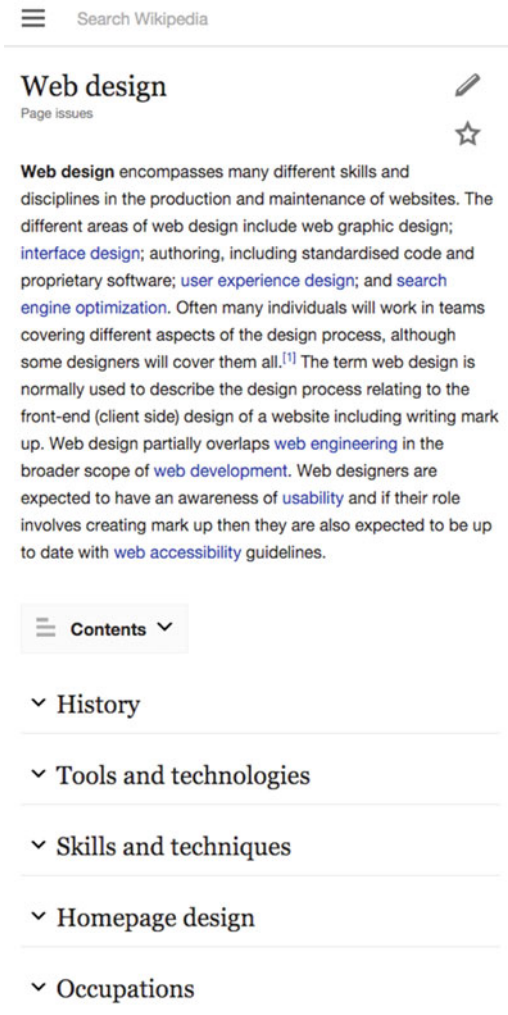


Figure 3-17. Mobile view of Wikipedia, with only the introduction and the section headings visible

If you have related content, advertising, and other things that are not part of your main content area and that exist on some type of left sidebar, chances are they come first in your HTML. If you squash these into headings, you will save the user some scrolling to get to the meaty bits.

A more complex solution for unordered content is to apply an off-canvas solution. Here you effectively hide the part of the content that you do not want to show front and center. This is typically applied to navigation, but it can also work on sidebars. A link or button on the edge of the viewport, when clicked, opens what usually behaves like a digital drawer, revealing more content or links (see Figure 3-18).

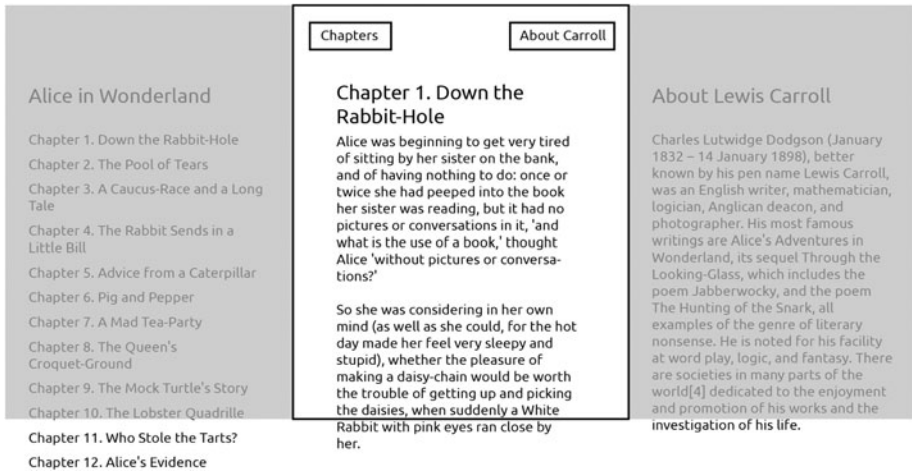


Figure 3-18. In an off-canvas solution, less important content can be hidden to the sides of the main content area and called by a link near the edge of the screen sides. In this example, clicking *Chapters* or *About Carroll* reveals the content to the left or right, respectively, of the chapter

I will not go into detail about how this can be implemented, because it can be complex. But it is a pattern to consider when restructuring and reorganizing your content for smaller screen sizes.

Finally, you can also consider using CSS flexbox to reorganize your content based on media queries. The power of flexbox lies in the ability it provides designers and developers to lay out content in any order or direction, regardless of its order in the markup. We will look at how to use flexbox in your responsive layouts in Chapter 4, in the section “Improve Your Markup.”

Adjust a Strict Typographic Scale

When moving from a large-screen, fixed-width site design to small screens, it is unlikely that you will be able to maintain the same font sizes. When defining font sizes across different breakpoints, you may start by determining the ideal sizes based on a precise typographic scale. You may even have already applied a carefully crafted scale to your existing site.

If this is the case, you can start with what you have and iterate from there. If not, there are several tools you can use to generate the most appropriate typographic scale for your needs, based on the size of key elements in your designs such as an important widget, the size of your featured images, or the width of your main content area, to name only a few examples.

These typographic scales usually produce font sizes that provide the perfect visual balance and ease of reading when used side by side. But you should not feel that you have to stay married to a strict typographic scale.

If you have a defined large-screen typographic scale that you plan to maintain, you can, for a start, reduce all the font sizes proportionately as the screen size reduces. Once you have done so, and you have a working prototype of the responsive scale, you should test the prototype on real devices to get a better perception of how it feels to read your content. You will easily spot where the text becomes too small or too large to be read comfortably, and you can improve the prototype until you have a typographic scale that works well across different screen sizes.

It is very different to read a piece of text in a narrow window on your large screen (or even an emulator) and to read it on an actual small-screen device, like a smartphone. So, make sure you test the prototypes on the real deal.

This process may sound simplistic, but it involves a lot of trial and error and testing across a range of devices; do not be fooled by how simple it seems at first. The readability of the words on your site should be one of the main goals of your responsive retrofitting project, and there is truly nothing like testing and iterating on real devices.

Handling Your Images

Most responsive retrofitting projects have to handle responsive images to some extent. One of the big differences when making an existing site—with lots of existing content, and therefore many existing images—responsive is that you may not be able to go through every single image and reformat or resize them to adapt perfectly to a new responsive state of affairs.

So what can you do when time is sparse, to make a difference when making your site responsive? What should be at the top of your priority list, and what things can you leave until later?

Make an Image Inventory

Before making any kind of image-related decisions regarding your responsive project, you must understand what kinds of images are being used across your site. Without this knowledge, you will not be able to make the best decisions for your site.

To find out what types of images are used on your site, you need to conduct an image inventory. Just as in a content inventory, this involves a thorough investigation of your site, with a focus on images and, potentially, other types of media such as video and audio.

This may sound like a daunting prospect. If you do not think you can accommodate a full image inventory in your schedule, you can consider analyzing only a representative section of your site. Do this if you think that it is more appropriate for your case and that you can confidently cover all the different ways images are used across your site.

The most common image types that you may find on your site are as follows:

- User interface icons
- Logos
- Illustrations
- Diagrams
- Infographics
- Stock photography
- Editorial photography
- Product photography
- Portraits
- Backgrounds
- Avatars
- Purely decorative images

For each type, you want to get a better understanding of

- The image's importance
- Where the image will be placed
- How the image plays with the rest of the content
- In which contexts the image is being used
- Who produces the images
- Who is uploading the image to the site

Just as you did for your content, create a spreadsheet in which you can catalogue all the information you find about your images. You should highlight the images that may be trickier to deal with, such as complex diagrams and infographics, by assigning to each image the difficulty of handling it in a responsive site (for instance, from 1 to 3, where 1 is easy and 3 is difficult).

You can use the following list as a starting point to lay out your own spreadsheet:

- *Name*: Name of the image, include a thumbnail if possible for easy recognition.
- *Location*: Where the image sits in your site's IA.
- *URL*: A link to where the image lives.

- *Type*: Illustration, UI icon, photograph, and so on.
- *Format*: The file format: .png, .svg, .jpg, .gif, and so on.
- *Author*: Who created the image?
- *Owner*: Who commissioned and is in charge of replacing or updating the image?
- *Difficulty*: Is it a 1, 2, or 3? (Or use whatever other scale you have decided on.)
- *Notes*: Anything else worth noting.

Only after you gather this information can you compare solutions effectively and possibly arrive at a few different solutions for various types of images.

Many of the ways in which images can be handled in a responsive retrofit have to do with the way the site is coded and built. I go into more detail about these in Chapter 4. But there are a couple of things you can do in your design and design process that will greatly impact how difficult this task is later.

Mind Those Bytes

It is easy to forget about the impact your designs will have on how fast and lean your sites are when visited by users. As I have mentioned before in this chapter, considerations of performance should not only be part of a developer's job description. By creating designs that are respectful of the user's time and money (many times, more bytes mean slower loading and more expensive data charges), you will save your team a lot of headaches later on in the process.

Before splashing enormous, transparent, Retina-ready PNGs all over the page you are designing, think about whether that will truly improve the user's experience. Do all the images need transparency? Do they all have to be full width? Can you do with only one larger image at the top and then smaller ones throughout the page? Can you design another solution that does not involve images at all?

Do not assume that users will take in stride whatever you throw at them. If the big images you put on your site do not provide any value, they may consider going somewhere else.

This tip may not necessarily be realistic to apply to your existing images. You probably do not have the resources to reconsider every image across your site and think about whether it can be removed. But it is something to keep in mind as you work through future iterations of your design, as you add new pages and new sections, and as you make further improvements to your responsive designs and processes.

Consider SVG

SVG stands for Scalable Vector Graphics. The name says it all. SVG images work like Illustrator or Sketch vectors. You can scale an SVG file without it losing quality, which is a super-useful thing when you are dealing with a responsive site. Clear candidates for a transition from bitmap to an SVG format are user interface icons, logos, diagrams, and illustrations.

As images go, this is one of the most important steps you can take when making your site responsive. Converting to or sourcing your icons and illustrations in SVG format may take a little time, but the benefits of using an image that does not become pixelated when stretched, and that is usually smaller in size than its bitmap counterpart, will keep paying you back. You should also remember to put in place a process that makes sure in the future these types of images are always created in SVG.

Do not think that all images should be converted to SVG, though. If an image has more complexity, such as gradients and drop shadows, re-creating it in SVG will likely increase its file size. So make sure you carefully compare the file size before and after and assess whether any increase is a compromise worth taking.

If you want to go a little further, you can even consider a more radical change to a flatter style of images and illustrations in future redesigns or updates to your site. Along with other responsive image techniques, I explore the more technical side of implementing SVG on your responsive site in Chapter 4.

Some Useful Responsive Web Design Patterns

Several design patterns have become commonplace when creating responsive sites and are becoming more and more intuitive for users as they come across these patterns on many different sites. You want to make sure, when you are converting your own patterns to responsive, that you look at what the most common design patterns are. In many cases, those will be just right for you. Other times, your particular case will require a little more design exploration, and you will need to come up with a different solution.

Do not disregard commonly seen responsive design patterns in a quest to be different. Many times, perhaps even more often than not, the common patterns are prevalent because they are the simplest to understand, not the other way around.

Here are a few key responsive design patterns that may be useful for your site when handling two of the trickier elements: navigation and tables.

Navigation Patterns

Smaller sites with a simple information architecture may find easier solutions, but when things start to get more complicated, thinking about navigation responsively can give you headaches aplenty. The following patterns tackle web sites with complex navigation systems using intuitive solutions that may work for your site, too.

Dropdown and Slidedown (Single and Multilevel)

This is arguably the most ubiquitous solution for web sites with sprawling IAs, including newspapers and e-commerce sites (see Figure 3-19). It usually involves either an icon or a label (or both) that sits in the header section of the screen and opens a list of links to the various sections of the site. The list can either cover the content (dropdown) or sit above it (slidedown) when it is open. Variations of this pattern include multiple levels of links, which can be opened and closed from within the list of top-level links (see Figure 3-20).

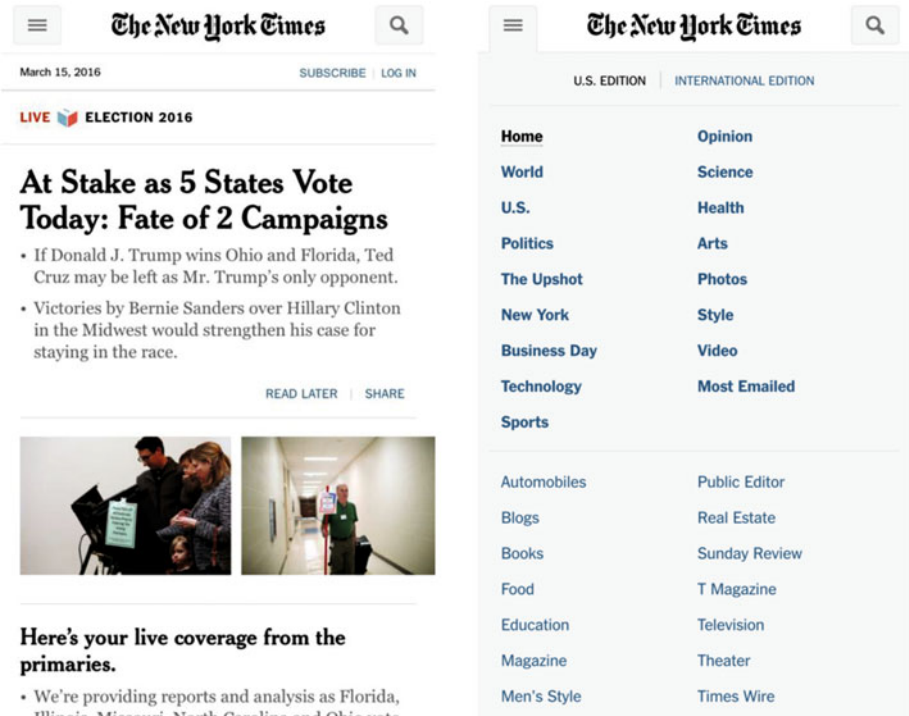


Figure 3-19. The New York Times uses a dropdown on its small-screen navigation

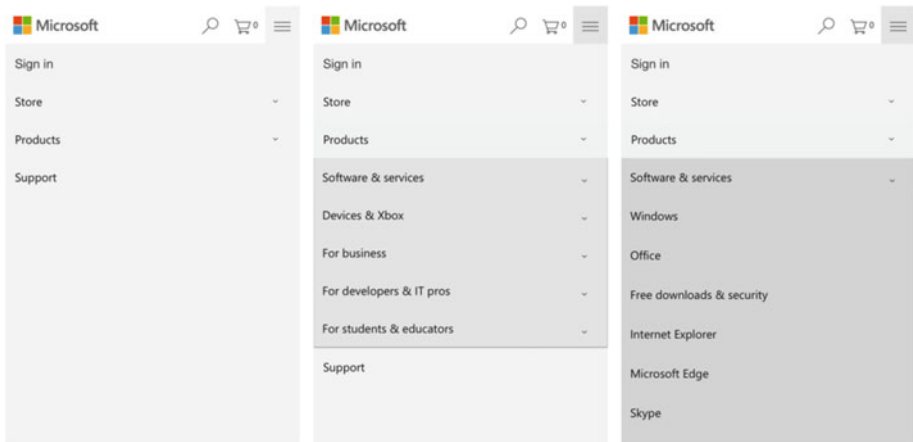


Figure 3-20. Microsoft also adopted the dropdown style of navigation on its web site, but it includes multiple levels of links

Side Drawer

The key feature of a side-drawer navigation is that the content is pushed to the side (either the left or right) so that a “drawer” containing the navigation items can be revealed (see Figure 3-21). To go back to the main content view, the user usually has to either click the button again or swipe the content back into view.

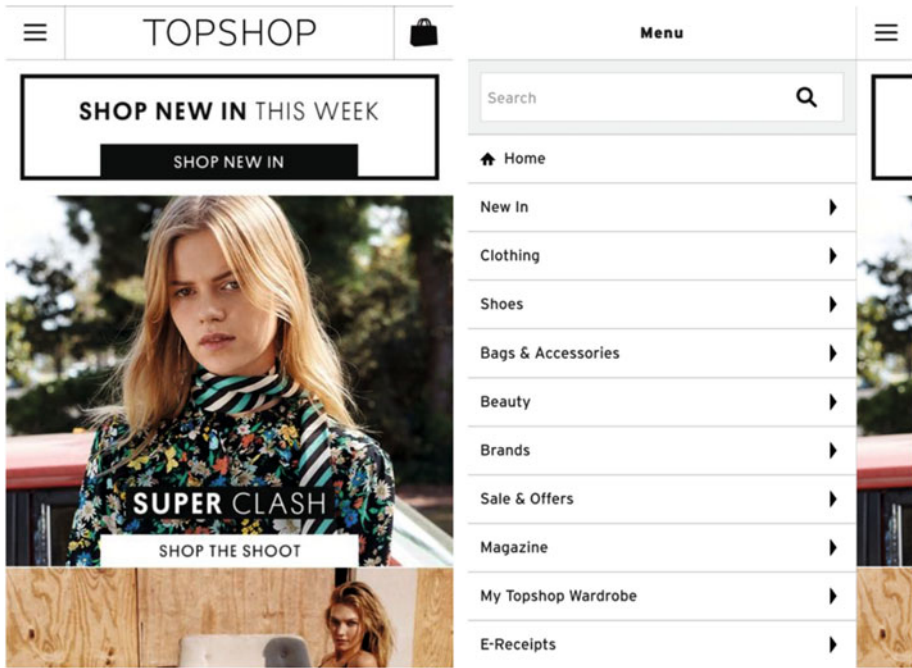


Figure 3-21. *Topshop.com* moves the content to the right to reveal a side drawer when the user clicks the menu icon

Priority+

The priority+ navigation pattern in a horizontal navigation bar works by hiding less-important links behind a More link, while the prioritized links are always visible. As the size of the viewport increases, more links can be revealed (see Figure 3-22).



