LUKKIEN

Advisory

Responsive web design

Authors: Nectaria Dimitropoulos & Jeroen van Rijn
Date: June 5, 2012
Version: 1.0

Prologue

This report contains advice as to how Lukkien could integrate responsive web design in projects that require this new approach. Lukkien will be able to decide when to use responsive web design and to manage this new process. This advisory also provides a workflow and guidelines for web designers and developers to design and develop a responsive website. It shows the mapping of the different stages within the process of developing a responsive website. The content of this report is gained from the research done for responsive web design. Based on this advisory web developers and web designers will be able to apply responsive web design in new projects.

Summary

Below are the main topics, with a brief explanation, which are reviewed in this document. Based on this information an advice is written on how responsive webdesign will affect the workflow and which choices Lukkien has to make.

RESPONSIVE WEB DESIGN
A new technique on web design which allows us to create one single website to deliver a great experience on almost any device. Whether the user is viewing your website on a smartphone, tablet or desktop, the website will automatically adjust to the user's device resolution, thus resulting in a proper user experience.

ADAPTIVE WEB DESIGN
A technique on web design which allows us to create interfaces that adapt to the user's capabilities (in terms of both form and function). It's a bit similar to responsive web design.

MOBILE FIRST
An approach for web development which starts with mobile and expands to more capable platforms afterwards. This forces us to focus on our products by embracing the constraints inherent to mobile design, thus simplifying the design and development process. Mobile first puts the 'workload' on the larger (desktop for instance) and more capable platforms instead of the weakest platform (mobile devices). The capable devices fill up the mobile 'baseline', instead of the full version of the site being the baseline. This prevents a 'weak' device from having to try to make a mobile version of it in terms of code. Mobile first can be well combined with responsive web design.

PROGRESSIVE ENHANCEMENT VERSUS GRACEFUL DEGRADATION
Two different approaches on developing websites. With progressive enhancement a website gets better as the device and the browser get more capable. Graceful degradation is the opposite: a website decreases as the device and the browser become less capable. Progressive enhancement may be a better approach for responsive web design.
Mobile first is actually a form of progressive enhancement.

DESIGN GUIDELINES
A few guidelines on designing a responsive website regarding a Photoshop grid, wireframes and visual designs.

TECHNICAL GUIDELINES
A few guidelines on creating a responsive website with a flexible grid, flexible media and media queries.

TESTING
A few examples and suggestions on how to test a responsive website. With an application, an online website/web tool or the device wall (of the interactive department).

Responsive web design

Definition

According to the founder, Ethan Marcotte, responsive web design is:
A flexible grid (with flexible images) that incorporates media queries to create a responsive, adaptive layout.
Speaking purely in terms of front-end layout, it takes three core ingredients:
1. A flexible, grid-based layout,
2. Flexible images and media, and
3. Media queries, a module from the CSS3 specification.

Another definition:
Responsive web design is flexible, device-independent design for the web and a context-sensitive layout that is aware of its own size.

Pros and cons

Here are some of the pros and cons of responsive web design:

Pros

- One HTML page, which means less maintenance. Instead of having to update two pages with the same content, you only have to update one universal page.
- CSS code styles the same elements. Chances are, if you're making a mobile site, you'll use slightly different tags, IDs, and classes than you would when creating a desktop page. With media queries, you're styling the same tags. You don't need to remember which tags you used on the mobile site, and which ones you used on the desktop site; it's all the same. Because of that, it will be easier to make edits to the CSS of both the main page and the mobile page.
- It isn't just about what device the user is using. Not everyone always has their browser full-screen. A lot of the time, it is much smaller. With responsive design, you'll never get a horizontal scrollbar.

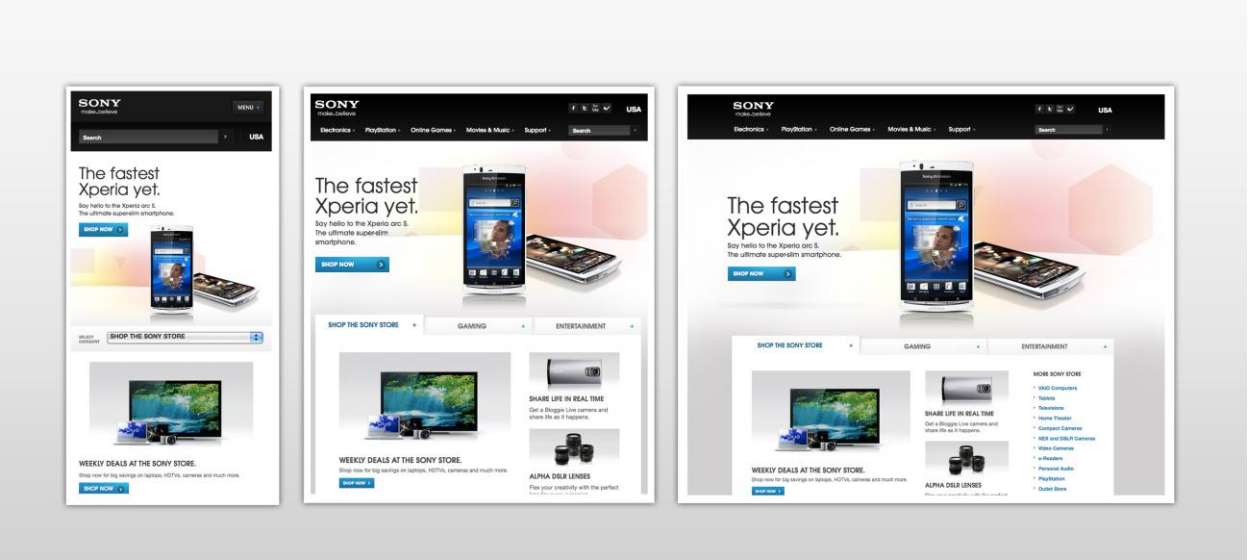
Cons

- Loading time is an issue. A user with an iPhone would load your site and get all of the code behind it. However, a lot of the things you have on the desktop site are set to "display: none" with the mobile media query. In other words, the user loads unnecessary content.
- Cross browser compatibility. Internet Explorer 8 and below can't work with media queries, and an unfortunate amount of people are still using these outdated browsers. A lot of people are also using old versions of Firefox.

Best practices

Below are some responsive websites which could provide useful information and even some inspiration:
http://css-tricks.com
http://foodsense.is/
http://seesparkbox.com/
http://newyorkbicycles.myshopify.com/collections/frontpage/products/bicicletta-uno
http://www.macdonaldhotels.co.uk/
http://us.illyissimo.com/
http://mediaqueri.es/

Sony, an example of a responsive website on different screen resolutions:



Adaptive web design

Adaptive web design is also an approach for when a client wants different devices supported, for instance mobile. This is mainly the case when the client's resources or wishes are inadequate for responsive web design.
Opinions and definitions of adaptive web design opposed to responsive web design are a bit divided as shown below:

Responsive versus adaptive webdesign

"Adaptive web design" refers more to the secondary and less fluid approach of adapting existing web designs, or designing for controlled adaptation as opposed to a truly fluid and flexible "responsive" design."

"Adaptive layout," a practice that combines the benefits of fixed-width design with the realities of multiple screen sizes, is no longer an alternative to responsive design; instead, it becomes a form of responsive design, albeit a less robust one than the fully responsive (fluid) method Ethan describes in his book.
Our understanding of "responsive design" should be broadened to cover any approach that delivers elegant visual experiences regardless of the size of the user's display and the limitations or capabilities of the device.