Figure 3-22. The BBC News web site reveals more top-level links as the viewport grows in width. But it hides the links completely behind a Sections button at the smallest sizes

Overflow

This pattern can be useful when you want to maintain the horizontality of a wide-screen navigation on a smaller screen size, also giving you the flexibility to have as many top-level links on your navigation as you need. In a way, it works similarly to the priority+ pattern; but in this case, to be able to see all links, the user scrolls horizontally within the navigation area to reveal more links.

If you apply this pattern on your site, you may want to consider purposefully cropping the last visible link, to hint to the user that there is more to see (see Figure 3-23). If you do not do this, you run the risk of your users not realizing there are more links available (see Figure 3-24).



Figure 3-23. *The Guardian hints to the user that there are more links under the All button*

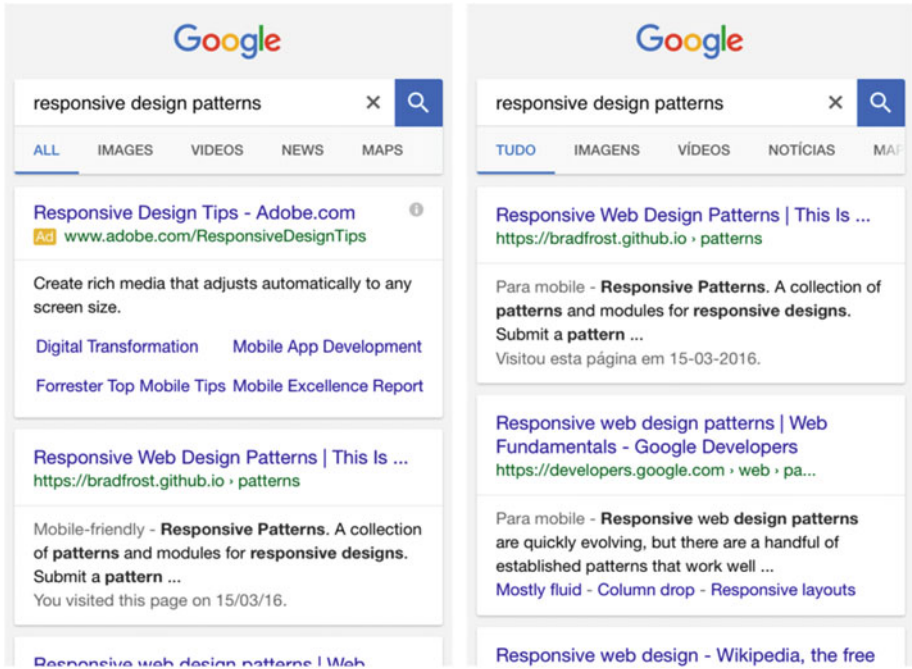


Figure 3-24. Google's search results in English give no indication that the user can see more type of results. However, a localized version of the same navigation crops the last link, suggesting that there is more

Tables

If we put images to the side, I think tables are one of the trickiest design problems to solve when converting what was once a fixed-width site to be responsive. The patterns that follow will provide you with some inspiration and ideas on how to handle these unwieldy creatures.

Overflow

This is possibly the simplest solution for responsive tables. In this case, the table shrinks with the viewport just until the content would become too squished in the columns. At this point, the table does not grow any smaller. Instead, part of the table overflows the width of the viewport, and the user can swipe to see the rest (see Figure 3-25) or change the orientation of the screen to increase the visible area.

Overview



Main article: [List of Lost episodes](#)

Season	Episodes	Originally aired		Nielsen
		Season premiere	Season finale	U.S. viewers (millio
1	25	September 22, 2004	May 25, 2005	15.69
2	24	September 21, 2005	May 24, 2006	15.50
3	23	October 4, 2006	May 23, 2007	17.84
4	14	January 31, 2008	May 29, 2008	13.40
5	17	January 21, 2009	May 13,	10.94

Figure 3-25. Some Wikipedia entries include some very large tables. The site’s mobile solution is to simply allow the user to swipe horizontally to see the part of the table that does not fit in the viewport

Table to List

When you apply this pattern to a data table, the content is reorganized so that it is effectively not a table anymore. Each row of data is rearranged into a list of label/description pairs so the user can easily understand which row and column heading relates to each data cell (see Figure 3-26).

Rank	Movie Title	Year	Rating	Reviews
1	Citizen Kane	1941	100%	74
2	Casablanca	1942	97%	64
3	The Godfather	1972	97%	87
4	Gone with the Wind	1939	96%	87
5	Lawrence of Arabia	1962	94%	87
6	Dr. Strangelove Or How I Learned to Stop Worrying and Love the Bomb	1964	92%	74
7	The Graduate	1967	91%	122
8	The Wizard of Oz	1939	90%	72
9	Singin' in the Rain	1952	89%	85
10	Inception	2010	84%	78

Rank	1
Movie Title	Citizen Kane
Year	1941
Rating	100%
Reviews	74
Rank	2
Movie Title	Casablanca
Year	1942
Rating	97%
Reviews	64
Rank	3
Movie Title	The Godfather
Year	1972
Rating	97%
Reviews	87
Rank	4
Movie Title	Gone with the Wind
Year	1939
Rating	96%

Figure 3-26. jQuery Mobile shows an example of how to reorder the content of a table at smaller screen sizes so that each row's content is grouped as a single unit

Priority Columns

By using the priority columns responsive pattern, you give users the ability to show and hide the columns they want to see on the table. You can choose to present only a selection by default, and users can then add or remove the columns they are interested in by means of a list of check boxes above the table (see Figure 3-27).

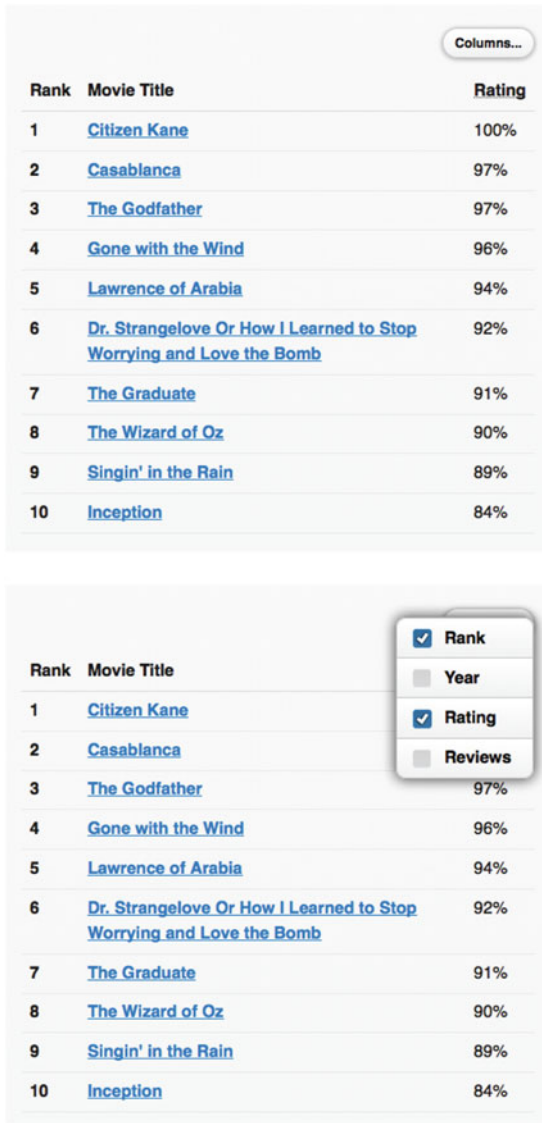


Figure 3-27. jQuery Mobile includes an example of a table that lets users show/hide the columns that are relevant to them

Getting the Most Out of Feedback and Reviews

Reviewing the work as you go along is part of the process of any design project, and yours is probably not an exception to the rule. There will be stakeholders who want to be kept up to date with your progress and others whose decisions will influence the path of the project. Whether they are the design director or the company's CEO, you should make sure you define the design-review and sign-off process for your responsive project.

When time is restricted, it is important that feedback be requested in a clever manner to be useful and actionable, that reviews run smoothly, and that they are translated into useful time spent first discussing designs and prototypes and then making improvements that in the end will benefit the user. It is also of the utmost importance that you and your team be able to justify with solid arguments why you have made certain choices over others in your designs.

Summary

If you are reading this book, you probably do not have the time and resources to spend on a complete redesign of your site. This chapter outlined a few tricks and techniques you can apply to your existing fixed-width site that will get you closer to a fluid, responsive web site in no time.

You have learned the following:

- How to focus on reusability and accessibility when redesigning your site
- Not to forget about performance while you design your responsive site
- How to make more decisions in the prototyping stages rather than using flat mockups
- How to quickly delve into prototyping with a rules document
- How to determine which breakpoints to apply to your site
- How to create your first style guide and think of it responsively
- How to do some quick-and-dirty user experience design and usability testing
- How to convert your existing grid to responsive
- How to start thinking about responsive solutions for your images
- Some useful, common responsive patterns
- How to get feedback and do design reviews right

The next chapter looks at some ideas and techniques you can apply to your responsive retrofitting project in order to improve the way your site is built and maintained.

CHAPTER 4



The Build Stage

Schedules, wireframes, sketches, copy documents, images, illustrations, and mockups are all pointless if they are not turned into reality by markup and code. Building a responsive site is in many instances an opportunity to assess and improve existing processes. When you are retrofitting an existing fixed-width site, the considerations are different than when you start a project from scratch. But you will still want to make sure you improve the way your site is built and maintained as much as possible. Ideally, you began thinking about the build of your responsive site at the same time you started planning and designing it, because all the different disciplines can and will impact on each other.

This chapter looks at a few strategies you can apply to building, releasing, and maintaining your responsively retrofitted site. These strategies will save your team time and do away with any fear that may come from experimenting with new technologies and processes as part of a responsive state of affairs.

In this chapter, you learn the following:

- The importance of trying things out in small projects
- How to consider accessibility in your code
- How to work with a performance budget
- How to start converting your grid and typographic scale into being responsive
- How to begin using media queries
- Useful solutions for dealing with images in a responsive world
- How to consistently test your responsive web site across different devices
- How to start considering improvements to your writing and publishing processes
- How to measure success after release

Experiment on Smaller Projects

When responsively retrofitting your site, in addition to simply making it responsive, you probably want to take the opportunity to introduce new technologies and ways of coding and building the site that can improve it and benefit your team's processes. But, many times, making those changes on a project of considerable size can seem like a daunting prospect. If this is your case, consider trying and testing new things on smaller projects.

It is less scary to try something new in less conspicuous and less complex projects, rather than jumping straight into the main site. When you try out new ideas and assumptions on smaller projects, it is easier to take a step back if something does not work out as planned. It is easier to revert changes, fix bugs, or try something else entirely. And if something goes wrong, it is also likely to affect a smaller number of people.

The things you can test on a smaller project do not necessarily have to be code related. However, it is on the building and coding side of things that having to deal with a bigger site and code base usually makes trying new things more complicated.

Here are a few things you can consider testing on smaller projects:

- Using scripts you have not used before, like Modernizr
- New image solutions, such as moving to SVG images
- New responsive typographic scale
- Modern CSS techniques, such as flexbox
- New design and interaction patterns, like a new navigation menu pattern
- Improvements to the build process, such as the addition of minification and concatenation of stylesheets
- New testing procedures, including automated tests
- A/B testing

There are countless more things you can consider trying on a smaller scale before you commit to them on your larger site. The key idea to keep in mind is that a change in the way you code and build your site does not have to be tested in its final location—the main site—and that smaller projects can serve as a playground for trying cool new techniques.

Focus on Accessibility

As I have mentioned in previous chapters, making sure your site is accessible should not merely be a step at the end of your project. Accessibility considerations begin with your content, whether it is in the form of words, images, videos, or any other format that communicates a message to someone or allows them to do something.

Accessibility thinking should also be ingrained in the design process: considering alternative ways of displaying the same message, and making sure designs and layouts do not get in the way of people accessing your content or completing a task.

But in the same way that mockups and wireframes are pointless without markup and code, great content and accessible design will not suffice if that same markup and code do not respect accessibility considerations. Andrew Hoffman sums up this idea very clearly:

Coding accessibility is not an extra thing to consider at the end of a project, but simply another thing to consider from the beginning.¹

Here are some of the key aspects to consider when coding your responsive site accessibly:

- Use clean, semantic markup as much as possible.
- Use WAI-ARIA landmark roles, if needed, to extend the semantic capabilities of your markup.
- Provide a skip-navigation link at the top of your site.
- Make sure non-decorative images include alt tags, but do not simply repeat text that is already mentioned elsewhere on the page.
- If you use JavaScript, provide fallbacks in navigation elements, because broken navigation can hamper access to content.
- Consider how people with touchscreens will use your site.
- Consider how someone with only a mouse or keyboard will navigate your site.
- Make sure someone using a screen reader can navigate your site.
- Consider how game console and TV users will navigate your site.
- Add accessibility tests to your build process to test things such as color contrast and the existence of alt tags.

This list should be a starting point for you to think about accessibility and to focus on making sure your site is accessible, if you do not do so already. This is by no means a comprehensive list, but rather a way for you to see how some small shifts in thinking about how you build your site can have a positive impact on its accessibility.

I encourage you to explore this topic in more depth, particularly in the areas that you feel least comfortable or knowledgeable about. Sometimes techniques that may sound complicated or cumbersome are not. You will only be able to advocate for and focus on accessibility in your code if you understand the true complexity of implementing such changes and are able to explain it to others.

¹Andrew Hoffman, “Accessibility: The Missing Ingredient,” May 13, 2014, <http://alistapart.com/article/accessibility-the-missing-ingredient>.

Focus on Performance

Converting an existing fixed-width site that was originally designed to fit medium to large screens is not just a case of shrinking images and removing floats from your CSS. One of the main secrets to making sure your site provides a good experience on any device, of any screen size, is to make it lighter and faster.

The classic idea of the mobile user who is moving from one place to another, is task-focused, and is on a slow connection is not accurate in many cases. Most of us can think of moments when we slouched on our sofa, browsing aimlessly on our phone while on a fast broadband connection, just a few feet from our laptop or desktop computer. Conversely, what about that time the usually fast connection on the desktop computer at the office was incredibly sluggish, and every web site took ages to load?

It is important that you avoid making assumptions about the context in which users will visit your site. You cannot assume that visitors will be accessing your site on only the fastest, low-tariffed Internet connections. A responsive site enables people access from an almost infinite number of devices, anywhere around the world. For this reason, it is key that you aim to build a responsive site that not only is *visually* responsive, but that also *responds* quickly to the user—and that respects the fact that they may be paying through the roof for any unwanted or unexpected downloads.

The nature of a responsive retrofitting project, where scope can be incredibly restricted, may mean improvements toward a performant web site have to be made in stages. There are a few aspects you should bear in mind as you make strides toward a more performant web site, and some things you can do quickly that will bring you closer to a super-fast site.

Work with a Performance Budget

Performance considerations should begin early in the design stages, but the bulk of the work that needs to be done still falls onto development. One aspect that should be determined from the get-go is what level of performance is deemed acceptable—this can form the basis of a performance budget.

A performance budget works just like your household budget, but instead of money, your currency is time or megabytes. You begin with a defined, finite amount, and you distribute the resources with thrift and prudence. Just as you have to cut down on restaurant visits when you really want that new Eames chair for the living room, you may have to cut down on your JavaScript libraries if you really want that large hero image on the site.

Setting the initial budget for your site is an exercise where you can and should involve team members with both design and development expertise. Start by looking at your own site and two or three competitor sites with the development tools in your browser open, and measure how long they take to load. Make an informed decision about where you want your site to land.

You can also use a purpose-built tool like WebPagetest (www.webpagetest.org), which allows you to test and analyze your web site's performance. The site gives you a plethora of useful information about your site and helps you identify and rectify particular bottlenecks.

When the BBC developed its responsive news site (see Figure 4-1)² half a decade ago, the stated goal was to have a working site in 10 seconds over GPRS (the mobile data standard that has since been superseded by EDGE, 3G, and 4G, but that is still the fastest available option in many places around the globe). By setting a goal like this, you can work out a few baselines: for instance, GPRS averages out around to a download speed of 100 Kbps, meaning you can download 125 KB in 10 seconds.

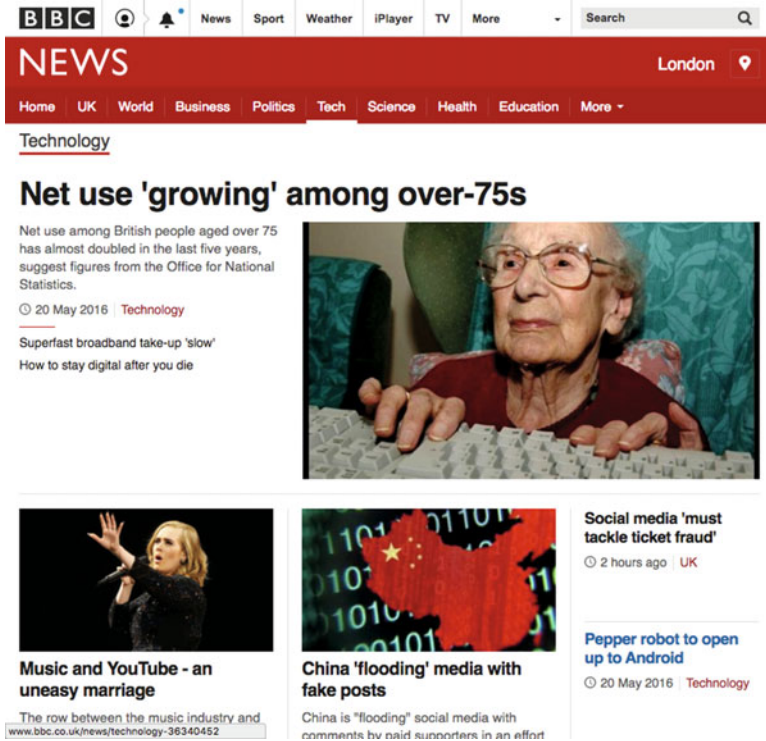


Figure 4-1. BBC News responsive web site (www.bbc.co.uk/news/technology)

Your goals will likely be different, but the approach is the same. On the basis of this information, you can make decisions about which and how many assets you can allow on each page.

Having a benchmark to work with that is easy to understand and communicate also helps in discussions with other team members and stakeholders. You have decided on a maximum page weight and set that as your maximum spending. If for any reason you or someone in another department deems it necessary to add images, banners, scripts, or video to the page, the discussion should always include the performance budget, and stakeholders should be made aware that it is an integral aspect of a successful design.

²Tom Maslen, “Moving Swiftly: The Story of How BBC News Fell in Love with Responsive Web Design,” Speaker Deck, September 25, 2012, <https://speakerdeck.com/tmaslen/moving-swiftly-the-story-of-how-bbc-news-fell-in-love-with-responsive-web-design#63>.

Because this is a retrofit, many design decisions that impact performance have already been made. By setting a goal, however, you can do several things under the hood to improve performance.

Optimizing the way your site requests its resources is one of the key ways to speed up the site. This is a science unto itself and deserves its own dedicated book, but the following checklist is a good starting point:

- *Size matters.* This may sound obvious, but a lighter page will download faster than a bloated one. It goes without saying that this becomes even more evident on a slow connection.
- *Minimize requests.* Every time the browser has to go fetch a resource, your page gets slower. Fewer requests mean a faster-loading page. One of the easiest improvements you can make is to ensure that you are loading as few stylesheets and JavaScript files as possible.
- *Put things in the right order.* Keep CSS in the HTML head element, before any linked JavaScript. That way, your browser can begin rendering the page before dealing with anything else.
- *Keep it lean.* The browser needs to download and parse both your HTML and all of your CSS before it can start rendering your page. Yes, size matters—it is worth repeating.
- *Scripts are slow.* As the browser parses your page’s HTML, it has to stop when it encounters a script. Then it has to execute the script to check whether the DOM has been altered. If the script is external, it has to fetch the resource, download it, and execute it. This can cause significant delays. If you can, let your page render before fetching and executing your JavaScript asynchronously.

These are only a few things you may want to consider improving on your newly retrofitted responsive web site that can make it considerably faster. But sometimes speed only needs to be skin deep, as you see in the next section.

Understand Perceived Performance

Page weight is not everything. A user does not know whether your web site has too many HTTP requests or is this or that many megabytes. Their experience is based on what they see. And you do not perceive that something is faster or slower in a vacuum; you can only tell that something is fast if you can compare its speed with something else.

When trying to improve your site’s performance, aiming for it to be 20% faster than before is usually a good goal. This value has its root in how human perception works. According to Weber’s Law,³ “the just-noticeable difference between two stimuli is proportional to the magnitude of the stimuli”; this means there is a point at which humans can perceive that there is a difference between two states, and a smaller difference than

³Wikipedia, “Weber-Fechner Law,” March 28, 2016, https://en.wikipedia.org/wiki/Weber%E2%80%93Fechner_law.

that will not be detected. This rule can be applied to anything from discounts to increases or reductions in how many potato chips are in a bag. It can also be applied to software development and user experience. The value of 20% is regarded as the baseline at which someone will perceive any difference in stimuli, so this number can be used to your advantage when assessing and determining performance-improvement goals.

Depending on how much earlier work and focus has gone into making your site performant, this performance gain may be achieved just by looking at some easy wins. Here are some things that can make a site feel faster:

- Having touch/hover states on buttons and links
- Optimizing time to first byte/time to render
- Including critical CSS inline
- Loading the most important above-the-fold content first
- Considering progressive/lazy loading, meaning you only load content that is visible in the viewport
- Avoiding spinners if you have to reload content, and using an animation instead
- Avoiding repaint, because pages that jump around as they load feel slower
- Considering loading fallback fonts before fetching web fonts to avoid a flash of invisible text (FOIT)

It is important to note that these tips and hints should be used in addition to the under-the-hood performance enhancements that reducing page weight, HTTP requests, and so on will bring.

Large Screens Like Lean Sites, Too

You should make sure your content has plenty of room to shine, and you should provide a comfortable reading experience for your user. There is only so much you can include in a small-screen view of your site, so you will probably be quick to do away with decorative elements and effects. If you do, stop to think about whether they add anything to the large-screen views, too. If your mobile users can live without that parallax effect on the background or that hijacked scrolling effect, are you sure it is necessary at all? We have a tendency to want to decrease page weight on mobile devices, but the benefits of doing so will improve the experience for every user, across all devices.

Consider the job your site is there to do. More often than not, a design is successful if you do not notice it. When reading a book, becoming engrossed in the characters and storyline is preferable to paying constant attention to the typography and layout. Similarly, if your site is there to provide content or a service, you should consider your priorities. Simple, unobtrusive, fast, and efficient is often better than complicated, noticeable, or even innovative and remarkable. As luck would have it, most people will love your fast, boring-looking site more than a slow, striking one.

Trim Down Your Web Fonts

Another simple thing you can do to shave some kilobytes off your site is to remove unnecessary character sets and font weights from your web font, if you are using one. Check whether you are using variants such as italics and bold—there may be design guidelines in place that restrict their use, and emphasis can be added to text in other ways. The gains of doing this can be in the hundreds of kilobytes.

If you are hosting your web fonts externally and the licensing of your fonts allows it, you can also consider self-hosting your fonts, because doing so cuts out the reliance on an external service. This allows you more fine-grained control of how and when to serve your fonts, and how to implement your caching strategy.

A downside to using web fonts is that some browsers (I am looking at you, iOS Safari) display no text at all until the entire page, including the web font assets, has loaded fully. This means you sometimes have to wait up to 30 seconds before seeing any text on the site. There aren't any simple solutions to this problem, apart from ensuring that your site loads fully very quickly, but there are a few tricks you can use to make the experience better.

One of these techniques, introduced by Filament Group, is to serve a fallback font the first time someone visits your site, load the web font asynchronously after the page has rendered, and serve the web font from that point on.⁴ Another, far more drastic approach is to remove the web font and rely on the fonts installed on your visitors' operating systems. In a recent site update for a small side project, replacing a single hosted web font with a system font saved 3.5 seconds in load time and a half-megabyte download. This technique is certainly worth at least considering when page weight is at a premium.

Removing web fonts is not possible for all projects, of course; but when you are reviewing design visuals hot out of Photoshop, remember that looking great is not the only way in which your site will be evaluated by users. You have to provide a good experience as well.

Some Handy Tools

Various tools can help you improve your site's performance. The ones listed here are some of my favorites and are popular among front-end developers because of their reliability. Make sure you take them for a spin.

WebPagetest

You have already been introduced to WebPagetest (see Figure 4-2) in this chapter. It is an open source tool supported by Google that aims to make the Web faster by providing metrics on all things related to page speed. It allows you to test your sites on real browsers across the world so that you can see how your site performs in different areas, which is great if you have a global audience.

⁴Scott Jehl, "Font Loading Revisited with Font Events," Filament Group, February 16, 2015, www.filamentgroup.com/lab/font-events.html.

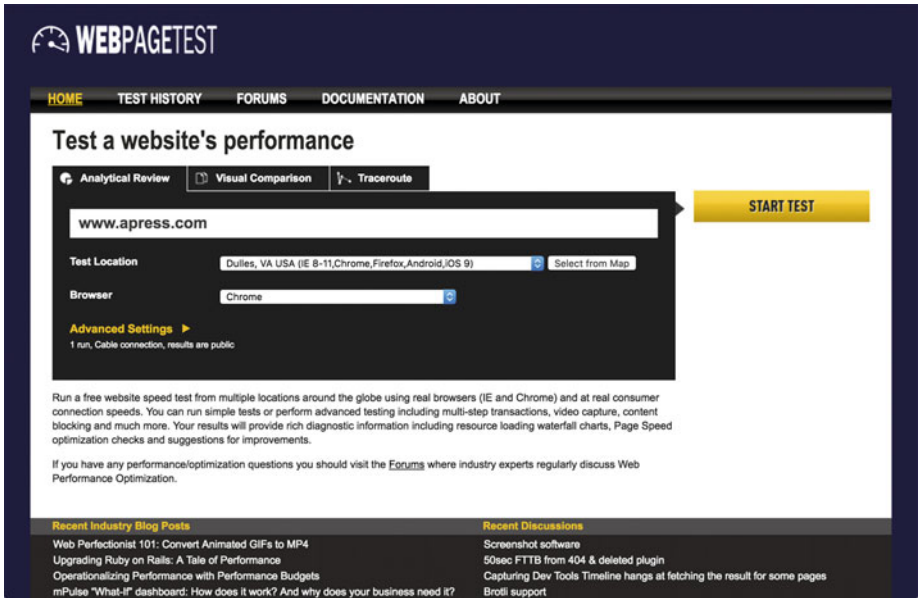


Figure 4-2. The WebPagetest tool

Speed Index

The Speed Index is one part of the WebPagetest suite (see Figure 4-3). It measures how quickly your page content visually renders. It can be particularly useful for before-and-after comparisons when doing optimization work, because it gives an indicator of the site's perceived loading speed. To keep up your team's morale, make a habit of noting the speed index before and after you make any changes that you think will have a performance impact, so you can see concrete, measured improvements as you go through the drudge of improving things behind the scenes.

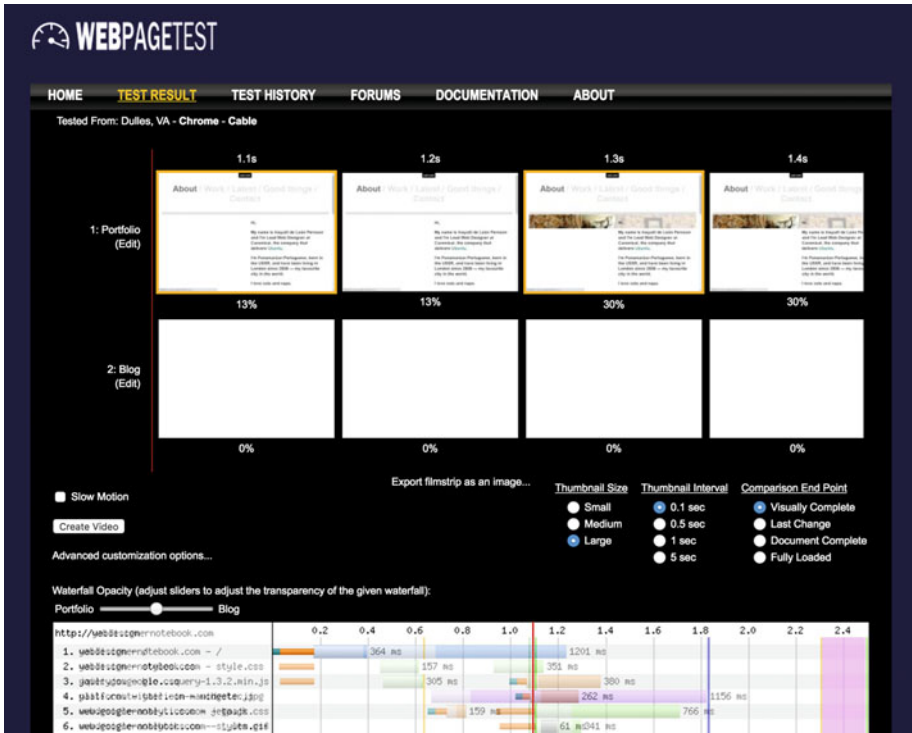


Figure 4-3. WebPagetest's Speed Index in action

What Does My Site Cost?

When carrying out a responsive retrofit, you will naturally have mobile use of your site in mind. The What Does My Site Cost? web site (see Figure 4-4; (<https://whatdoesmysitecost.com/>)), which forms one of the metrics of the WebPagetest, tells you how much mobile network users across the world have to pay for the data required to download and use your site. Discussions about performance often focus on reducing the number of requests and rendering pages as quickly as possible, so page weight often gets pushed into a corner. This tool serves as an important reminder that if your site becomes bloated and overweight, it costs people real money.



Figure 4-4. *What Does My Site Cost? web tool*

These tools form only a small subset of those available to analyze and improve performance. You can find more in the appendix.

Fluid Grids and Type

Fluid grids, along with media queries and responsive images, are one of the three tenets of a responsive site, according to Ethan Marcotte’s seminal A List Apart article “Responsive Web Design.” When converting a site to being responsive, one of the most important and visible tasks you will perform is making your grid and your typographic scale responsive. These changes do not have to be complicated, but it is important that the code you create is robust and that it provides a solid foundation for your content to live in.

Fast Track to a Fluid Grid

There is a quick test I like to do on sites that are not yet responsive, which can quickly provide you with a good idea about how much work you will have to put into making your grid responsive. All you need to do is remove float and width rules from your CSS to see how your site behaves in a linear format. This can be done quickly by using Firebug or another inspector and resizing your browser window, but it is even better if you can create a prototype that can be tested across a selection of real devices and that the team can huddle around and discuss.

You may be surprised by how good your site looks in this simplified attire, and by the small number of changes you would have to make to achieve a passable responsive experience. But fiddle with your prototype for long enough, and you will begin to find numerous ways in which you can massively improve the small-screen experience of your users. List these, prioritize them, and make yourself a plan.

Improve your Markup

Ideally, your site was built following the principle of separation of concerns, where the HTML gives your content a consistent and logical structure, the CSS describes style and layout, and JavaScript adds behavior and other enhancements. If this is not your case, you should give serious consideration to refactoring your markup to better reflect these ideas before you delve any deeper into your retrofitting work. This is obviously much easier said than done, but remember that when you are working with messy HTML, you will more likely than not end up with messy CSS. Your responsive retrofitting project, although tight in scope, should still provide you with a venue to make some much-needed improvements to the foundations of your site.

In his article “Responsive Retrofitting,” Ben Callahan makes this very same point evident:

*Avoid doing retrofit work when the site doesn't have a solid foundation of clean, semantic HTML.*⁵

There are some things to look out for when you are considering improving your markup:

- Use semantic markup such as headings, paragraphs, lists, and HTML5 sectioning elements to describe your content.
- Keep your markup consistent by making sure everyone follows the same guidelines.
- Keep divs and other elements that do not describe content structure to a minimum, especially when they are used to describe layout.

⁵Ben Callahan, “Responsive Retrofitting,” April 3, 2013, <http://webstandardssherpa.com/reviews/responsive-retrofitting/>.

- Use the HTML5 doctype if you do not do so already.
- Validate your markup: there may be things you did not catch that a validator will.
- Consider making the HTML improvements suggested by Google Webmaster Tools for your site.