According to Aaron Gustafson, "Responsive web design," as coined by Ethan Marcotte, means "fluid grids, fluid images/media & media queries." "Adaptive web design," is about creating interfaces that adapt to the user's capabilities (in terms of both form and function). "Adaptive web design" is just another term for "progressive enhancement" of which responsive web design can (an often should) be an integral part, but is a more holistic approach to web design in that it also takes into account varying levels of markup, CSS, JavaScript and assistive technology support.
For the record, it's important to draw a distinction between "adaptive web design" and "adaptive layouts" because "adaptive layouts" implies only the use of media queries, which may not be done in a progressively enhanced way. Adaptive layouts achieved in a mobile-first manner, however, are very likely progressive enhancement and, thereby, a means of "adaptive web design."

According to Brendan Falkowski, adaptive websites use media queries to adjust the layout at specific breakpoints, but aren't necessarily 100% fluid in between those breakpoints.

So the main difference between responsive and adaptive web design is that responsive web design is entirely fluid whereas adaptive web design is less fluid and less robust.

Progressive enhancement versus graceful degradation

Progressive enhancement and graceful degradation are two different approaches on developing websites, two technical starting points. With progressive enhancement a website gets better as the device and the browser get more capable. Graceful degradation is the opposite: a website decreases as the device and the browser become less capable.

Graceful degradation

Graceful degradation simply means “display a basic and simple version if the full layout can not be rendered”. In short, the target was to always make sure that designs would degrade gracefully if displayed on an older web browser. The code structure would be built in a way that kept a text-only, no-CSS version of the HTML output logical and well-structured.

Progressive enhancement: mobile first

Where graceful degradation was all about building the full layout first and making it usable on mobile devices, progressive enhancement turns the order around and builds from another perspective. A designer building with the progressive enhancement philosophy starts with creating the design and layout for small-screen devices and, once the mobile-friendly version is complete, moves on to adding more advanced layouts and design elements that are displayed on devices with higher resolutions and bigger screens. Which makes mobile first a form of progressive enhancement.

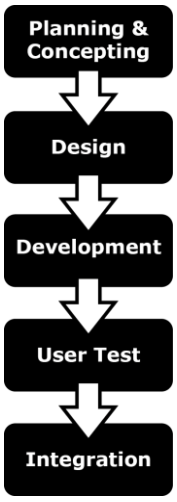
Responsive designs scale down if you reduce the width of your web browser, most often by switching to a simpler layout that is better suited for small-screen devices. In more advanced designs, there could be several layouts included – one for each pre-defined screen width.

A mobile first responsive web design comes down to using progressive enhancement as a foundation for web strategy and design. Designing with progressive enhancement involves smartly adding layers of enhancements to a strong foundation in order to deliver an accessible (and hopefully optimized) experience to all.

When building a website you should consider those two approaches and which one is more appropriate to use in that case. In case of a responsive website it’s preferable to choose a progressive enhancement approach because you start with the most simple layout. Combining this approach with mobile first thus allows us to create a great responsive website.

The way of working

Current way of working (traditional way)

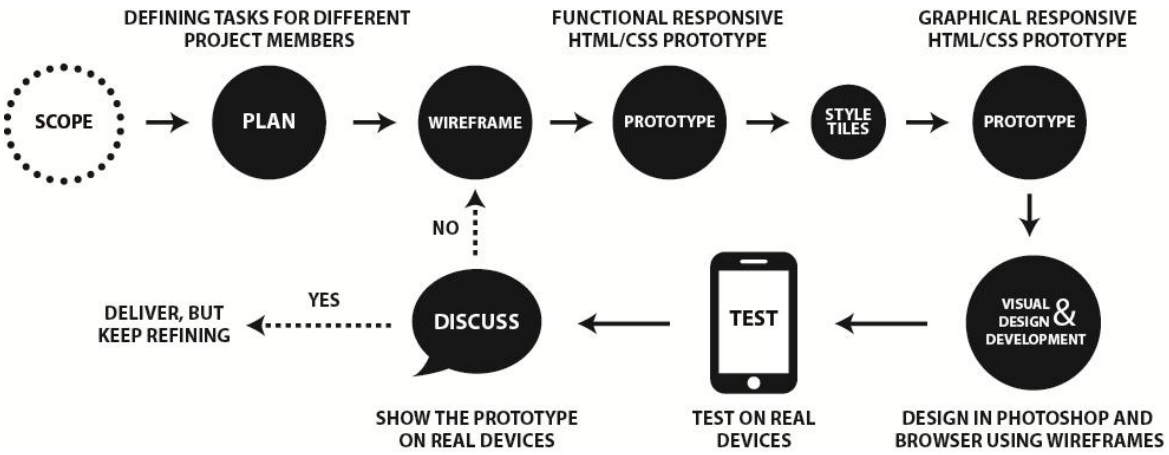


Developing a website, according to the waterfall model is usually divided into separate development phases, dividing the work of the project into task-based phases. First the flowchart(s) and the wireframes are being created, in which the functional scope is defined. These are presented to the client for approval through a number of feedback rounds. Then after the approval the design process starts: all the different page templates are designed (i.e. homepage, content page, FAQ page, etc). These designs are then presented to the client for approval. Then the development process starts.

The new way of working (responsive way)

One of the issues of the traditional way of working is the delivery of design 'comps' to a client. With responsive web design this simply doesn't work, because of the high number of screen resolutions it is impossible to create designs for every one of them.

After the *scope* has been approved by the client and the tasks for the different project members have been defined, the deliverables can be created.
The *flowchart(s)* can be made according the traditional way, because they don't contain layout technical things (it's derived from the sitemap). During the *wireframing* process the position of the content elements on different devices is being defined. Then the *functional prototyping* starts: a (basic) grid is defined (global position of (content) blocks, basic elements) based on the (global) wireframes, made by the designer. Meanwhile the designer adds more details to the global wireframes and creates *style tiles*. Then the *graphical prototyping* starts: all design elements from the style tiles are inserted in the fluid grid, so the prototypes become more detailed. The result is a responsive website which is in fact 'bare', but it shows how the elements move at different screen resolutions. The designer then creates a set of sample web pages (templates) in different resolutions, such as homepage, product page, contact page etc. This will be typical *visual designs* just like in the traditional way of working. Meanwhile the developer continues on building the website based on the prototypes and the few visual designs the designer created. Finally the website will be tested on various (mobile) devices.



The client gives his approval three times during the project:
- On the scope and flowchart(s)
- After both halves of the prototyping process (functional and graphic)
- On the sample designs (and partly the development progress)

Scrum
The best possible way for developing a responsive website is to pull all members of a team together to design and build in multiple phases. There should always be ongoing communication between the web designer and the web developer to answer any issues about what's possible with elements visually.

How does responsive web design affect the current web development method, considering:

- *Defining the breakpoints*
  In the traditional way of working the web designer choses a screen resolution for the (desktop) website, based on the project's requirements and the client's wishes. A responsive website however will be viewed on more than one device, so multiple screen resolutions are chosen. Based on the devices and their resolutions that will be supported, the web designer and web developer define breakpoints together.
- *Functional design*
  Depending on the number of devices that will be supported various wireframes will be created, starting with the smallest supported device (mobile first approach).
  The designer first creates global wireframes with the content elements only. After the functional prototyping he adds more details to these wireframes, resulting in detailed graphical wireframes, which the developer uses during the graphical prototyping.
- *Graphic design*
  A design mock-up is no longer a single ".psd" file as it was previously the case, instead style tiles will be used. Style tiles are a good way to design the look-and-feel and it's possible to create them in a short period of time. In addition, the most important pages are being designed in Photoshop. Based on the style tiles and the designs of the most important pages, the remaining screens can be developed.
- *Development*
  The great difference between the responsive and the traditional way of working is the presence of a developer during the prototyping process, which thus moves a significant part of the technical process (in the traditional way) to the design process. This is an advantage which is similar to the Scrum project methodology: decide and build together simultaneously instead of a waterfall process. Moreover the client can also provide feedback on a working (basic) version of the website, which wasn't the case previously.
- *Testing*
  A responsive website will be tested on various (mobile) devices. It's important because this way the content will be displayed optimal on each device.
- *Client*
  Instead of giving feedback on static mock-ups, the client gives his feedback on style tiles, prototypes and the most important visual designs.
- *Communication*
  Constant communication is required when approaching a responsive website. The project will go smoother if both designer and developer discuss problems and solutions as soon as they turn up.