In some cases, it may be possible (or necessary) to leave the existing HTML untouched. In this situation, it is likely that your CSS will have to work harder to be able to target the HTML you need to style. You can also consider using CSS flexbox to reorder content at different breakpoints, independently of the order of the content in the markup, if it is not possible for you to edit the HTML. Although not necessarily recommended for full-scale grid-layout design, flexbox can come in handy to reorder smaller bits of content in your pages.

Let's imagine you have a row that, on a large screen, shows on its left half a heading and a paragraph of text, and on its right half an image (see Figure 4-5):

```
<section>
  <h2>Down the Rabbit-Hole</h2>
  <p>Alice was beginning to get very tired of sitting by...</p>
  
</section>
```

Down the Rabbit-Hole

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures...



Figure 4-5. A simple row with text and an image

If you simply reflow and stack this content in a smaller screen view, you will have the heading, followed by the paragraph, followed by the image. However, you may think the image should come after the heading on a small screen. Flexbox can solve this:

```
section {
  display: flex;
  flex-direction: column;
  align-items: flex-start;
}
```

```

h2 {
    order: 1;
}

p {
    order: 3;
}

img {
    order: 2;
}

```

First, you define that the children of the parent container, section, use flex. Second, you define the order (or axis) in which you want to lay out content: in this case, in a column. Third, you define whether the items should take up the entire width of the column or be aligned another way along the axis. In this case, you want to keep the items' left alignment, so you define `flex-start`. If you did not add this property, the elements would try to fill the entire width available, and the image might get distorted in the process. Finally, you define a different order than the one in the markup, in which you want your items to be arranged: first the heading (`order: 1`), second the image (`order: 2`), and third the paragraph (`order: 3`).

As you can imagine, there is much more to flexbox than this simple example. But now you know that even when you cannot touch the markup, there is the possibility to reorder some elements within it.

The global support for flexbox is just over 94%,⁶ with Internet Explorer only supporting the property from version 11 onward. So, make sure you consider how non-supporting browsers will render your web site.

Remove Inline Styles

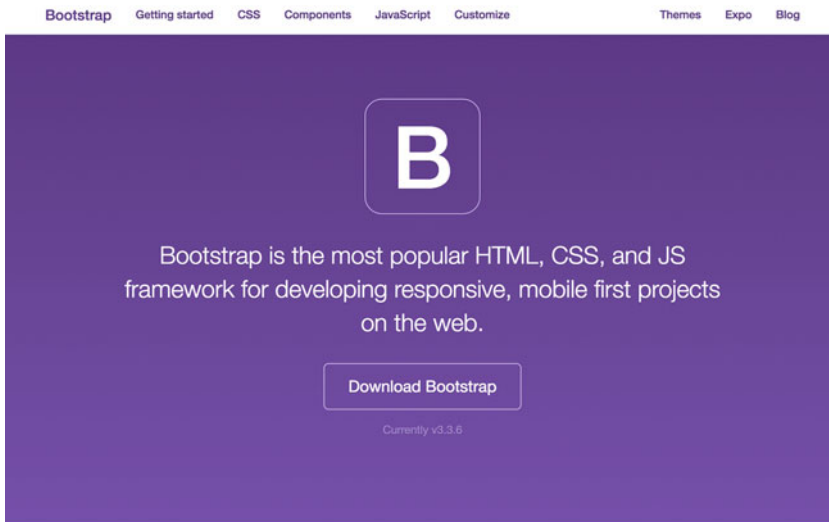
It is not uncommon for inline CSS to creep its way into your markup. Sometimes things need to be built that do not follow any existing patterns and that are enough of an exception that adding a single style tag is the simplest way to achieve the intended effect. But once you make your site responsive, these inline styles may come back to bite you.

Before you move too far into your development, you may want to sweep your site to find any pesky inline styles, including those added via JavaScript, so that you can then make the decision to either leave them or remove them, case by case. Depending on how many styles you have let sneak into your markup over the years, this decision-making process may be a quick task or something more involved; but it is something you do not want to remember at the last minute or after the site has gone live. Some tools online can help you with this task, such as HTML-Cleaner (<https://html-cleaner.com>). While you do this, you should also search for instances of inline JavaScript and consider refactoring those before they become a problem.

⁶Can I Use, "Flexible Box Layout Module," <http://caniuse.com/#search=flexbox>.

Get Inspired

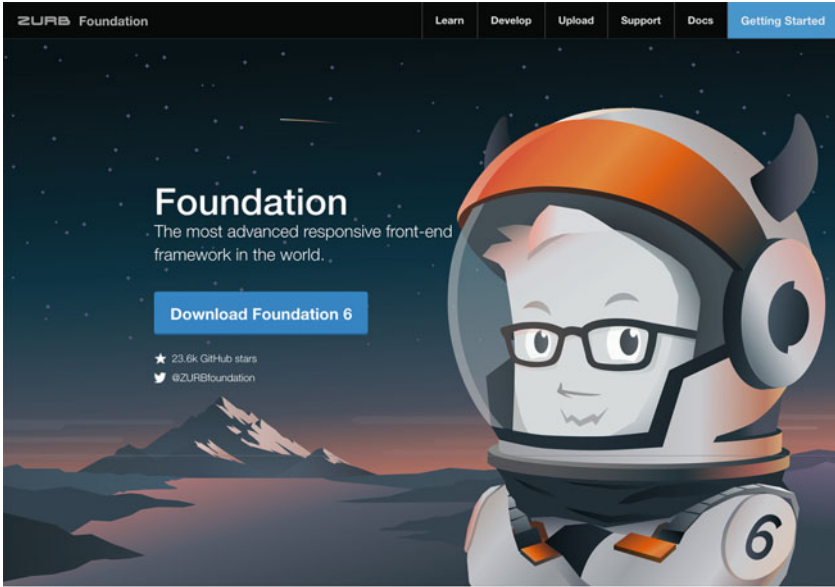
Consider looking into how popular front-end frameworks deal with responsive grids (see Figures 4-6 and 4-7). When I say this, I do not mean you should simply copy and paste their CSS, or even use the framework; but checking them out is a good way to see how some issues that come with creating a flexible, robust, responsive grid can be solved.



Designed for everyone, everywhere.

Bootstrap makes front-end web development faster and easier. It's made for folks of all skill levels, devices of all shapes, and projects of all sizes.

Figure 4-6. *Bootstrap, currently the most popular front-end framework (<http://getbootstrap.com/>)*



Responsive design gets a whole lot faster

A Framework for any device, medium, and accessibility. Foundation is a family of responsive front-end frameworks that make it easy to design beautiful responsive websites, apps and emails that look amazing on any device. Foundation is semantic, readable, flexible, and

Figure 4-7. Foundation by ZURB (<http://foundation.zurb.com/>)

When going through these frameworks' code, remember that their target audience is in many cases everyone. This means their markup and CSS are made to be much more malleable than yours probably needs to be. This need for malleability can and does cause bloat and unnecessary classes and markup, so remember that this is not necessarily your case.

Another thing you may want to do is to look at well-designed responsive sites that have a solid grid, check how their grids are built, and see how the CSS was written and how the markup works with it. Look at how your competitors have solved the same problems, and think about whether you can come up with a better or cleaner solution that is more adapted to your needs. This is, and always has been, the beauty of the Web: the ability to view source (see Figure 4-8). Use it to your advantage.

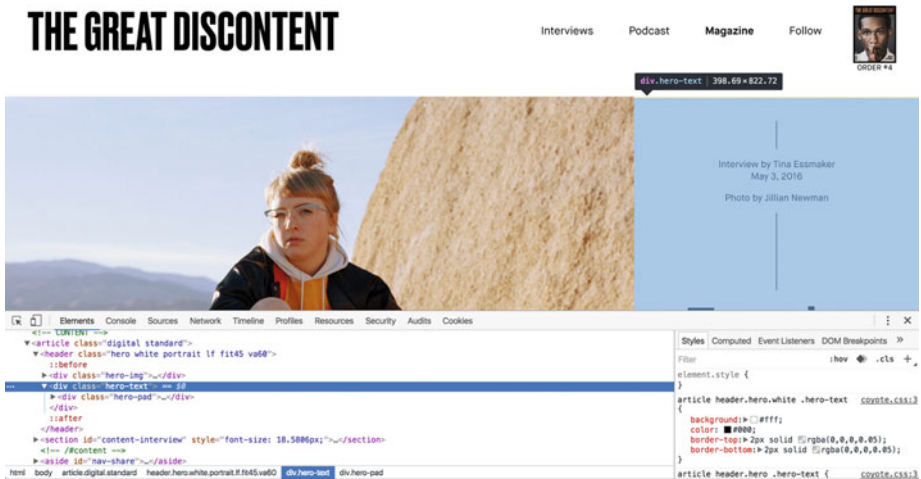


Figure 4-8. Inspecting the beautiful responsive grid of *The Great Discontent's* web site (<https://thegreatdiscontent.com/>)

Abandon Absolute Units

Depending on how your stylesheets have been created in the past, you may have to convert the way you created your grid and font sizes from using absolute units of measurement, such as pixels and points, to using relative units, such as em and percentages. When you define a column to be a certain width in pixels, that is the width it will always be, regardless of whether you are looking at the site on a small or large screen. The goal of making your site responsive is that it can adapt to any screen size and allow the content to use the space available as well as possible. You cannot do this with absolute units. Bear in mind that design guidelines and pattern libraries can be defined in pixels, because this is the unit in which measurements are generally computed, but that does not mean you should be using pixels in your CSS.

In a responsive retrofit project, you will most likely be converting some sort of absolute unit to percentages to have a fluid grid. You may run into trouble when you need to mix percentages with absolute units for things like gutters and sidebars—for instance, you may want a sidebar on your site that takes up only 200 pixels in width, and the main content should take the remaining width depending on the viewport. In this case, consider using the CSS function `calc()`, which allows you to do exactly that type of math:

```
aside {
  width: 200px;
}

main {
  width: calc(100% - 200px);
}
```


It sounds like magic, right? Bear in mind, though, that this function's support is less than 80% globally at the time of writing, so make sure to add fallback for non-supporting browsers.⁷

Scalable Type

As I mentioned in the “Adjust a Strict Typographic Scale” section in Chapter 3, nothing can replace testing your responsive typographic scale on real devices with different screen qualities. (Remember not to test only on the latest Retina screen.) The same applies to creating the CSS that goes along with it.

The main change in making sure you have a responsive typographic scale is usually the conversion of a unit such as pixels into something more flexible. The unit of choice tends to be the em. However, because the em is relative to the inherited font size of the target element, compounding issues may arise.

If you wish to avoid the problems that can come with using em units as your font-size unit, you can use rem units instead. Rem means “root em,” and it is relative to the root element (the html element in this case) instead of being relative to its parent.

Rem browser support is quite good, with almost 95% global support at the time of writing.⁸ It is supported by Internet Explorer from version 9, with the caveat that IE 9 and 10 lack support for styling pseudo elements and require you to specify the font-size values separately instead of in the shorthand format throughout. You may want to look at the REM unit polyfill if you want to provide rem support for older IE versions (<https://github.com/chuckcarpenter/REM-unit-polyfill>).

Something else to keep in mind when building your typographic scale is that if you specify a unitless value for the line-height property (instead of one with a specific value, like em), that value will multiply the font-size value by the defined amount. So if the font-size is 1 rem, the computed value is 16 pixels, and the line-height value is 1.5, it translates into the computed line-height value being 24 pixels (16 pixels × 1.5). This means you can set a line height across all breakpoints, instead of defining one for each breakpoint, which will make your type even more responsive.

Media Queries

Media queries are the second of the three technical tenets that form the basis of a responsive web site. The purpose of media queries, as specified by the W3C, is to allow you to tailor your page “to a specific range of output devices without changing the content itself.”⁹

⁷Can I Use, “calc() as CSS Unit Value,” <http://caniuse.com/#search=calc>.

⁸Can I Use, “rem (Root em) Units,” <http://caniuse.com/#search=rem>.

⁹W3C, “Media Queries,” June 19, 2012, www.w3.org/TR/css3-mediaqueries/.

According to the W3C specification, a media query is made up of one “media type and zero or more expressions that check for the conditions of particular *media features*.” Let’s take a look at a simple example and break down this statement down:

```
@media screen and (width: 500px) {
...
}
```

The first part, `@media`, is what is known as a conditional CSS `@rule`. It contains a statement that can be evaluated to true or false. If true, the declarations following the rule are evaluated.

Next, `screen` specifies the media type. Other supported types are `all`, `print`, and `speech`. Although previous revisions of the media query specification included more media types, these are now deprecated in favor of media features, which offer more fine-grained control.

Finally, `(min-device-width: 500px)` is an example of a media feature. In this case, it checks that the width of the device is equal to or larger than 500 pixels. If the conditions after `@media` are true, the declarations in the curly brackets (`{ . . . }`) are executed.

With the power of media queries, you can tailor the design and experience of your site as precisely as you feel is necessary so that it adapts to different screen sizes. There are a few things to bear in mind when transitioning from a media query-free state of affairs to using media queries.

Mobile First, and Enhance Progressively

As part of the work you do on your CSS, consider refactoring it to work in a mobile-first way. This means the initial CSS rules are the ones designed for small screens, whereas the rules in media queries target larger screen sizes.

If you currently use any type of media query in your CSS, it is likely following the opposite approach: the default styles target large (and perhaps medium) screen sizes, and separate rules are added in a media query to target smaller-screen devices. The notion behind this approach ties deeply with the principles of progressive enhancement.

When following the mobile-first principle, you usually start with a more linear representation of your content. This simplified layout is likely to work well in old browsers and devices, as well as for screen readers and other accessibility aids, and it will form a good foundation on which more complex layouts can be built.

The process of simplifying your layout to work in a mobile-first kind of way falls mostly in the disciplines of design and user experience. But you can also apply this principle to your stylesheets independently.

Consider spending some time turning your CSS upside down. Depending on the scale and complexity of your existing stylesheets, this may be an easier or more difficult task, but it is worth it in the long run if you can afford it.

This exercise consists of moving all the CSS rules that are not relevant to a mobile-sized view into a media query. Let's look at an example. Say your layout includes a sidebar and an article area:

```
<article>
  <h1>...</h1>
  <p>...</p>
  <p>...</p>
</article>

<aside>
  <ul>
    <li>...</li>
    <li>...</li>
  </ul>
</aside>
```

And your CSS looks something like this:

```
article {
  float: right;
  width: 75%;
  background-color: #f7f7f7;
}

aside {
  float: left;
  width: 20%;
  background-color: #333333;
}
```

In a linear, mobile-first world, you can determine that your article area will be stacked above your sidebar, and both will fill the entire width of the container. In that case, your refactored CSS looks like the following:

```
article {
  background-color: #f7f7f7;
}

aside {
  background-color: #333333;
}
```

You remove the float properties, because both containers are stacked; and you also remove the width properties, because they take the full width of the viewport.

The next step is to add a media query at the breakpoint where you want the original large-screen layout to kick in, and reinsert the float and width properties. Let's say this should happen at a minimum viewport width of 700 pixels:

```
article {
    background-color: #f7f7f7;
}

aside {
    background-color: #333333;
}

@media screen and (min-width: 700px) {

    article {
        float: right;
        width: 75%;
    }

    aside {
        float: left;
        width: 20%;
    }

}
```

This is, of course, a very simplified view of the type of work you will have to carry out, but the principle is always the same: outside of media queries should only live the styles that power a beautiful small-screen view of your site. As with anything else, you should make sure you test a prototype on real devices as soon as possible.

The advantage of this approach is not only related to the fact that you are progressively enhancing your layout with media queries. You are also slightly improving the speed of your page render. When your browser attempts to render a page, it loads the HTML first and parses it to build the Document Object Model (DOM): that is, the structure of the page. It also loads all of your CSS and builds the CSS Object Model (CSSOM), informing the browser how to style the page. Importantly, any CSS in a media query is marked as non-render-blocking until the media query has been evaluated. This means although the entire stylesheet is being downloaded, the browser does not have to stop and evaluate anything in a media query that does not apply to the current viewport. In other words, the critical content your browser is parsing first is all being used. For a device with a slower processor, this can only be a good thing, because evaluating, discarding, and re-evaluating stylesheets takes up valuable processor cycles.

Browser Support

At the time of writing, the global audience using a browser with media query support is of almost 95%.¹⁰ Only Internet Explorer 6 through 9 (released in 2011) and Firefox 2 and 3 (released in 2008) do not support media queries, so you may be tempted to ignore the small percentage of users with no media query support. But 5% of global usage represents a fair number of people, so it is good practice, and important, to also consider the experience you are delivering to visitors using browsers that do not support media queries.

If your CSS has been created following mobile-first principles, this means non-supporting browsers show the small-screen layout, even on large screens. This may be acceptable in your case, but depending on the number of users who are still visiting your site on older IE versions, you may have to consider using a polyfill that enables these browsers to understand media queries.