Mobile First

Luke Wroblewski created a philosophy called 'Mobile First' that highlights the need to prioritize the mobile context when creating user experiences. Starting with mobile first:

- Prepares you for the explosive growth and new opportunities emerging on mobile today.
- Allows websites to reach more people (a large part of the world's population has a mobile device equipped with a browser)
- Forces designers and developers to focus on core content and functionality
- Lets designers innovate and take advantage of new capabilities and new technologies native to mobile devices (such as geolocation, touch events and more)

The time is now

Fueled by capable devices and faster networks, mobile internet usage is exploding. Building mobile first not only positions you to take advantage of this growth, it also opens up new opportunities for engaging your customers.

This isn't just an opportunity to create a mobile version of your web product; it's an opportunity to provide an improved overall experience for your customers.

Embracing constraints

Mobile comes with a natural set of constraints that may at first seem limiting. Screens are small, connections are slow, and people often only give you their partial attention or short bursts of their time. Designing for mobile first forces you to embrace these constraints to develop an elegant mobile-appropriate solution. But the benefits go well beyond mobile.

Small screen sizes force you to prioritize what really matters to your customers and business. There simply isn't room for anything else. Slow connections and limited data plans require you to be vigilant about performance, resulting in fast loading websites everywhere.

Location and time act as constraints on the mobile design process because they force you to think differently about how people will use your products throughout their day. They also create new opportunities for engagement that can help you innovate.

Laying out the land

As the mobile landscape continues to change, we have to be ready with layouts that adapt to the task at hand—from big differences between device experiences to filling in the small
gaps between device screen sizes and orientations.

- Accept and embrace the rate of change in mobile isn't going to change anytime soon.
- Use the meta viewport tag to let mobile browsers know you've thought of them in your designs.
- Account for differences in screen density by making higher resolution images available to the devices that support them.
- Adapt to device variations with responsive and fluid layouts.
- Account for the differences between device experiences in your responsive web design or server-side solutions.
- Reduce complexity for yourself and your customers by cutting down to the minimum amount of functionality necessary for your website or application.

So starting with mobile forces us to focus on the most important content and functionality, thus paying attention to the less 'capable' and small devices, which can result in better and fast loading websites.

Advice to Lukkien's clients

Which method should be advised to a client, responsive web design, adaptive web design or traditional web design, when keeping budget and the amount of the must-be supported devices in mind?

This choice between those methods depends on the client's wishes and the website's requirements. Based on the client's intention, Lukkien should advice responsive or adaptive web design. For instance when there are just a few devices that need to be supported or the client has a low budget adaptive design can be advised. But when a wide range of devices will be supported responsive web design should be advised. If however the client just wants a desktop website and maybe in the (near) future a mobile version, then traditional web design can be used. When a responsive approach is chosen by the client, you need to persuade him of the benefits and communicate that it will for example add more time to the project.

Reasons to consider responsive web design

- **One solution to rule them all** - Having a single version of a website that works on every screen width and device is a pretty great deal. It will not only be beneficial for the users, it will also be easier to maintain afterwards.
- **Avoiding device-alternating issues** - One of the main downfalls of building both, a desktop version and a mobile version of a site, is the fact either way the user is likely to share the URL, and the person who opens that shared link isn't always using the same device through which it was shared. Responsive web design avoids this effectively.
- **It's not just about the device** - Even if we have a very large screen, we don't always have a full-screen sized window open for the web browser. With a responsive web site, even if the window is resized, we never end up with that horrible horizontal scroll.
- **Keeping costs low** - Solid responsive solutions require additional design and front-end development time, but doesn't too heavily impact application development. It could take around 20-30 per cent longer to develop a responsive site, but it's still faster than creating an additional mobile site or app. Developing a site this way also means that you only need to develop, manage, and maintain the one site, so it can reduce these costs too.
- **A website that works even when new devices are released** - A mobile site needs to be able to recognize the user's device; when new devices are released, the site needs to be updated. As the responsive solution only recognizes the browser's width, no new updates would need to be made. This means it's much more future-proof and scalable.

Costs and benefits

The changes required in layout (and, potentially, the functional components that are used to present content) for a responsive website means there will be a need for multiple instances of wireframing, prototyping and testing phases for each different browser type, potentially lengthening the project timeline. However, the consistency offered by having one web presence rendering across all devices rather than a separate mobile application presents a major benefit in user experience. The explosion of content-receiving devices is likely to continue and, as such, responsive site design will grow into each new device without the need for a complete redesign.
Moreover a responsive website will need less maintenance, which may reduce long-term costs for a corporate site for instance.

Presenting a responsive website: the client's involvement

Since responsive web design is a new process/workflow for Lukkien as well as for her clients, it is of great importance to involve the client on an early stage for a successful progress of the project. For a client a responsive approach is about being flexible in the process. If they only see the wireframes and the visual designs, there is a danger that they are surprised by the outcome of the website when seen in a browser. So during a responsive design project it can be very useful to involve the client already in the design process. The client can give his feedback on the wireframes, the style tiles and the prototypes. Thus, his feedback can be processed immediately and the upgraded versions can be presented to the client. And because he sees a working version of the website and provides feedback on it, he won't be surprised. If this process is repeated frequently during the course of the project this may result in a better product, which will meet the needs and requirements of the client.

Content priority list

Making a priority list will provide a good impression about the most important content items on different devices. It will also help making important decisions about what content to emphasize and what content could be removed.

Before the wireframing process starts, ask the client to sort the content on a page into a hierarchy list by priority, with the most important items appearing at the top of the list. Let the client create lists of content items for mobile, tablet and desktop. The wireframes will be based on the outcome of this list.

This is an example of two priority lists:

Content items for desktop:

1. Title
2. Contact Us Call-to-Action
3. 4 Work Samples
4. Primary Navigation
5. About Us
6. Contact information
7. Services list
8. Social Media links
9. Latest Blog Entry
10. Copyright Information

Content items for mobile:

1. Title
2. Contact Information
3. Primary Navigation
4. 1 Work Sample
5. Social Media links
6. Copyright Information
7. Contact Us Call-to-Action
8. Latest Blog Entry