Respond.js (<https://github.com/scottjehl/Respond>) is the most popular polyfill for this effect. It weighs only 1 Kb when minified and gzipped, and it adds support for the media query types that are most relevant to building a responsive web site: `min-width`, `max-width`, and `media` types. It is worth exploring it to see if it matches your site's particular needs.

Whatever way you decide to cater to older browsers, it is unlikely that you will be able to not use media queries on a responsive web site, because they are so fundamental to the notion of what *responsive* is.

Responsively Retrofitted Images

Responsive images are the third tenet of responsive web design, and probably the one that has been the cause of the most headaches and debate among web developers. As much as we would like to think that all we need to do to have responsive images on our sites is to set their width to 100%, it is not that simple.

As you saw in the previous chapter, each web site can have an assortment of different image types that are used in different ways and contexts, all requiring a different type of responsive solution. Let's look at some of the most relevant solutions you may want to put into practice when converting your site to being responsive.

SVG Images

If there is only one thing you change in the way you deal with images on your new responsive site, it should be starting to use SVG images. As I mentioned in the previous chapter, user interface icons, logos, diagrams, and illustrations are prime candidates to become SVG images.

SVG is not the best format for all types of images, though. More complex images like photos and complex logos are probably better as bitmap images. SVG lends itself to images that were initially created as vector graphics, because it best describes lines, shapes, and flat colors. When in doubt, compare the file size of the resulting SVG image with its bitmap equivalent. For complex images, in many cases, even doubling the physical size of the bitmap image will result in a smaller file size than SVG, which compensates for the loss of resolution when the image is sized up.

¹⁰Can I Use, "CSS3 Media Queries," <http://caniuse.com/#feat=css-mediaqueries>.

When it comes to browser support, SVG has nearly 95% global support at the time of writing.¹¹ If you want your images to display correctly on Internet Explorer 8 and below, you may have to rely on a polyfill or another fallback technique. The same is true for IE 9 through 11, depending on how you need to implement your SVG images, because these browsers support SVG only if used in certain conditions.

You can use Modernizr to detect whether a browser supports SVG, and serve it a different format image. This will not solve the defective support in IE 9–11, though, because these browsers do support the feature—just not at its full capacity.

Regardless of the downsides, the benefits of using SVG images where possible are manifold. The two key benefits are tightly coupled with the goals for which any responsive retrofitting project should be striving: fast and lean experience, and light but perfectly scalable graphics.

Picture and srcset

The `picture` element and the `srcset` attribute were added to the HTML5 specification to simplify the creation of responsive images by relegating the responsibility for selecting the best image for the situation at hand to that which knows best: the browser. Let me give you a quick overview of the two before we move on.

In simple terms, `srcset` allows you to define a number of images and either their width in pixels or their pixel density, so that the browser can select the most appropriate image. It is better suited on its own (without using the `picture` element) if all you need to do is provide the same image in different sizes for the browser to choose from. Let's take an example:

```

```

Here you specify a small version of an image in the `src` attribute. This image will also serve as fallback in case the browser does not support the `srcset` attribute. You then specify a medium-sized image that is 724 pixels wide, a large image that is 1,024 pixels wide, and a super-large image that is 1,600 pixels wide. All the browser needs to do is choose the one that is more appropriate for the viewport at hand.

You can also specify the images' pixel density instead:

```

```

¹¹Can I Use, “SVG (basic support),” <http://caniuse.com/#search=svg>.

If you need to provide different images for each situation (for instance, if at a smaller size you want to show a cropped version of a larger image), you should use the `picture` element. Let's take another example:

```
<picture>
  <source media="(min-width: 900px)"
    srcset="medium.jpg 900w, large.jpg 1600w"
    sizes="(min-width: 1400px) 50vw, 100vw">
  
</picture>
```

Let's catch our breath and begin from the top:

- The group of `source` and `img` tags is wrapped in the `picture` tag so the browser knows how to interpret them.
- If the media query in `source` is matched—the viewport has a min-width of 900 pixels—the browser *will* choose one of the images in `source`'s `srcset` and skip any elements that follow.
- If the media query in the `source`'s `sizes` attribute is a match to the viewport (`min-width: 1400 px`), the image chosen will be displayed at the defined size (50 vw), otherwise, it will be displayed at the default size (100 vw).
- If no other elements match, the `img` tag's `src` is used.
- The `img` tag is always required.

You can have multiple `source` tags in a `picture` tag, and you can also have `srcset` and `sizes` in the `img` tag, but I did say I was going to be brief. It would easily take as much space as this entire chapter to go through all the ways in which the `picture` element and `srcset` attribute can be used and combined to provide the best image for each situation. I encourage you to dig into the current spec for yourself and experiment with different solutions.

At the time of writing, global browser support is just over 60% for the `picture` element¹² and just over 66% for the `srcset` attribute¹³ but growing quickly. And remember that if the browser does not support either of them, there is always a fallback image to rely on.

If you are wary of using this type of cutting-edge technique, remember what I mentioned at the start of this chapter: you can always experiment on a smaller project to test the waters, and if it does not work for you, find another solution. This could also be a way to test the feasibility of existing polyfills, such as Filament Group's `Picturefill` (see Figure 4-9).

¹²Can I Use, "Picture Element," <http://caniuse.com/#search=picture>.

¹³Can I Use, "Src Attribute," <http://caniuse.com/#search=srcset>.

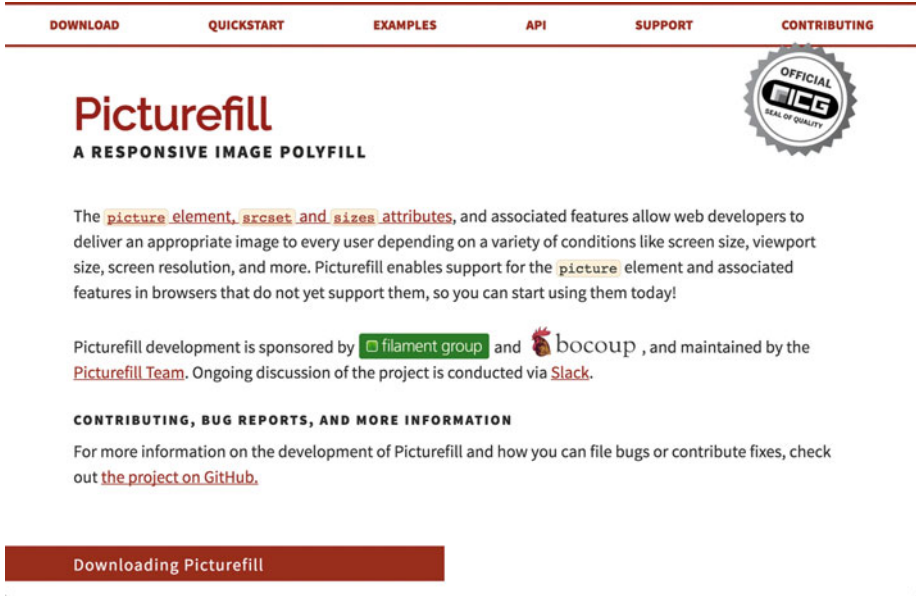


Figure 4-9. Filament Group’s Picturefill polyfill (<https://scottjehl.github.io/picturefill/>)

Even if you decide that these are not appropriate for your web site right now, it is important that you look into them carefully: they will become ubiquitous at some point in the future, and learning them is key.

Image Caching

Good responsive images are not just ones that scale up and down with the viewport. They are also responsible images: they should not be larger than necessary and cause users to download more than they need to render the site on the device of their choosing. If your site uses the same images in more than one place, you should make sure you are always using the same instance of the image.

If you display on your site logos from partners or products, or user interface icons for things like search, shopping cart, and so on, anyone working on the site should never have to create a new instance of the same logo or icon. And similarly, a user who visits your site more than once should not have to download these assets twice.

You can create logo packs and icon packs that live in a central location where someone working on new designs can download them. Ideally, every time you need to reference them in your site, you should use the same source file in the same location. You should also ensure that your site’s server is set up to cache the images appropriately, if you are not doing so already.

Compress Those Bitmaps

Often, when deadlines are looming, content and image creators are not necessarily paying much attention to the size of the images they upload onto content management systems—and many times, CMSs are not great at cleverly compressing images when they are unnecessarily large. Another of my favorite exercises that you can follow when making your site responsive is to go through the site with a tool like YSlow (<http://yslow.org/>) to pinpoint the location of the worst offenders, so you can resize them for better performance.

Once you have a list of where the extra-large images are hiding—they tend to hide in plain site, though—compress the bitmap images that do not need super-high quality or a transparent background. This is a quick, simple way to reduce the file size of the images across your site.

A neat trick you may want to try when you want crisp images on Retina displays but still want to save on file size is to export the JPEG at twice its size using a high compression rate. Because you will not display the image at its original size, the fact that it is a little blurry will not be noticed. Even though you are using an image that is twice as large as necessary, the file size you save with compression makes the resulting file size smaller than if you had exported the same image at the correct size with little or no compression. Always make sure the balance between the resulting file size and image quality passes a visual inspection, though, because in some cases the quality of the compressed image may not be what you are looking for.

Optional Images

Optional images are ones that are not crucial for the experience of your site. All decorative images fall into this category. If an image is simply adding a little flourish to the page to liven it up, you may want to consider loading it only on larger screen sizes, making sure users on small screens are not unnecessarily downloading the image.

In theory, decorative images should be added via CSS as background images. If you do this, the background image can be added in a media query that targets larger viewports, and you avoid the extra download on smaller screens. If the image is added directly into the HTML, however, it will be much more difficult, if not impossible, to stop the unnecessary download.

It would be nearly impossible to cover all possible solutions for responsive images that are currently in circulation; but now you have a good overview of some things you can test and try on your retrofitting project, along with some key techniques to consider. Bear in mind that, depending on your site's needs and the types of images you are using, you may well have to apply more than one technique to build a solution that is best for your users.

Testing

Making sure you test your web site and prototypes on real devices as soon as you begin working on your responsive retrofit is without a doubt one of the key ingredients for a successful project. If you leave testing for later, you run the very real risk of getting too far into the build and the decision-making process before you spot big mistakes and bugs that could have been more easily and cheaply fixed at earlier stages of the project.

What to Test On and What to Fix

It is a good idea to define which browsers, devices, and operating systems you are going to test your site on. By doing this, you are not determining that you will not fix bugs or make improvements on anything else; this simply serves as a guide for your team, and other teams, to ensure that testing is conducted in a systematic and unambiguous manner.

Having a document and process that everyone follows, no matter who is testing or what is being tested, ensures that your site will always comply with certain standards. Determining what to test on depends greatly on your existing users.

A good starting point is to analyze your analytics numbers and see which devices are the most popular ones that people try to access your site with. The keyword here is *try*. If you do not have a good mobile experience, it is unlikely that your analytics will reflect the same usage as if you did. Repeat visits would probably be higher, and the devices used most frequently might be different, too. Regardless, you have to begin somewhere, so existing numbers are perfectly acceptable.

Do not focus simply on mobile device data, though. You want to make sure your new responsive site has been tested across a wide range of browsers and operating systems, and that includes desktop-like environments.

I cannot tell you what the minimum threshold should be for your team to determine whether an issue should be taken care of; that is up to you and your special circumstances. It is not possible to say that you should not fix any bugs for browsers with fewer than 1% of visitors, because 1% can mean an enormous number of people for high-traffic web sites. If your site gets 100,000 visitors per month, 1% translates into 1,000 users. If your site gets 10,000,000 visitors per month, 1% means 100,000 people. Is it worth improving the experience of 1,000 people? What about 500? Or 100? That is a question that only you, your team, and your company can answer.

You should also check the most popular current devices and combinations of browsers and operating systems. Your own analytics may not reflect the current state of affairs, but that does not mean it will stay like that once you release your beautiful new responsive site.

BrowserStack has created a very convenient list of devices and operating systems that you should be testing on, based on global trends (see Figure 4-10). This list is updated frequently, so it is a useful resource when determining which devices to add to your testing guidelines and keep on hand.

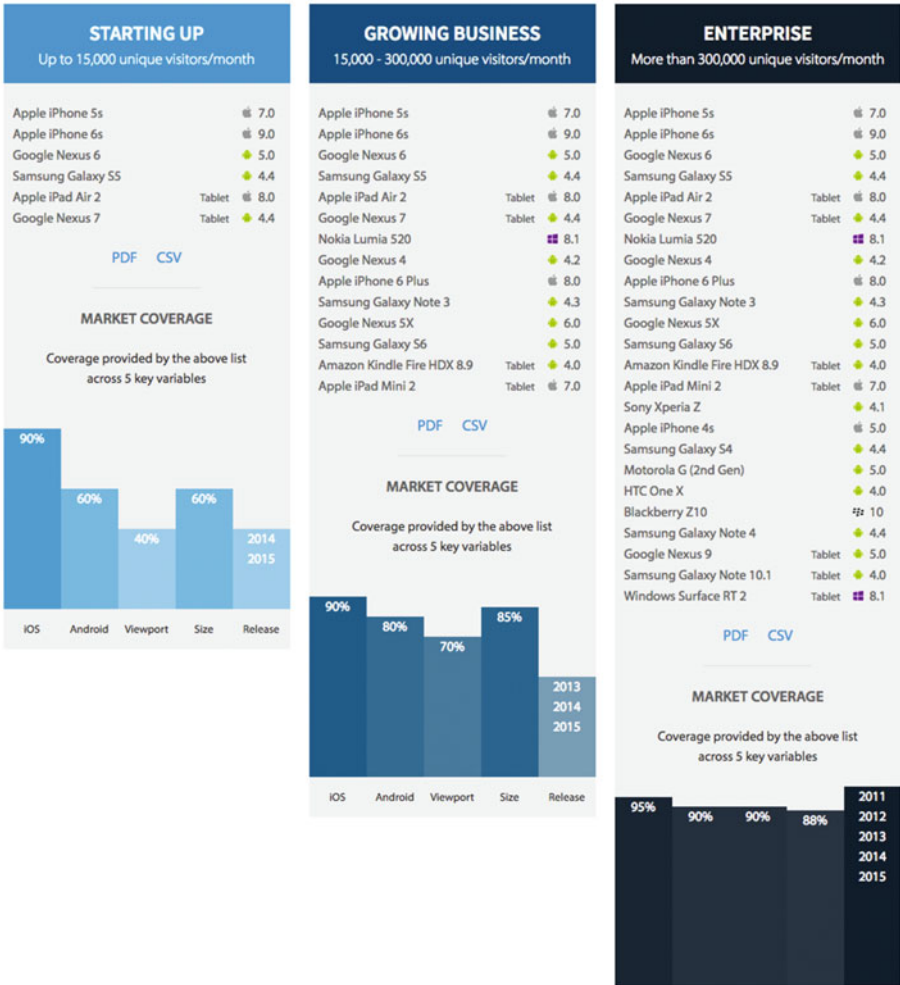


Figure 4-10. BrowserStack’s “Test on the Right Mobile Devices” page (www.browserstack.com/test-on-the-right-mobile-devices)

Remember to add tricky operating systems and browsers to your testing protocol, too. Make sure you do not just test on easier ones, like the latest iOS phone and tablet and the latest Android release. Test on older versions of these operating systems, and on non-default browsers, such as Opera Mini and UC Browser, which are popular in China and India and have an expanding user base.

Build Your Device Lab

The idea of having your own set of devices to test on can sound like an expensive endeavor. But you can build up a device suite, even if you are on a tight budget, by considering your analytics and which devices your visitors are using. Building your own device lab should be something you do early in the project, because you want to start testing as soon as possible.

Just as I recommended in the previous section, fall back on your site's analytics data to see the devices on which people are using your site. This will give you a better idea of which devices are more of a priority for your team to get access to.

Once you have a wish list of devices you feel are important for you to own, there are a few things you can do to put together your own lab:

- Use auction sites like eBay to get a better deal on new or like-new devices.
- If you decide to not spend on more expensive devices like iPhones and iPads, there is money left in the pot to buy a larger number of other devices.
- Ask if anyone in your company and your team has old devices that they do not use anymore that they would like to donate. People tend to forget about these at the bottom of their drawers, so it is always worth asking.
- Set up a dedicated testing machine that runs several instances of Windows and Linux with different versions of Internet Explorer and other browsers that may be relevant for you to test on.

As the project evolves, your device suite will likely grow as your needs change. After you release the first iteration of your site, whether you are doing it in a closed way, internally, or open to the world, people will begin to access your site from devices that you may not have tested on. An easy-to-follow strategy to expand your device lab is to acquire devices with operating systems from which you receive a fair number of bug reports.

As I said before, ideally, you will be testing on your own set of real devices; but you can also consider using a public device lab, if there is one in your area. In some cities across the world there are Open Device Labs, which are open for designers and developers who wish to test their sites across a multitude of real devices but do not have a suite of their own (see Figure 4-11). There may be one or more near you, so it is always worth checking.