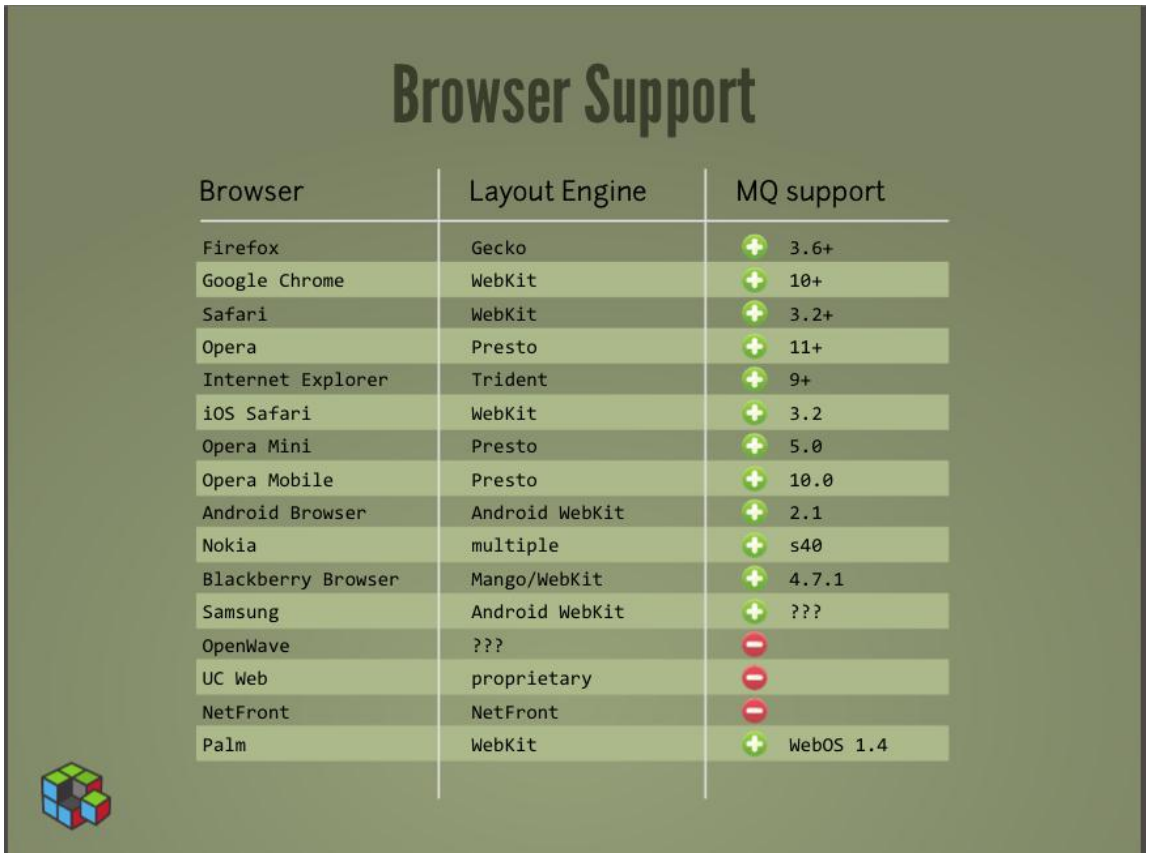
Browser and device support

Before the design and development processes start, it is necessary to do a quick research about statistics on browser and device support.
According to Ethan Marcotte, it should be noted that media queries enjoy incredibly robust support among modern browsers. Desktop browsers such as Safari 3+, Chrome, Firefox 3.5+, and Opera 7+ all natively parse media queries, as do more recent mobile browsers such as Opera Mobile and mobile WebKit. Of course, older versions of those desktop browsers don't support media queries. And while Microsoft has committed to media query support in IE9, Internet Explorer currently doesn't offer a native implementation.

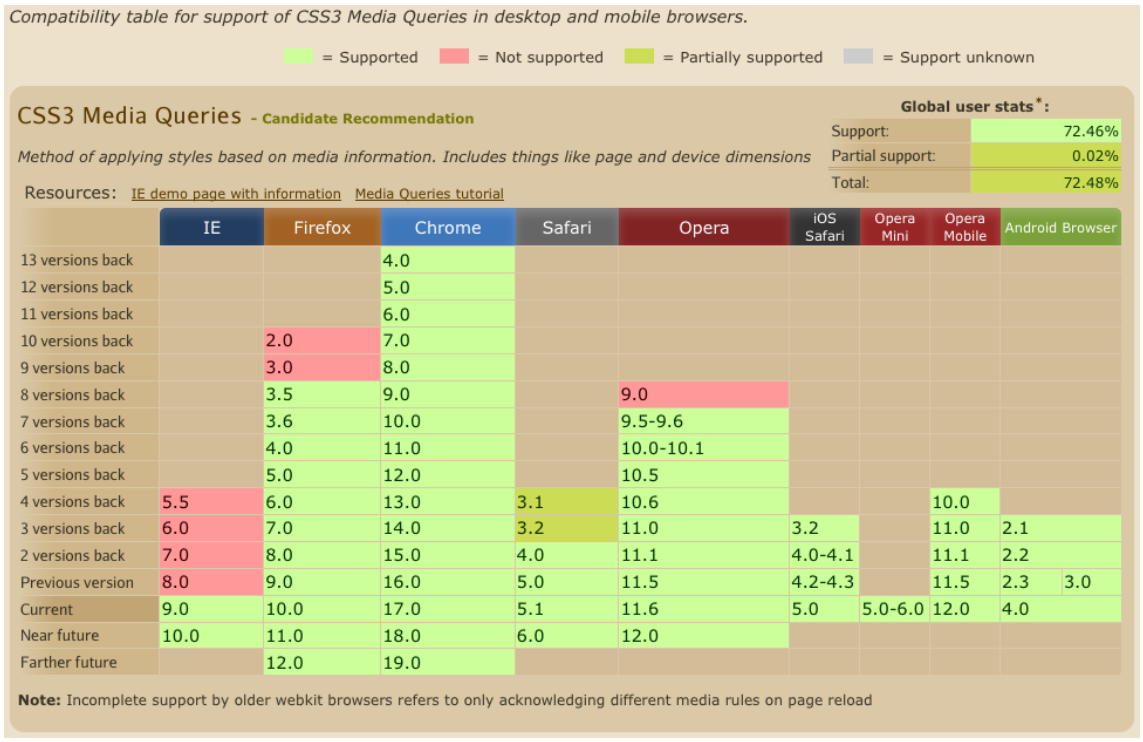
Websites on statistics

The images and table shown below are however merely a snapshot because of the ever changing landscape of Internet. So in order to get a good picture of which browsers and devices are supporting responsive web design statistics about browser and device support should be checked again before a responsive webdesign project begins.
To few more recent statistics on browser and device support, you can check this website:
http://caniuse.com/
and on browser and device use, you can check this website:
http://gs.statcounter.com/

Browser support



Desktop and mobile browser support of CSS3 media queries.



Compatibility table for support of CSS3 Media Queries in desktop and mobile browsers.

= Supported = Not supported = Partially supported = Support unknown

CSS3 Media Queries - Candidate Recommendation

Method of applying styles based on media information. Includes things like page and device dimensions

Resources: IE demo page with information Media Queries tutorial

Global user stats*:

Support:	72.46%
Partial support:	0.02%
Total:	72.48%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser	
13 versions back			4.0							
12 versions back			5.0							
11 versions back			6.0							
10 versions back		2.0	7.0							
9 versions back		3.0	8.0							
8 versions back		3.5	9.0		9.0					
7 versions back		3.6	10.0		9.5-9.6					
6 versions back		4.0	11.0		10.0-10.1					
5 versions back		5.0	12.0		10.5					
4 versions back	5.5	6.0	13.0	3.1	10.6			10.0		
3 versions back	6.0	7.0	14.0	3.2	11.0	3.2		11.0	2.1	
2 versions back	7.0	8.0	15.0	4.0	11.1	4.0-4.1		11.1	2.2	
Previous version	8.0	9.0	16.0	5.0	11.5	4.2-4.3		11.5	2.3	3.0
Current	9.0	10.0	17.0	5.1	11.6	5.0	5.0-6.0	12.0	4.0	
Near future	10.0	11.0	18.0	6.0	12.0					
Farther future		12.0	19.0							

Note: Incomplete support by older webkit browsers refers to only acknowledging different media rules on page reload

Desktop and mobile browser support of CSS3 media queries.

Support of IE and other old browsers

To support older (versions of) browsers, like Internet Explorer 6-8, you can use JavaScript libraries or utilities like Modernizr (http://modernizr.com/), Selectivizr (http://selectivizr.com/) or eCSStender (http://ecsstender.org/).

Note:

In the chapter 'Useful websites' at the end of this document you can find some information about these JavaScript tools.

Device support

The best part of responsive web design is that it covers almost the entire spectrum of displays, from 40lb beige CRT monitors to whatever device Apple or Google dream up next.