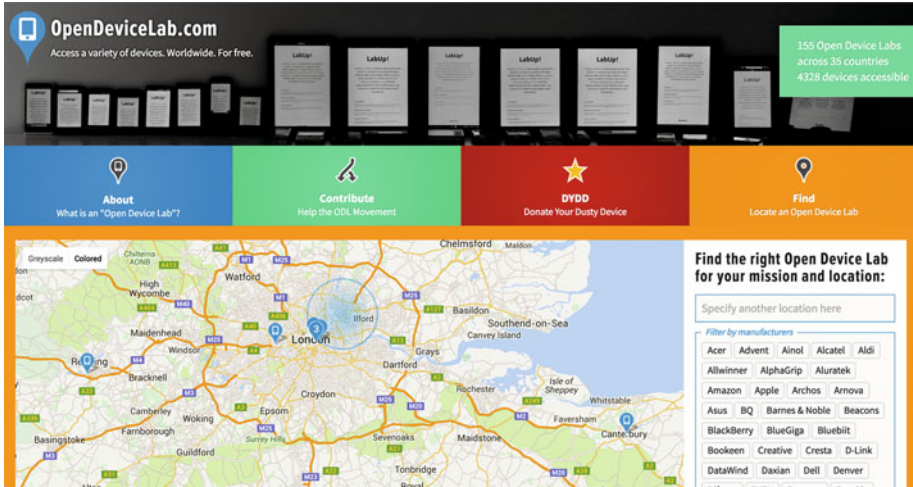


Figure 4-11. Open Device Lab web site (<https://opendevicelab.com/>)

Emulators and Other Tools

Even if you build the ultimate device lab, you will not be able to physically have all the possible devices people may use to browse your site on. Several tools can help you fill in the gaps when the real deal is not available or if you are working from another location but still need to do testing.

BrowserStack is a popular testing tool among designers and developers because it provides cloud access to an incredible array of operating systems and browsers without the need to install virtual machines on physical devices (see Figure 4-12). One of the biggest advantages of using BrowserStack is that it provides access to operating systems that are harder to find in real life. Older versions of popular operating systems can sometimes be tricky to locate, so it is handy to be able to test a specific bug that may only be reported on a less-common platform. It also lets you securely test sites that live on development and staging environments, which is extremely useful particularly in the early stages of the project.

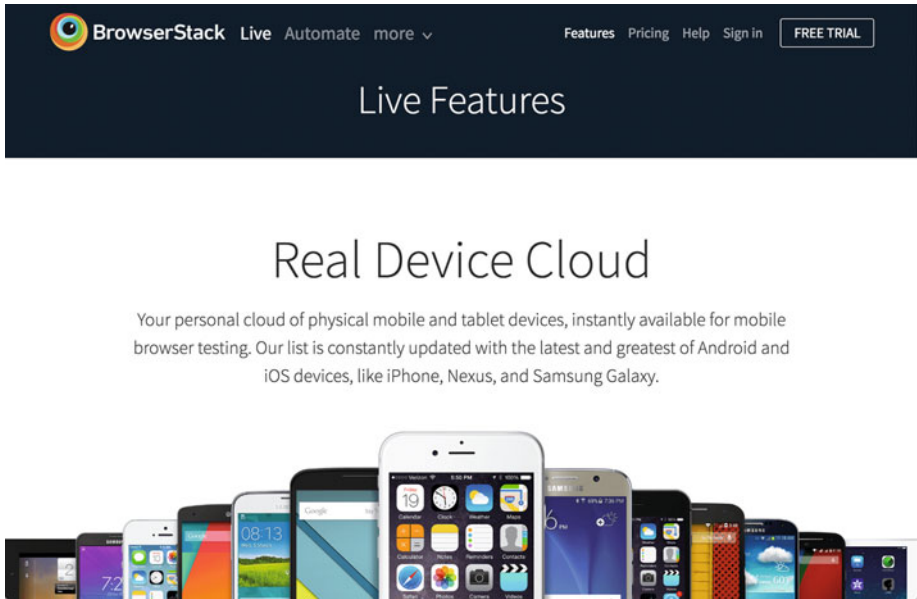
The image is a screenshot of the BrowserStack website. At the top, there is a dark blue navigation bar with the BrowserStack logo on the left, followed by the text "Live Automate more" with a dropdown arrow. On the right side of the navigation bar, there are links for "Features", "Pricing", "Help", and "Sign in", along with a "FREE TRIAL" button. Below the navigation bar, the main heading "Live Features" is centered in a large, white font. Underneath this, the sub-heading "Real Device Cloud" is also centered in a large, black font. Below the sub-heading, there is a paragraph of text: "Your personal cloud of physical mobile and tablet devices, instantly available for mobile browser testing. Our list is constantly updated with the latest and greatest of Android and iOS devices, like iPhone, Nexus, and Samsung Galaxy." At the bottom of the screenshot, there is a row of various mobile devices, including iPhones and Android phones, each displaying a different mobile operating system interface.

Figure 4-12. BrowserStack offers you a virtual device lab

Another useful tool to have in your arsenal is Ghostlab (www.vanamco.com/ghostlab/). It performs a different purpose than BrowserStack (see Figure 4-13): Ghostlab makes it possible to test simultaneously on various browsers and physical devices. So if you are testing a certain interaction pattern on a specific page, whatever you are doing on your computer also happens on all the other connected screens at the same time. You can even inspect the CSS and markup on one device, make changes, and see them reflected in all other devices. Very neat, right? Ghostlab also supports Sass, Less, Haml, CoffeeScript, TypeScript, Jade, and Stylus, and testing on development environments.

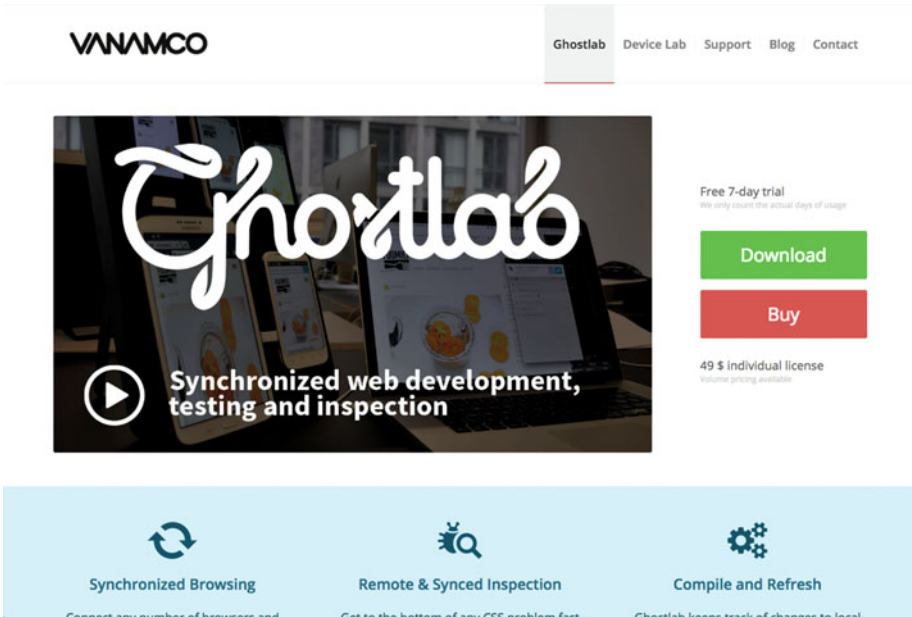


Figure 4-13. Ghostlab allows you to test simultaneously on multiple devices

Finally, Percy (<http://percy.io/>) is a visual regression tool that integrates with your continuous integration. If you already have CI set up, this is a great little addition. Percy takes screenshots of your site before and after you make changes to the CSS and push them to the server. It also, gloriously, allows you to test your responsive designs. You can give the service a list of breakpoints, and it takes and evaluates browser screenshots to check how the CSS changes have affected your site on desktop as well as mobile devices.

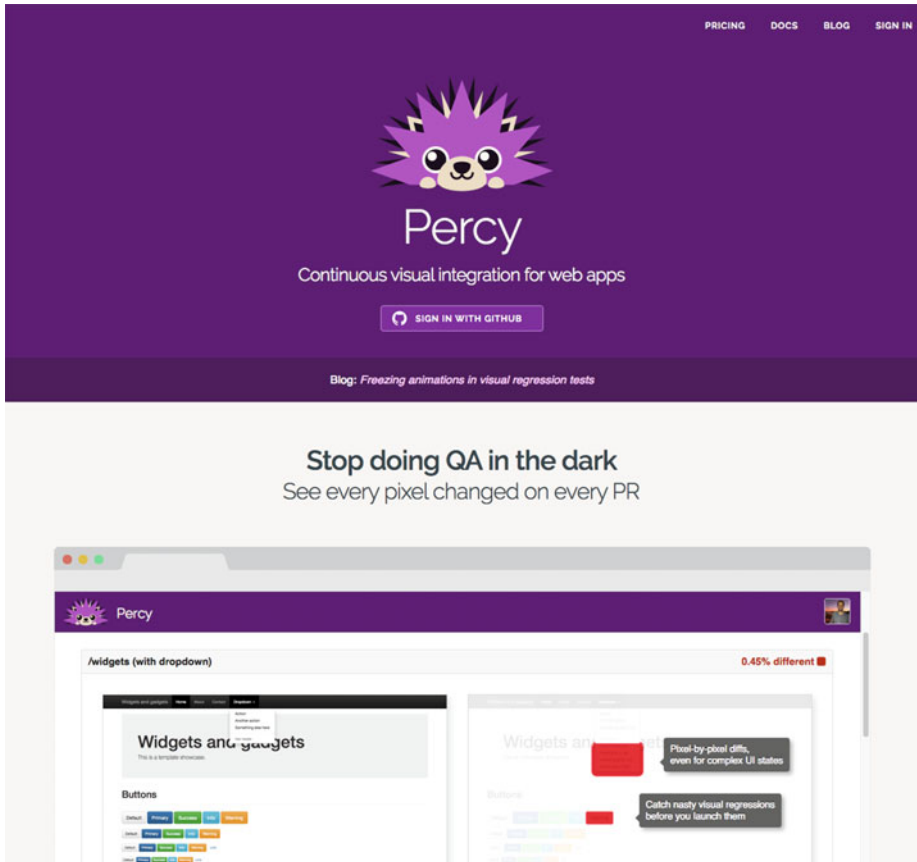


Figure 4-14. Percy, a continuous visual integration tool

Regardless of the number of devices you have, or whether you need to use emulators, testing is a vital part of any responsive design project and should be performed constantly and consistently throughout. Whether you have a very small or a sprawling suite of devices to test on, or you rely mainly on emulators and other tools, make sure to instill on your team the need to test early and often in a responsive world.

Improving Your Process

It is well beyond the scope of this book to go into any kind of detail about the many and varied options available to development teams today. But there are a handful of principles you may want to strive for as you think about how to improve your team's processes and workflow.

As I hinted earlier in this chapter, undertaking a responsive retrofit is a great opportunity to take a look at your working processes when writing, publishing, and maintaining code. Let's dive in.

Writing Code and Markup

When working in a team, you can follow a few core principles to make writing HTML, CSS, and JavaScript a more pleasant experience for everyone. Focusing on the quality and robustness of the code is important, but almost as important are the factors that matter most when you are knee-deep in stylesheet declarations or buried to the nose in HTML: consistency, readability, and predictability. Establishing a few rules and guidelines will help everyone collaborate and contribute consistently.

Have a Markup and Code Style Guide

It makes sense to formalize the conventions you are using. Write them down, share and use them, and do not forget to update them, too.

If you do not already have a style guide covering markup and code conventions, there are several good examples you can use as a starting point. For instance, the guidelines for Primer (<http://primercss.io/guidelines/>), the framework that powers GitHub, are a good, open source example to take inspiration from (see Figure 4-15).

The image shows a screenshot of the GitHub Primer code guidelines page. The page has a blue header with the Primer logo and navigation links for Docs, About, GitHub, and Install. A left sidebar contains a table of contents with categories like Scaffolding, Layout, Type, Buttons, Forms, Navigation, Alerts, Blankslate, Avatars, States, Tooltips, Utilities, Code guidelines, Colors, and Markdown. The main content area is titled 'Guidelines' and describes best practices for writing HTML and CSS. Below this is a 'Contents' section with a bulleted list of topics including HTML (General formatting, Boolean attributes, Lean markup, Forms, Tables), SCSS (Spacing, Formatting, Misc, Examples), File organization (Bundles, Including (S)CSS files), Pixels vs. ems, Class naming conventions, and Specificity (classes vs. ids, CSS Specificity guidelines). The page also shows the start of an 'HTML' section with a sub-section for 'General formatting' which includes a bullet point about using soft-tabs.

Figure 4-15. GitHub's Primer code guidelines

So, what best practices should a style guide cover? The answer depends on how you work, the projects you are working on, how many people are on your team, and other individual factors. The key is ensuring that your guidelines help you avoid confusion and conflict within your team and make things clear for new starters. Consider including guidelines about the following:

- How to format HTML markup.
- How to use indentation and spacing consistently.
- How to mark up specific elements, like forms and tables.
- Class-naming conventions. The use of prefixes, capitalization, and underscores can help identify what type of class you are dealing with.
- How CSS declarations should be formatted. This removes doubt when writing your stylesheet, and makes reading and updating it easier, too.
- How to use whitespace in your documents to aid readability.
- Whether to use tabs or spaces.
- How and when to write comments.
- Documentation practices in general.
- How to write tests, what to test, and when.
- Source-control workflow, including best practices for commit messages and pull requests.

If you are using Sass, your guide should also cover best practices regarding things like how deeply you should nest declarations, which helps to limit specificity issues that result from a too-deeply nested stylesheet.

The key is to have guidelines that work for your situation, are consulted and updated when conventions change, and help your team work together with less friction.

Using Sass Responsively

As you are toiling away in your text editor, one of the things that can cause more agony is having to repeat yourself unnecessarily. Typing `#ff69b4` over and over is not much fun; and if you grow tired of hot pink, having to find and replace all instances over several stylesheets will seem like a chore. This is the kind of problem preprocessors exist to help you with.

Simply put, preprocessors are scripts that take a file in one format, manipulate it in clever ways, and output the converted result at the other end. Although there are preprocessors for HTML and JavaScript, I am focusing on CSS in this section because it is likely the one most relevant to the process your team will be undertaking during a responsive retrofit.

One popular preprocessor that you may be using, or have considered using, is Sass (Syntactically Awesome Style Sheets; see Figure 4-16).¹⁴ Sass takes some of the pain out of writing CSS by eliminating the need to repeat yourself quite so much. But that is only the start of what it can do.

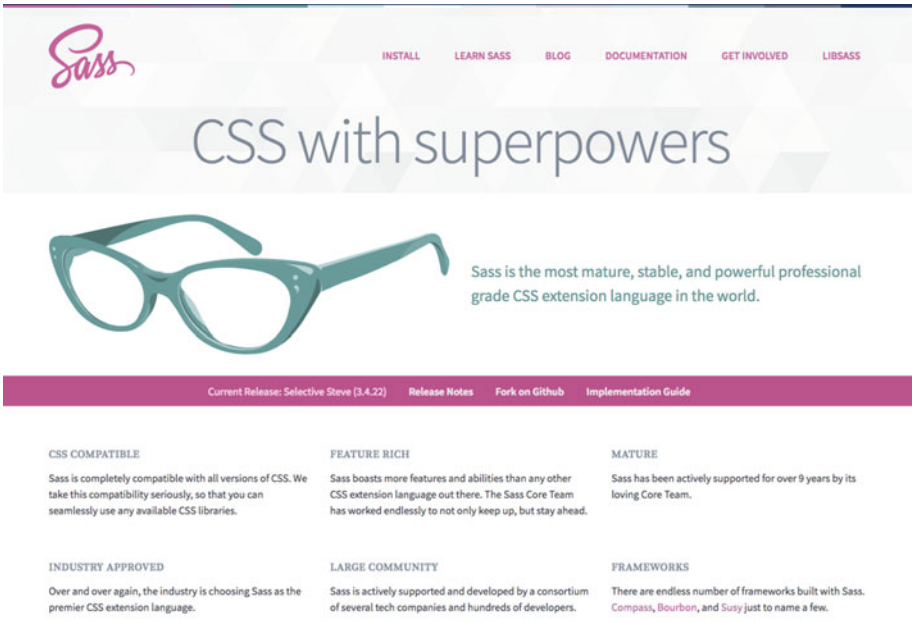


Figure 4-16. CSS preprocessor, Sass’s web site

Sass also fills in and makes some of the shortcomings of CSS easier to deal with: it provides variables, nested declarations and properties, mixins, functions, and much more. If Sass’s only feature was variables, it would still be worth using. By defining all of your site’s colors, measurements, and grid layout in one place, you can abide by the Don’t Repeat Yourself (DRY) directive¹⁵ and save countless tears, both when writing and when maintaining your site.

One example of how variables can help in creating a responsive stylesheet is in the definition of the grid. Let’s say you want to define a simple grid with a set number of columns and a defined max-width. Keeping these values in a central location means you know where to go looking for them.

¹⁴Sass is available at <http://sass-lang.com/>, where you can also find instructions for installation and use.

¹⁵Wikipedia, “Don’t Repeat Yourself,” https://en.wikipedia.org/wiki/Don%27t_repeat_yourself.

You can name the file anything you like, but `_variables.scss` has a nice ring to it. If you are curious, the underscore prefixing the filename tells the Sass compiler that the file is a *partial*: it will not compile this file into CSS, because the file's purpose is to be included in other Sass files using the `@import` directive. Put your grid and column definitions in this file:

```
// Defining our grid and columns
$grid-max-width: 55em;
$col-num: 12;
```

As you will have noticed, Sass variables are prefixed by a dollar sign (\$), making them easy to spot when used in the stylesheet. When you enter the variables to use, they look like this:

```
// We need to include the partial in our SCSS stylesheet.
// Note that we do not need to include the underscore prefix
// or the file extension in the import declaration:
@import "variables";

.container {
    max-width: $grid-max-width;
}
```

When this is compiled and output into CSS, the result looks like the following:

```
.container {
    max-width: 55em;
}
```

Usually, variables are used several times throughout a stylesheet, saving repetition when you have to change or update values. There is a case to be made, however, for keeping values that you only use once in a stylesheet as a variable too; doing so gives you one central point—your variables partial—where you can go to find all the relevant, crucial definitions for your site, such as color, grid measurements, typographic scale, and more.

Mixins and Functions

Another way in which Sass can help you avoid repetition is by using mixins and functions. Mixins let you define groups of declarations that you want to reuse throughout your stylesheet. Mixins also let you pass in values, making them even more flexible. For instance, you can use a simple mixin to avoid having to repeat the media query declaration every time you need it:

```
@mixin breakpoint($bp-size){
    @media (min-width: $bp-size) { @content; }
}
```

You can then combine this mixin with some variables that define your breakpoint sizes:

```
// Define our breakpoint variables
$bp-small: 20em;
$bp-medium: 50em;
$bp-large: 62em;
```

When using the mixin in a stylesheet, you do this:

```
@include breakpoint($bp-medium) {
    /* style declarations here */
}
```

This is just one example of how you can use the power of Sass to create responsive code snippets that can be used throughout your site to make life easier and your stylesheets easier to maintain.

If you are familiar with functions from any programming language, you will notice that they have many things in common with mixins. Sass also has something called *functions*, however. Whereas mixins are used to help write repeatable code just once, functions have the sole purpose of making calculations; they output a single value.

Functions are great if you want to make calculations where you need to output a number, but they can also be used to calculate strings, colors, and more. When you are getting into the intricacies of calculations for responsive design, functions may well be a useful addition to your toolbox.

Ethan Marcotte's book *Responsive Web Design*, First Edition (A Book Apart, 2011) introduces a function that allows you to calculate the percentage widths of a pixel-based design—a common task in a responsive retrofit. For instance, if you have a sidebar that is 375 pixels wide and a container that is 950 pixels wide, the function turns the sidebar's pixel value into a percentage. The formula $target / context = result$ can be expressed like this using a Sass function:

```
@function percentify($target, $context) {
    @return ($target / $context) * 100%;
}
```

You can then use this function in your Sass:

```
.sidebar {
    width: percentify(375px, 960px);
}
```

This is compiled as follows in your stylesheet:

```
.sidebar {
    width: 39.0625%;
}
```

We are often faced with numbers that seem to have been picked out of thin air when reading stylesheets. Having the functions that calculated them within reach is not a bad idea, especially when you are troubleshooting or refactoring your code. Anytime you have to use a calculator when writing CSS, you should ask yourself whether you would be better off using a Sass function, instead.

Organizing Your Stylesheets

There are several considerations when writing CSS; and as often is the case, these considerations can be conflicting. You want to write lean, efficient CSS, and you want your CSS to be performant; but you also want CSS that is easy to maintain. The good news is that with a few tricks and tools, you can have your CSS cake and eat it too. It just requires some thinking ahead.

It probably makes sense for you and your team to split the site's stylesheets up into several files dealing with distinct areas of your site. This is not required, and some teams prefer to work with one or two larger stylesheets; but separating your CSS into different files may provide some flexibility in your workflow and make your styles more manageable, as long as everyone knows the structure and follows the conventions.

When serving the site, however, having several stylesheets is a bad idea, because each request slows down the site. Luckily, our great web developer community has already solved this kind of problem with the introduction of build tools.

These tools let you maintain a working environment where your stylesheets are logically separated and well commented, and rules are spaced out and easy to read, maintain, and contribute to. When it is time to ship, the build tools take care of all the grunt work, such as

- Combining separate CSS files into one
- Ensuring that the caching for CSS assets is set correctly so they do not have to be fetched again when you return to the site
- Replacing the notoriously slow `@import` directive with link tags
- Ensuring that your CSS is always placed before any script tags in your document head
- Inlining layout-critical CSS to really turbo-boost the initial rendering of the page

Depending on the particular stack you are working with, the best tools to use will vary, but the key objectives remain the same.

The beauty of working with Sass, or other tools with similar functionality, is that you can break up your stylesheets into as few or as many separate files as you like. When it comes to testing or making the site live, the separate stylesheets will be concatenated, which is another way of saying they will be combined. This reduces the number of time-consuming requests the page is required to make to the server, thereby speeding up your site.

How you break up the stylesheets is a matter of taste, but common wisdom dictates that keeping common variables in one place is a good idea. You can then consider breaking up specific items by theme:

- Typography
- Grid and layout

- Color
- Buttons
- Forms
- Tables
- Reset styles, and so on

You can also add separate stylesheets for CSS needed for components from third-party sources, like a datepicker library or a slideshow plug-in. It usually makes sense to keep these in a specific folder, perhaps called `vendor`, to differentiate them from the in-house styles you developed. This also makes it easier to find that particular piece of CSS when you need to remove or update a third-party resource.

The amount of work involved in adding consistency to your markup and your code can be substantial and perhaps unfeasible within your time constraints. It is, however, something that is worth investing in, because robust, flexible code can save you considerable time and effort in the future.

Publishing Your Site

The process of publishing your site is an area where you can make tremendous improvements with a few lines of code. Several tedious tasks can benefit greatly from a sprinkling of automation.¹⁶

There are some things you should consider adding to your build process:

- Minifying your documents
- Removing comments
- Concatenating stylesheets
- Ensuring that your assets are cached appropriately and perhaps served from an asset server

Forty years ago, Bell Labs developed Make ([https://en.wikipedia.org/wiki/Make_\(software\)](https://en.wikipedia.org/wiki/Make_(software))), a program for the Unix operating system. Still widely used today, Make works by taking instructions from a text file to transform source code files into working software. Most of the build-automation tools in use today, which help you through the process of publishing HTML, CSS, and JavaScript, work on the same principle: they run through a list of tasks, doing what computers do best repetitive, predictable chores.

¹⁶Most of the deployment tools used today can be tied in to the version control system of your choice and are based around a workflow that assumes the existence of version control. If you are not using version control for some reason, this is the time to set that straight. In his pocket book *Version Control with Git*, Ryan Taylor has outlined how to get started with Git, and version control. It is worth a read for the uninitiated (<https://gumroad.com/1rXch>).

The build tools, sometimes referred to as *task runners*, can help with repetitive chores relating to development, testing, optimizing assets, and deploying your site to the server. Here are some of the popular tools you can use to help you automate the most tedious tasks:

- Rake (http://guides.rubyonrails.org/command_line.html#rake) is a Ruby implementation of Make. It is most commonly used to create a list of tasks that needs to be completed in a specific order, much like a recipe. Although it is associated with Ruby on Rails, you can use Rake to help automate any manual build process.
- Grunt (<http://gruntjs.com/>) is another build tool that can help with the chores involved in building a web site. It has its roots in JavaScript and Node.js (<https://nodejs.org/>) development, and you write the list of tasks you want it to run in JavaScript. It has a solid reputation and community around it, meaning most of the tasks you will need are available as plug-ins.
- Gulp (<http://gulpjs.com/>), a challenger to the Grunt throne, has features and requirements very similar to Grunt's. Gulp can run several tasks at the same time, giving it a reputation for speed.
- Brunch (<http://brunch.io/#why>) is a lean, fast, simple tool for the most common build tasks. Like Grunt and Gulp, it uses Node, and its recipes are written in JavaScript.
- CodeKit (<https://incident57.com/codekit/>) differs from the other tools on this list in that it does not require you to use the terminal—it is an app with a nice-looking GUI. Like the other tools on the list, it allows you to compile and optimize your assets, check the quality of your code, and more.

Most of these tasks sport a command-line interface (CLI), meaning you invoke and run them through the terminal. There are many tools on offer, and they all do a good job of relieving you of the type of tasks a computer does much better than any human. If you are not sure which tool is the best one to pick, choose one, try it, and keep on trying until one fits your needs. All the tools on this list will free you to concentrate on more important—and more creative—matters.

There are many things you can do to your writing and publishing processes that can save you time and effort. They all require some initial investment as you investigate the best tools for you and change the ways in which you are used to working. If you are reading this book, time probably is not your most abundant commodity, so take it one step at a time.

Measuring Success After Release

The feeling of accomplishment after you finally get to release your new responsively retrofitted web site is terrific. But even if you managed to make as many changes and improvements as you wanted, work should not stop there. You need to make sure the hard work has paid off in a positive way, so it is important to have defined ways to measure success post-release.

You can do this in simple or in extremely complex ways. Here I give you a few ideas about what you should keep an eye on before and after you publish your new site to get you started, if you haven't done so already.

Once your site is up and running, you want to make sure it keeps running smoothly and that you are alerted when something goes wrong. In addition to monitoring the health of your site—uptime, server capacity, and database connections, for example—you should gather insights on your site's visitors.

One metric you should keep an eye on is the number of users visiting your site on mobile devices. Remember that the numbers after release can be a bit skewed, because previously the user experience of your site on small-screen devices may have been too painful to navigate. But an increase in mobile visits is nonetheless something to celebrate, if not to be expected.

You also want to make sure you are tracking the following before and after release:

- Drop-off rate
- Session duration
- Repeat visits
- Key journeys, such as logins, signups, and checkouts

Each site and organization will have different ways of measuring success. Page views, shares on social media, advertisement revenue, active users, new signups, number of successful checkouts, visit length, number of pages per visit, repeat visits, and even how fast the visits are (the faster the better, on some government web sites, for instance) can all be good indicators of how well, or not, your site is doing.

The most important takeaway is that whatever metric you have determined defines success for your web site should be tracked before, throughout, and most certainly after release. Also consider how visitors to your site can provide feedback, if they wish to. This is particularly important in the first few weeks after release, when the site is new and people are more likely to comment on anything they think may be wrong with it; but it is important to consider doing this in the long term, too. If you make it easy for users to submit feedback and bug reports, your users will provide an important and invaluable layer of testing that you would not be able to replicate on your own.

There are a few relatively simple and, most important, inobtrusive ways in which you can do this:

- Create a form (like a Google Form) to capture visitors' feedback, and add a link to it on your site
- Include a feedback button on your site

- Include a mini-survey directly on your site, with one or two key questions you want to know more about
- Add a link to submit a bug report directly to your issue tracker, ensuring that there are guidelines on how to do so

You can also consider something more sophisticated, like using a product such as Intercom.io (www.intercom.io/; see Figure 4-17), which allows you to engage with your users in a much more targeted and customized manner.

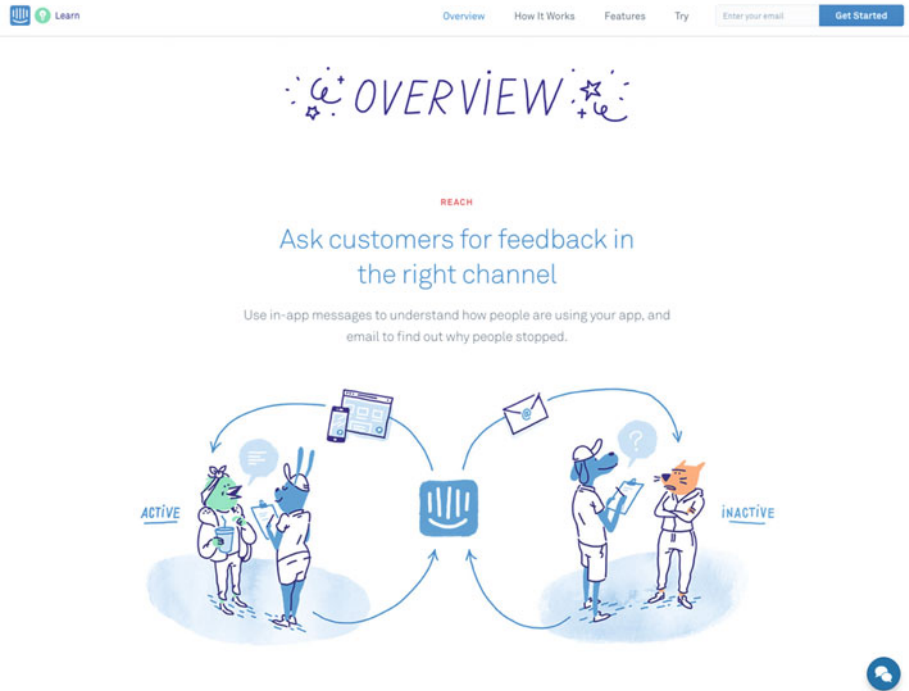


Figure 4-17. Intercom.io, a customer communication platform

Do not let momentum die down. Your team’s schedule is probably fully packed for the weeks and months after the release of your newly responsively retrofitted site. But if you planned subsequent stages into your schedule, and if you have captured all the other things you want to improve in your backlog, you should be able to keep on improving.

The next phases of your project may be much smaller than the first push to go live. You may want to divide the subsequent work into smaller projects that tackle a specific problem or enhancement. And this list of projects is probably never going to be completely finished, because you are sure to see potential for improvement in many areas of your site. But that is the nature of our work.

Summary

Ideas are great, but to have a responsively retrofitted web site, you have to actually build one. There are technical challenges for every project, but the notion of creating a site that adapts to any screen size, any device capabilities, or any other context, from touchscreens to slow connections, can be overwhelming—especially on a site of considerable size—if you have not done so before. Hopefully this chapter has shed some light on how to focus your coding and overcome some of the key challenges and questions that come with converting an existing site to being responsive.

After reading this chapter, you should know the following:

- That you can try new ideas and techniques in smaller projects
- That having a strong focus on accessibility and performance will improve your chances of success
- How to get started with converting your existing fixed-width grid and type into fluid ones
- How media queries work, and how best to tackle them
- Some helpful tips and techniques on how to handle responsive images
- How to go about testing your responsive site on real devices
- What improvements you can make to your process, if time allows
- What to keep in mind when releasing your responsive site
- What baseline metrics to track before and after you release your site

Now that you have covered the plan, content, and design of your responsively retrofitted web site, you have reached the end of the final stage of the project: the build. You are also nearly at the end of this book: join me for some final thoughts in the Conclusion.

Conclusion

The emergence of responsive web design, more than six years ago, showed us the vision for our future sites: a world where users can have great experiences no matter what devices or what screen sizes they have those experiences on. However, it was also clear to us that this change was not going to happen without effort—and a tremendous one, at that.

The beautiful thing about responsive retrofitting projects—and what can also make them look so scary—is the number of improvements that can be validated by the need to make your site’s experience better on any device. As professional web designers and developers, we often know perfectly well what our sites need in order to be greatly improved. But sometimes the task ahead seems too complex, too riddled with obstacles, and too big to feel realistic and achievable.

Our teams are not made up of an infinite number of people who can work on our responsive retrofitting projects day in and day out, leaving everything else on standby. So it is key that any retrofitting project be thoughtfully planned and broken down into digestible chunks that you, your team, and your team’s schedule can comfortably handle. My hope is that this book has made what can often seem like an insurmountable task feel a little more attainable.

The technologies of the Web change quickly. Some of the techniques I have mentioned here will likely be surpassed soon after this book goes to print. But hopefully the core ideas about how to approach planning, content, design, and development will be helpful regardless.

Even though this is where this book ends, this is certainly not the end of your journey into responsive web design. There are many excellent resources that can take you further into understanding how best to design and build a responsive site. You will find some helpful resources in the appendix, and you are bound to also discover others that I have not included in that short list.

I would love to hear about your experiences making existing web sites responsive. What problems were tough to solve? What solutions are you proud of? What was easier than expected? Feel free to find me online, and send me your stories, case studies, blog posts, and conference talks. I await your comments eagerly.

APPENDIX



Resources

The constraints that come with the book format are evident when it comes to a subject as broad as responsive web design: there is so much to say, in such a finite space. But luckily, there are plenty of excellent resources available to further improve your ability to plan, design, and deliver a great responsively retrofitted web site. Peruse this list at your leisure.