The following table shows which mobile devices / operating systems support responsive web design:

Mobile/Platform	Screen variations	HTML5/CSS3 support	CSS3 Media Queries
iPhone 4S / iOS	Landscape & Portrait	Good	Good
Windows 7	Landscape & Portrait	Good	Good
Sony Ericsson Xperia / Android	Landscape & Portrait	Good	Good
HTC Hero / Android	Landscape & Portrait	Good	Good
Nokia / Symbian	Landscape & Portrait	Average	Good

Responsive workflow

The workflow for creating a responsive website is discussed in the next chapters with more detailed information. The three main phases of responsive web design and which discipline is involved:

1st phase: wireframes; designer
2nd phase: prototyping; designer en developer
3rd phase: development; developer

Design guidelines

A perfect responsive web design will be fluid all the way. However, in a project a website is made for targets, such as all smart phones or screens of 1024px wide or more. Responsive web design presents new factors to consider since layouts are no longer static, but dynamic.
The design process starts with defining the breakpoints for the different devices; for example smartphones, tablets, laptops and desktop computers.

Defining breakpoints

Before the design process starts, the developer and the designer have to choose a grid system based on the chosen breakpoints. The breakpoints are defined depending on the devices that need to be supported according to the client. Or if the client doesn't know which devices need or will be supported, you can create breakpoints for common screen resolutions, like 1024px, 768px, 480px, 320px etcetera.

This is an example of breakpoints in Ethan Marcotte's book:

320 pixels	For small screen devices, like phones, held in portrait mode.
480 pixels	For small screen devices, like phones, held in landscape mode.
600 pixels	Smaller tablets, like the Amazon Kindle (600x800) and Barnes & Noble Nook (600x1024), held in portrait mode.
768 pixels	Ten-inch tablets like the iPad (768x1024) held in portrait mode.
1024 pixels	Tablets like the iPad (1024x768) held in landscape mode, as well as certain laptop, netbook, and desktop displays.
1200 pixels	For widescreen displays, primarily laptop and desktop browsers.

TABLE 5.1: A list of example resolution breakpoints.

You can create breakpoints based on different device types with common screen resolutions:

Photoshop Grid

Each web designer can use a grid system appropriate to the project goals. First, define the grid structure for each device. For example, when creating three pages for the screen widths 1024px (desktop), 768px (tablet), 320px (smartphone), it is best to define a grid structure for each of these widths. As you create the layouts you'll need to think about the number of columns and how the components within these will adapt as the page width narrows.
These are fine examples of possible grid systems:

1000px grid

In making the move to responsive web design, one of the potential hurdles is the maths for calculating the percentage-based widths for fluid layouts. Elliot Jay Stocks created a 1000 pixel grid for dealing with flexible layouts. Designing with a 960px grid in Photoshop using six columns, each 140px wide, can be difficult. You divide 140 by 960 to get a percentage-based width: 14.583333%. The point is that the final figure looks like a random number with no immediate relation to the container's pixel width (960) or the element's pixel width (140). Compare that to a container that has a width of 1000px. Dividing by 1000 results in clean percentages. A 140px column inside a 1000px container is 14%. A 500px column in a 1000px container is 50%. The point of designing with round numbers is that it's about making the design and development process easier.

978px grid

Using the 960px grid with 20px gutters and a 940px content area can feel difficult to design when creating a responsive web design. The gutters are narrow, causing text and content to appear close to each other. 978px grid alleviates this by making the gutter space bigger and taking advantage of more screen space. Nick La crafted the 978 grid system and uses 12 columns at 54px, with extra wide 30px gutters. This comes out to a total of 978px, which uses up most of the space on a 1024x768 monitor. Almost every desktop computer and tablet is capable of displaying designs built on this grid.

Note:

In the chapter 'Useful websites' at the end of this document you can find some websites about these and other grid systems.

Responsive patterns for layout and navigation

Before getting to the wireframing process you can consider these layout and navigation patterns. When core content on a page should be displayed, it's important to determine where it is positioned within the different screen sizes. This will give you an idea about adapting content to a variety of screen sizes. Designing a mobile navigation requires a different approach than normal desktop navigations. Looking at the various ways of applying it for small screen sizes gives you insight about navigation patterns.

These are three frequently used layout patterns:

1. Mostly Fluid

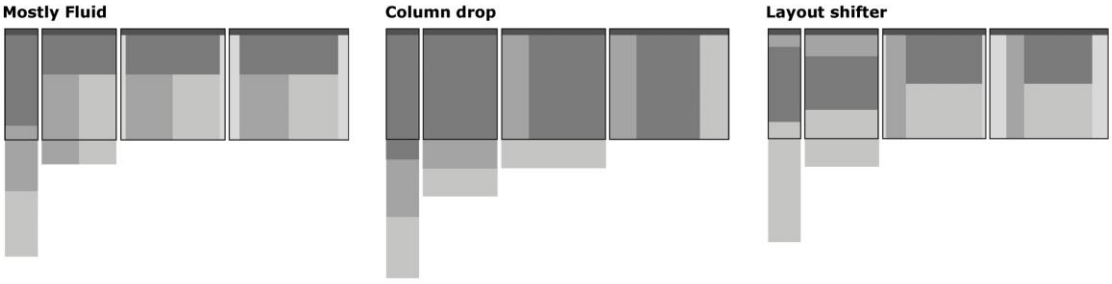
The core structure of this pattern doesn't change until the smallest screen width. It's a multi-column layout that uses larger margins and images on big screens to scale from large screens down to small screen sizes, and stacks columns vertically on mobile devices.

2. Column Drop

This pattern starts with a multicolumn layout and ends up with a single column layout, dropping columns as screen sizes get narrower. Unlike the mostly fluid pattern, the overall size of elements in this layout tend to stay consistent.

3. Layout Shifter

The structure of this pattern changes on every screen. It uses different layouts on large, medium, and small screens.



Brad Frost researched different techniques for handling navigation in responsive web designs:

1. Top Navigation
Navigation is always at the top of the page. When the screen gets narrower, adding a new line with menu items will give you more margin.

2. Footer Anchor
The header contains an anchor button linking to the footer. The navigation sits at the footer of the site. This approach clears up a lot of room for the core content.

3. Select menu.
The navigation transforms into a select (dropdown) menu for small screens.

4. Toggle
The navigation slides open right in the header. The toggle menu appears in place, which doesn't disorient the user.

5. Left Navigation Fly out
The navigation is accessed by a menu icon. Selecting the menu item reveals a frame that slides in from the left and moves the main content over to the right.

6. Footer only
This technique is similar to the footer anchor approach, only without the link button in the header.

Consistency
Make sure to use consistent elements throughout. Always make navigation clearly marked and easily found, because users may have trouble navigating your site on a device when they are used to their desktop design.

Wireframes

The main goal of the wireframing process is to define the logic of the content layout.

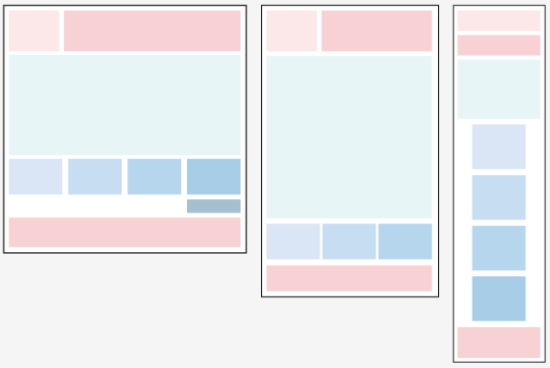
Content Layout

After setting up the grid structure, make a list of all the content that will need to be displayed on the homepage for each device. To give a good impression of the layout, containers can be indicated with modules.

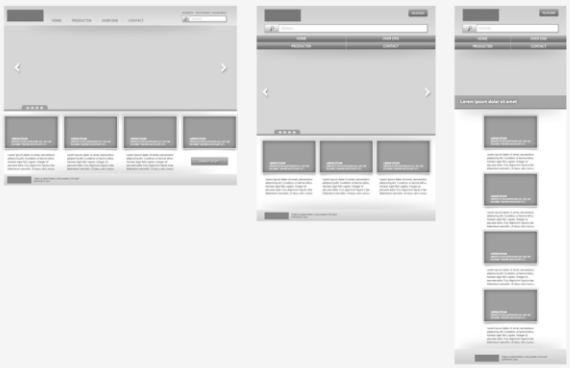
For example:
Header; Navigation; Content; Image; Footer;

In order to best design responsively, you should think about the different parts of your design, not as pieces of a puzzle that must stay in the same place relative to one another, but as modules that can be reorganized and resized. Building a site with modules makes it easy to imagine how your layout will adapt for different viewport sizes.
When the first wireframes are created you can give them more detail, like main navigation, submenu's, search box, content items etc.

An important part of wireframing is having the developer take an active role. The developer can give suggestions and should know right away if the content positioning is going to work or not. Because of the large amount of testing involved, responsive web design is best done in the browser. As soon as a basic wireframe layout is agreed, it is best to begin the development of a prototype right away. Having layouts viewable from a browser helps prevent a lot of design problems.



Wireframe: content layout

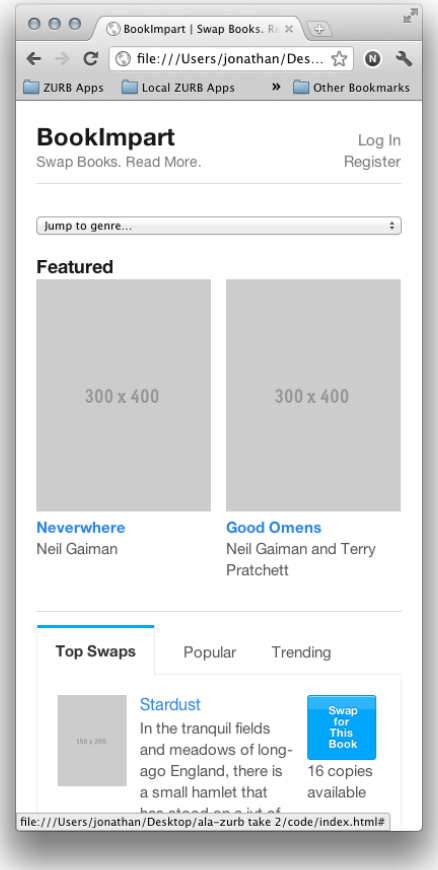


Wireframe: detailed

Prototyping

During the design process the developer is already involved and will also be working closely together with the designer in the prototyping process.
You can think of prototypes as coded wireframes. During a responsive web design project there are multiple instances of prototyping.
The developer starts with the functional prototyping after the wireframing process by creating a prototype based on the content blocks in the wireframes, created by the designer. This way the developer starts building the website step-by-step while the designer continues on making more detailed wireframes.
Then the graphical prototyping starts: the developer builds on the simple prototypes by adding more details based on these wireframes. This way all the design elements from the style tiles and more details from the upgraded wireframes are being inserted in the fluid grid.
By building these prototypes you get, on an early stage, an idea of what goes well and what needs to be changed. For instance, when starting with the prototype for mobile devices you can see if buttons are clickable or if images have the right size. This way the website will be a better product.
What's more, by prototyping you can create more certainty about the final development process, because the client has already approved more than just static designs, as would be the case in the traditional way. The client can better imagine a representation of the final product and can give his feedback, through the feedback rounds on the intermediate phases, on the technique that would otherwise only be possible after a large part of the site was built (which would be much more expensive to adapt).

An example of a prototype (for mobile) based on a wireframe:



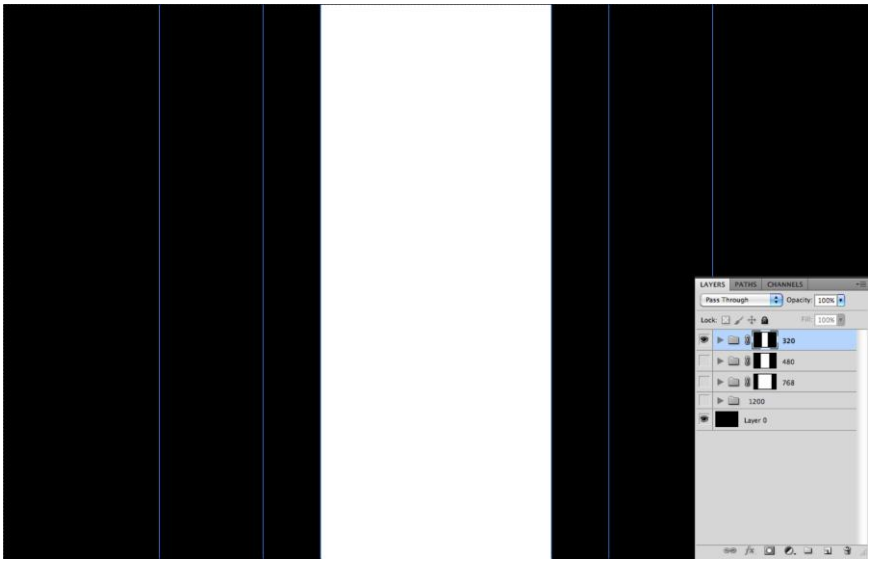
Visual Design

The actual calculated width of each column in a responsive website changes every time the browser window changes size, and cannot be the same across different devices.

Knowing that the website won't have the exact same pixel measure for each column as in your design, there are a few other measures you should take to insure that your layout scales without issue.

Design tool

It's recommended using one .psd for all the layouts. So you can copy and reuse elements for different screen sizes.



Start for example with a new .psd at 1200 pixels wide. The other viewports will be 1024, 768 and 480 pixels wide. To start, unlock your background layer and duplicate it for as many viewports as you need plus one. Fill in the original background layer black and leave the rest white. Put each white background layer in a folder and name it for its viewport (example: "480").

Next set up each viewport in the .psd. You only need to do this one time and you can use the template for all responsive projects.

First add guides at the edges of each viewport. Add guides at 90, 216, 360, 840, and 1110 pixels.
Next create Layer Masks on each folder, which will form the edges of each viewport. Use the Rectangular Marquee Tool to select the area inside the center of two guides.

With the selection made and the correct folder selected in the Layers Pallet, click the Add Layer Mask Button to apply it. Do the same step for the other viewports. Now to see a viewport simply turn off the other folders.

Textures

Try not to use textured borders in columns. Textures would be difficult for developers to use in a responsive website, because they wouldn't scale well horizontally.

Horizontal gradients

Horizontal gradients have also the inability to scale horizontally. If you do have a textured background in a column, make sure it is one that can be tiled easily.

Flexible Layout
Consider what happens to elements in smaller or larger screen widths and what happens to a design when viewed in landscape mode on mobile devices. Creating a fluid architecture creates a better user experience for multiple screen sizes.

Typography: Thinking with Proportions
In scaling the design to accommodate smaller screens, there will come a point at which some text on the page will be too big. For example, text will be too large and will have to wrap over several lines, reducing the impact of the original decision. To help the developer as well as to help maintain the original design, you should decide what text on the page remain constant and what text should be adjusted. You also have to decide how it should be adjusted.

The wrong way to do it:
x should always be 20pt bigger than y. This makes it complex for the developer and it also doesn't make much sense, because you're using a constant value that will never change no matter what the screen size.

The right way to do it:
x should always be 1.5 times as big as y. This way accepts that the only true meaning a design gets out of font size is how big or small the text is compared with other text on the page. Once this is decided, make sure to convey this information to the developer. That's the way to make sure that your decisions are displayed in the actual web page.