Responsive web design and responsive retrofitting in general:

- *Responsive Web Design Podcast*, Karen McGrane and Ethan Marcotte, <http://responsivewebdesign.com/podcast>. Episodes 7, 9, 15, 30, 38, 42, 43, 44, and 53 are especially relevant.
- *Implementing Responsive Design: Building Sites for an Anywhere, Everywhere Web*, Tim Kadlec (New Riders, 2013), www.implementingresponsivedesign.com.
- *Responsive Web Design*, Ethan Marcotte (A Book Apart, 2011), <https://abookapart.com/products/responsive-web-design>.
- *Going Responsive*, Karen McGrane (A Book Apart, 2015), <https://abookapart.com/products/going-responsive>.
- *Responsive Design: Patterns & Principles*, Ethan Marcotte (A Book Apart, 2015), <https://abookapart.com/products/responsive-design-patterns-principles>.

Project management and planning:

- “Effective Project Management: Three Critical Activities,” Laura Dallas Burford, *Nonprofit Technology Network*, www.nten.org/article/effective-project-management-three-critical-activities.
- “Top 11 Project Management Tools for 2016,” Robin Muilwijk, *opensource.org*, <https://opensource.com/business/16/3/top-project-management-tools-2016>.
- “Project Management Apps: Which Is Best for Your Team?” Laura Shin, *Forbes*, www.forbes.com/sites/laurashin/2014/10/21/project-management-apps-which-is-best-for-your-team/#2715e4857a0b1f982d5831d2.

Content strategy:

- “Content Choreography in RWD,” in *Smashing Book 5: Real-Life Responsive Web Design*, Eileen Webb (Smashing Magazine, 2015), 99–126, <https://shop.smashingmagazine.com/products/smashing-book-5-real-life-responsive-web-design>.
- *A Practical Guide to Information Architecture*, Donna Spencer (Five Simple Steps, 2010), <http://uxmastery.com/practical-ia>.
- “Writing for the Web,” *Usability.gov*, www.usability.gov/how-to-and-tools/methods/writing-for-the-web.html.
- “Content Strategy Basics,” *Usability.gov*, www.usability.gov/what-and-why/content-strategy.html.
- “The Elements of Content Strategy,” Erin Kissane (A Book Apart, 2011), <https://abookapart.com/products/the-elements-of-content-strategy>.
- “The Discipline of Content Strategy,” Kristina Halvorson, *A List Apart*, <http://alistapart.com/article/thedisciplineofcontentstrategy>.
- “Content Strategy for Mobile,” Karen McGrane (A Book Apart, 2012), <http://abookapart.com/products/content-strategy-for-mobile>.
- “Aligning Content Work with Agile Processes,” Brendan Murray, *A List Apart*, <http://alistapart.com/article/aligning-content-work-with-agile-processes>.
- “Why and How to Perform a Content Audit,” Laura Rives, *Hannon Hill*, www.hannonhill.com/resources/blog/2016/why-and-how-to-perform-a-content-audit.html.
- “How to Quickly Create a Written Style Guide for Your Company,” Emily Nix, *Nectafy*, <http://nectafy.com/written-style-guide/>.

User research and testing:

- “User Research Basics,” *Usability.gov*, www.usability.gov/what-and-why/user-research.html.
- “User-Centered Design Process Map,” *Usability.gov*, www.usability.gov/how-to-and-tools/resources/ucd-map.html.
- “Guerrilla Testing: Getting Input into Products and Services,” Government Service Design Manual, *Gov.uk*, www.gov.uk/service-manual/user-centred-design/user-research/guerrilla-testing.html.

Accessibility:

- “Accessibility Basics,” *Usability.gov*, www.usability.gov/what-and-why/accessibility.html.
- “Reframing Accessibility for the Web,” Anne Gibson, *A List Apart*, <http://alistapart.com/article/reframing-accessibility-for-the-web>.
- “All Technology Is Assistive,” Sara Hendren, *Backchannel*, <https://backchannel.com/all-technology-is-assistive-ac9f7183c8cd#.is88svol2>.
- “What’s the Difference Between Usability and Accessibility?” Liam McDermott, *A Padded Cell*, www.apaddedcell.com/what-s-the-difference-between-usability-and-accessibility.

CSS units:

- “The Lengths of CSS,” Chris Coyier, *CSS-Tricks*, <https://css-tricks.com/the-lengths-of-css>.
- “<length>,” *Mozilla Developer Network*, <https://developer.mozilla.org/en/docs/Web/CSS/length>.
- “Font Sizing with rem,” Jonathan Snook, http://snook.ca/archives/html_and_css/font-size-with-rem.

Media queries:

- “Using Media Queries,” *Mozilla Developer Network*, https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries.
- “Media Queries Level 4,” *W3C*, <https://drafts.csswg.org/mediaqueries>.
- “@media,” *Mozilla Developer Network*, <https://developer.mozilla.org/en-US/docs/Web/CSS/@media>.

Responsive images:

- Responsive Images Community Group, <http://responsiveimages.org>.
- “Responsive Images: If You’re Just Changing Resolutions, Use srcset,” Chris Coyier, *CSS-Tricks*, <https://css-tricks.com/responsive-images-youre-just-changing-resolutions-use-srcset/>.
- “The Anatomy of Responsive Images,” Jake Archibald, <https://jakearchibald.com/2015/anatomy-of-responsive-images>.

Performance:

- “Setting a Performance Budget,” Tim Kadlec, <https://timkadlec.com/2013/01/setting-a-performance-budget>.
- “How to Make a Performance Budget,” Daniel Mall, <http://danielmall.com/articles/how-to-make-a-performance-budget>.
- “What Your Site Costs Users,” Tim Kadlec, <https://timkadlec.com/2015/03/what-your-site-costs>.
- “Render Blocking CSS,” Ilya Grigorik, *Google Developers*, <https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-blocking-css>.
- “More Weight Doesn’t Mean More Wait,” Scott Jehl, *Filament Group*, www.filamentgroup.com/lab/weight-wait.html.
- “Inlining Critical CSS for First-Time Visits,” Jeremy Keith, <https://adactio.com/journal/8504>.
- “Media Query & Asset Downloading Results,” Tim Kadlec, <https://timkadlec.com/2012/04/media-query-asset-downloading-results>.
- “The Perception of Performance,” Luis Vieira, *SitePoint*, www.sitepoint.com/the-perception-of-performance.
- “An Introduction to Perceived Performance,” Matt West, <http://blog.teamtreehouse.com/perceived-performance>.

Measuring performance:

- Google PageSpeed tools, <https://developers.google.com/speed/pagespeed>.
- gzipWTF, <http://gzipwtf.com>.
- Pingdom Website Speed Test, <https://tools.pingdom.com>.
- ImageOptim, <https://imageoptim.com>.

Measuring success, analytics, and data gathering:

- “Web Analytics Basics,” *Usability.gov*, www.usability.gov/what-and-why/web-analytics.html.
- Mixpanel mobile analytics, <https://mixpanel.com>.
- Segment analytics API and customer data hub, <https://segment.com>.
- Kissmetrics customer intelligence and web analytics, www.kissmetrics.com.
- Intercom customer communication platform, www.intercom.io.

Other resources:

- *Style Guide Podcast*, Anna Debenham and Brad Frost, <http://styleguides.io/podcast>.
- “HTML5 Cross Browser Polyfills,” Modernizr, <https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills>.
- “Responsive Resources,” Brad Frost, <http://bradfrost.github.io/this-is-responsive/resources.html>.

Index

■ A

- Absolute units, 72
- Accessibility, 25, 27, 94–95
 - responsive web design, 52–53
- Accordion, 38
- A List Apart’s pattern library, 61
- Amazon, 54
- Anne Gibson’s alphabet
 - of accessibility, 26
- Argos, 39
- Asana tool, 7
- Axure tool, 70

■ B

- Basecamp tool, 6–7, 21
- BBC News web site, 85
- BBC’s home page redesign, 48
- Bootstrap, front-end framework, 50–51
- Boston Globe, xxv
- Breakpoints, responsive web design, 59
- BrowserStack, 119
- Building strategies
 - accessibility, 94–95
 - experiment, smaller projects, 94
 - inobtrusive ways, 134–135
 - Intercom.io, 135
 - markup and code style guide, 126–127
 - metric, 134
 - mixins and functions, 129, 131
 - performance
 - budget, 96–98
 - handy tools, 100
 - lean sites, 99
 - mobile user, 96
 - perceived performance, 98–99

- responsive retrofitting project, 96
- site cost, 102–103
- Speed Index, 101–102
- visually responsive, 96
- web fonts, 100
- WebPagetest, 100–101
- publishing, site, 132–133
- Sass, 127–129
- site and organization, 134
- small-screen devices, 134
- stylesheets, 131–132
- testing
 - browsers and
 - operating systems, 119–120
 - BrowserStack, 119, 122–123
 - decision-making process, 118
 - definition, 119
 - device lab, 121
 - document and process, 119
 - Ghostlab, 124
 - Percy, 124–125
 - web site and prototypes, 118
 - writing code and markup, 126

■ C

- Canonical.com’s first responsive test, 58
- Chicago Manual of Style, 43
- 12-Column grid layout,
 - content boxes in, 73
- Command-line interface (CLI), 133
- Component inventory folders, 62
- Consultants, project, 3
- Content
 - accessibility and usability, 24, 27
 - accordion, 38
 - audit, 30

Content (*cont*)

- Canonical.com, 38
- column headings, 29
- content inventory, 28
- creation and management, 23
- duplicated work, 32
- ongoing maintenance, 32
- real, 23–24
- simplification, 40
- style guide, 41
- tabs, 39
- updates and improvements, 32–33
- content audit, 28, 30
- Content boxes
 - adjustments to narrower
 - viewport adapt to, 75
 - in 12-column grid layout, 73
 - in screen one-third smaller, 74
- Content inventory, 28
- Content management system, 23
- Content specialist, 47
- Content strategy, 40–41
 - mobile, 31, 45
- CSS, xxii–xxiii, xxv, 59
- CSS Object Model (CSSOM), 113

■ **D**

- Dated content, 36
- Deadlines, responsive
 - retrofit project, 17–18
- Designers, project, 16–17
- Designers, web site, xxii
- Design-pattern library, 44
- Design stage, responsive web design
 - on accessibility, 52–53
 - determining breakpoints, 59
 - analytics, 60
 - in browser, 59
 - elements to analyze, 60
 - tweakpoints and breakpoints, 59
- evolution, 47–48
- feedback and reviews, 91
- flats, 54–55
- grids and type, 72
 - explore responsive grids, 73–74, 76
 - grid to percentages, 72
 - reorder content, 76, 78
 - typographic scale, 78–79
- handling images, 79
 - bytes, 81

- make image inventory, 79–80
 - SVG, 81
- performance, 54
- on reusability, 49–50, 52
- setting rules, 55
 - building prototype, 58
 - writing ideas, 55–57
- standardize across sites, 67
- style guide, 60–61
 - building style guide, 65–66
 - clean up style sheets, 66
 - examining, 66
 - pattern library, 66
 - responsively rationalize, 64
 - screenshots, 62–63
- useful patterns
 - dropdown and slidedown, 82
 - navigation, 82
 - overflow, 86–87
 - priority+, 84
 - priority columns, 90
 - side drawer, 84
 - tables, 87
- UX, 68
 - prototyping, 68–69
 - sticky-note-sized wireframes, 68
 - testing prototype, 70–71
 - things to test, 71
- Developer, front-end, 2–3
- Developers, project, 16–17
- Digital record, low-fidelity planning, 6
- Document object model (DOM), 113

■ **E**

- eBay’s November 2000 web site, xxiii
- Emergence of responsive web design, 137
- European Organization for Nuclear
 - Research (CERN),
 - web site for, xxii
- Evernote tool, 8
- Evolution, responsive web design, 47–48

■ **F**

- Fixed-width site, xxi, xxiv, xxvi, 17
 - creating style guide for, 64
- Flat mockups, 55
- Fluid grids
 - absolute units, 109–110
 - bootstrap, 107

- fast track, 104
- markup, 104–106
- remove inline styles, 106
- scalable type, 110
- ZURB, 108

Folders, component inventory, 62

Front-end developer, 2

■ **G**

Ghostlab, 124

Going Responsive, 19

Google's material design spec, 51

Grids and type, responsive web design, 72

- explore responsive grids, 73–74, 76
- grid to percentages, 72
- reorder content, 76, 78
- typographic scale, 78–79

Guardian and Observer Style Guide, 43

Guerrilla testing, 70–71

■ **H**

“How to Make a Performance Budget” 54

HTML, xxv

HTML5 Boilerplate framework, 49

■ **I, J, K**

iMac, 68

Informal testing, prototype, 70

InVision tool, 70

■ **L**

Lea Verou's contrast ratio tool, 53

■ **M**

MailChimp's acclaimed

- Content Style Guide, 42

MailChimp's style guide, 42

Media queries

- advantage, 113
- breakpoint, 113
- browser support, 114
- CSS rules, 112
- float properties, 112
- linear, 112
- print and speech, 111
- stylesheet, 113

- technical tenets, 110
 - W3C specification, 111
- Microsoft, dropdown navigation style, 83
- Mobile view of Wikipedia, 77

■ **N**

Navigation pattern, responsive web

- design, 82

New York Times uses dropdown

- navigation, 83

■ **O**

One-hour test, responsive

- retrofit project, 15

Online banking project, 19

Open source tools, 6–8

■ **P**

Paper prototype, 69, 71

Pattern Lab tool, 65

Percy, 124–125

Performance, responsive web design, 54

Photoshop, 55

Pixels, 72

Priority columns responsive pattern, 90

Priority+ navigation pattern, 84

Project leader, 3–4

- role of, 12, 20

Project-management tools, open source, 6

Prototyping, super-speedy, 68–69

■ **Q**

[Q]uick fix remedies

■ **R**

Redesigns, responsive, 48

- BBC's home page, 48

Responsive images

- compress, bitmaps, 118
- image caching, 117
- optional images, 118
- picture and srcset, 115–117
- SVG images, 114–115

Responsive retrofit project, 44

- account for downtime, 12–13
- assigning tasks, 10

Responsive retrofit project (*cont*)
 define project leader, 3–4
 deprioritize other projects, 10
 do not use everyone at once, 12
 finding time in schedule, 9
 goal of, 68
 involve right people, 2–3
 keeping record, 20–21
 keep tight scope, 13
 break down tasks, 14–15
 define stages, 16
 determining out of scope, 16–17
 one-hour test, 15
 organize, 14
 prioritize, 14
 start with wish list, 13
 testing, 17
 low-fidelity planning, 5
 digital record, 6–8
 gaining perspective, 5
 managing site updates, 20
 participation, 4
 rollout strategies, 18–19
 set deadlines, 17–18
 team, 2
 understand calendar, 9
 use everyone at once, 11
Responsive retrofitting project, 104, 137
Responsive web design, xxi
 emergence of, 137
 enter, xxiv
 time, xxvi
 web used to be responsive, xxi–xxiii,
Reusability, responsive
 web design, 49–50, 52
Rewriting content, 34

■ S

Salesforce's Lightning Design System, 61
Scalable vector graphics (SVG), 81
Scope creep, 17
Screenshots, responsive
 web design, 62–63
Senior designer, 3
Side drawer navigation, 84
Sketch tool, 55
Slack messaging app, 20–21

Sony Android, 68
Sound content strategy, 41
Standards-based HTML, xxiii
Sticky-note-sized wireframes, 68
Style guide, responsive web design, 60–61
 building, 65–66
 clean up style sheets, 66
 examining, 66
 or pattern library, 66
 responsively rationalize, 64
 screenshots, 62–63

■ T

Tables, responsive web design, 87
 jQuery Mobile, 89–90
 to list, 88
 Wikipedia entries, tables, 88
Task-management software, 7
Testers, project, 3
Testing, project, 17
The Elements of Content Strategy, 37
Tools
 Asana, 7
 Axure, 70
 Basecamp, 3, 6–7, 21
 Evernote, 8
 InVision, 70
 Lea Verou's contrast ratio, 53
 Pattern Lab, 65
 Sketch, 55
 Slack, 20–21
 Trello, 8–9
Topshop.com, side drawer, 84
Trello tool, 8–9
Typographic scale, 78–79

■ U

ubuntu.com, xxvi, 4
 component inventory
 uncovered variations on, 64
Usability testing, 71
User experience designer, 2–3

■ V

Visual designer, 47

■ **W, X, Y, Z**

Web design

- professionals, xxvi
- technical factors, xxiv

Web site

- designers, xxii
- eBay's November xxiii, 2000

- for European Organization for Nuclear Research (CERN), xxii
- responsive redesigns of, xxv

Web Standards Project's

- mission, xxiv

Wikipedia,

- mobile view of, 77

Words—no mockups, 55