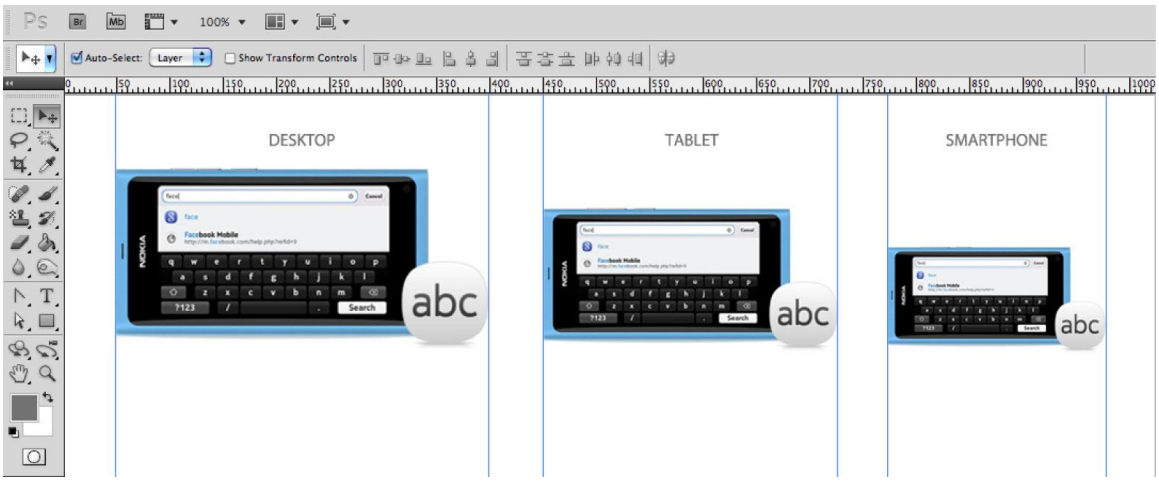
Fonts
When designing in Photoshop, make sure custom fonts translate nicely to mobile devices. Keep in mind that using custom fonts increases the page size and can slow down page load time. Also keep in mind that not every mobile browser supports custom fonts so you may need to make style adjustments for devices.

Flexible Images
An important part of responsive web design is working with flexible images. An image adapts to the size of a device screen. This can be applied this way:

Design graphics and images together. You could create multiple versions of an image, one for desktop browsers, as well for tablets and smartphones. The server could serve the most appropriate image for the specific resolution.

This provides a more global and contextual view:

- It helps eliminate alignment errors and continuity issues.
- It helps maintain consistency. Images don't have to be identical but the overall message should not be lost.
- You can also vary file types (PNG, JPG) to increase the performance. This way you keep the file size down and have images that will work in each viewport size.



The Priority Guide
With the priority guide, you create a single deliverable that provides direction for content-focused design and mobile-first development in a wireframe.

A mobile-sized approach is narrow and forces more of a single column layout. The single column causes a linear display of content and features. This linear display makes priority and hierarchy much more apparent than a desktop-sized wireframe.

With just this priority guide, the designer sets off to create a traditional desktop resolution website. For a web designer it is in line of thinking to make design decisions for a desktop resolution while looking at a mobile plan.

Benefits

- The developer is provided a direction, both desktop and mobile. Two guides to follow, each offering unique information. Some interpretation still has to be made, but there is far less guess work.
- The designer is given a wireframe that provides hierarchy but does not dictate desktop-sized layout.
- From the prototype and the linear approach, hierarchy can be understood fairly quickly, and the foundation for mobile-first markup and style is implied but flexible.

Style Tiles

Style tiles are a design deliverable consisting of fonts, colors and interface elements that communicate the look and feel of the website. They are for when a mood board is too vague and a design mock up is too specific. Style tiles show a direct connection with actual interface elements without defining the layout.

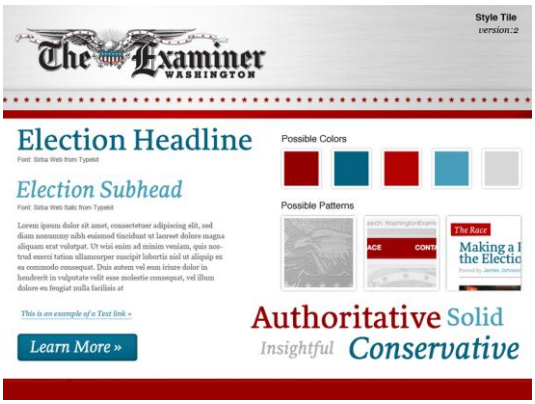
Advantages

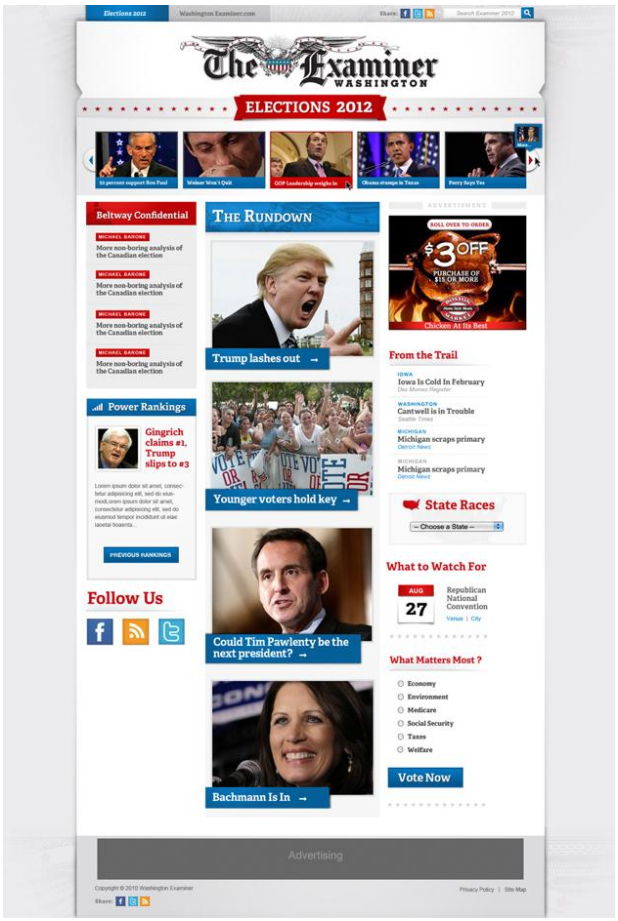
1. It's possible to create several style tiles in a short period of time.
2. Client feedback is easier to integrate.
3. You can create the look of elements without worrying about specific device dimensions or breakpoints.

Disadvantage

Clients want to see the development process of a website. Showing them elements, provides only a basic understanding of the website.

An example of a style tile and the actual website:





Note:

In the chapter 'Useful websites' at the end of this document you can find a download link.

Technical guidelines

Creating a fluid grid

Every aspect of a flexible (fluid) grid – the rows and columns, and the modules draped over them – can be expressed as proportions of their containing element, rather than in unchanging, inflexible pixels.

From pixels to percentages

Instead of a value set in pixels, we need to express a width of i.e. 900px in proportional terms: specifically, describing it as a percentage of the width of its containing element (i.e. 960px).
With the formula: *target ÷ context = result* (i.e. 900 ÷ 960 = 0.9375).
This is the result: width: 93.75%; /* 900px / 960px */. You can put the math behind the measurement in a comment on the right side of the CSS rule.
So with some simple math you can create a percentage-based container and flexible columns, creating a layout that resizes in concert with the browser window. And as it does, the pixel widths of those columns might change – but the proportions of our design remain intact.

Margins and padding

To preserve the integrity of our flexible grid we can also create percentage-based margins and padding. And we can reuse the target ÷ context = result formula to do so.
Before we start in with the math, it's important to note that the context is slightly different for each, and depends on whether you're setting flexible margins or flexible padding:
1. When setting flexible margins on an element, your context is the width of the element's container.
2. When setting flexible padding on an element, your context is the width of the element itself. Which makes sense, if you think about the box model: we're describing the padding in relation to the width of the box itself.

i.e. Our target value is 48px. And since we're working with relative padding, our context should be the width of the element itself. But once again, since there's no explicit width set on the element, we can simply use 900px, the width of its parent element, for our context:
padding: 5.33333333%; /* 48px / 900px */.
You may be tempted to round 5.33333333% to 5.33% but it perfectly represents the padding in proportional terms. What's more, browsers are perfectly adept at rounding off those excess decimal places as they internally convert the values to pixels. So giving them more information, not less, will net you a better, more accurate result in the end.

But building a flexible grid isn't entirely about the math. The target ÷ context = result formula makes it easy to articulate those proportions into stylesheet-ready percentages, sure – but ultimately, we need to break our habit of translating pixels from Photoshop directly into our CSS, and focus our attention on the proportions behind our designs. It's about becoming context-aware: better understanding the ratio-based relationships between element and container. But a fluid grid is just our foundation, the first layer of a responsive design.

Note: *In the chapter 'Useful websites' at the end of this document you can find some websites about grid systems.*

Using fluid media

Fluid images

To prevent images from exceeding the width of their container we can write a rule:

```
img {
max-width: 100%;
}
```

First discovered by designer Richard Rutter, this one rule immediately provides an incredibly handy constraint for every image in our document. Now, our img element will render at whatever size it wants, as long as it's narrower than its containing element. But if it happens to be wider than its container, then the max-width: 100% directive forces the image's width to match the width of its container.
What's more, modern browsers have evolved to the point where they resize the images proportionally: as our flexible container resizes itself, shrinking or enlarging our image, the image's aspect ratio remains intact.

Fluid media

The max-width: 100% rule can also apply to most fixed-width elements, like video and other rich media. In fact, we can beef up our selector to cover other media-ready elements, like so:

```
img,
embed,
object,
video {
max-width: 100%;
}
```

max-width in Internet Explorer

The cold, hard truth is that Internet Explorer 6 and below don't support the max-width property. While IE7 and above are positively brimming with support for max-width. But if you're stuck supporting IE6 or lower, our approach needs refinement.

If you tend to favor a more lo-fi, CSS-driven approach. Namely, all modern browsers get our max-width constraint, you can use this:

```
img,
embed,
object,
video {
max-width: 100%;
}
```

But in a separate IE6-specific stylesheet, you'll have to include the following:

```
img,
embed,
object,
video {
width: 100%;
}
```

The difference is that IE6 and lower get width: 100%, rather than the max-width: 100% rule.
A word of warning: tread carefully here, for these are drastically different rules. Whereas max-width: 100% instructs our images to never exceed the width of their containers, width: 100% forces our images to always match the width of their containing elements.
Most of the time, this approach will work just fine. But for smaller images like thumbnails, or most embedded movies, it might not be appropriate to blindly up-scale them with CSS. If that's the case, then a bit more specificity might be warranted for IE:

```
img.full,
object.full,
.main img,
.main object {
width: 100%;
}
```

There are a few other options for working fixed-width images into a fluid context.
One method Ethan Marcotte has used on a few occasions is the overflow property. As discussed earlier, wide images will, by default, simply bleed out of their containing elements. And in most cases, the max-width: 100% rule is the best way to constrain them, snapping them back down to a manageable size. But alternately, you could simply clip off that excess image data by applying overflow: hidden. So rather than setting our inset image to resize itself automatically:

```
.feature img {
max-width: 100%;
}
```

We could instead simply clip off all that excess, overflowing data like so:

```
.feature {
overflow: hidden;
}
.feature img {
display: block;
max-width: auto;
}
```

Now, as you can see, this isn't really a workable solution. In fact, the overwhelming majority of cases, overflow is generally less useful than scaling the image via max-width. But still, it's an option to be considered, and one you might find some use for.

It's worth noting that both the overflow and max-width: 100% approaches to flexible images are actually pretty robust, and work remarkably well for most kinds of media.
However, both approaches are ultimately "content-blind." Each establishes some basic rules for the way an image interacts with its container: max-width: 100% scales oversized images down to match the width of their containers, while controlling overflow allows the designer to conceal any image data that might bleed out of its containing element.
If however your image is especially information-rich, simply scaling or cropping it might be less than desirable—in fact, those approaches might actually impede your readers' ability to understand the content contained in that image.
If that's the case, it might be worth investigating ways of delivering different versions of the same image to different resolution ranges. In other words, you could create multiple versions of your infographic—

one ideal for desktop browsers, as well as another, more linearized version for small-screen devices. With those options established, a server-side solution could intelligently serve the most appropriate image for that resolution range.

Note:
In the chapter 'Useful websites' at the end of this document you can find some websites about image and video scaling.

Applying media queries

Media queries are actually a very important part of responsive web design. A media query allows us to target not only certain device classes, but to actually inspect the physical characteristics of the device rendering our work. For example, following the recent rise of mobile WebKit, media queries became a popular client-side technique for delivering a tailored style sheet to iPhone, Android and other phones. With these media queries you can define in your stylesheet the CSS that must be applied to a specific device. This way we can create different and appropriate experiences on every device.
When you start with 'mobile first' for instance, you begin by defining a layout appropriate to smaller screens, and then use media queries to progressively enhance your design as the resolution increases.

Media queries are an incredibly robust mechanism for identifying not only types of media, but for actually inspecting the physical characteristics of the devices and browsers that render our content. An example:

```
@media screen and (min-width: 1024px) {
   body {
   font-size: 100%;
   }
}
```

Now, every media query—including the one above—has two components:
1. Each query begins with a media type (i.e. screen), drawn from the CSS2.1 specification's list of approved media types (http://www.w3.org/TR/CSS21/media.html#media-types).
2. Immediately after comes the query itself, wrapped in parentheses: (min-width: 1024px). And our query can, in turn, be split into two components: the name of a feature (min-width) and a corresponding value (1024px).

Think of a media query like a test for your browser. When a browser reads your stylesheet, the screen and (min-width: 1024px) query asks two questions: first, if it belongs to the screen media type; and if it does, if the browser's viewport is at least 1024 pixels wide. If the browser meets both of those criteria, then the styles enclosed within the query are rendered; if not, the browser happily disregards the styles, and continues on its merry way.
Our media query above is written as part of an '@media' declaration, which enables us to put queries directly inside a stylesheet. But you can also place queries on link elements by inserting them into the media attribute:

```
<link rel="stylesheet" href="wide.css" media="screen and (min-width: 1024px)" />
```

Or you can attach them to '@import' statements:

```
@import url("wide.css") screen and (min-width: 1024px);
```

The @media approach could be more preferable since it keeps your code consolidated in a single file, while reducing the number of extraneous requests the browser has to make to your server. But no

matter how you write your queries, the result in each scenario is the same: if the browser matches the media type and meets the condition outlined in our query, it applies the enclosed CSS, otherwise it won't.

Media features

Two guidelines:

1. In the spec's language, every device has a "display area" and "rendering surface." This may seem clear, but think of it this way: the browser's viewport is the display area; the entire display is the rendering surface. So on your laptop, the display area would be your browser window; the rendering surface would be your screen.
2. To test values above or below a certain threshold, some features accept min- and max- prefixes. A fine example is width: you can serve CSS conditionally to viewports above 1024 pixels by writing (min-width: 1024px), or below 1024 pixels with (max-width: 1024px).

The media features are: width, height, device-width, device-height, orientation, aspect-ratio, device-aspect-ratio, color, color-index, monochrome, resolution, scan and grid.

What's really exciting is that we can chain multiple queries together with the 'and' keyword:
@media screen and (min-device-width: 480px) and (orientation: landscape) { … }.
This allows us to test for multiple features in a single query, creating more complex tests for the devices viewing our designs.

Testing

As all websites need testing as usual, below are listed some methods to test a responsive website. Usualy this process consists only a desktop website but by adding a tablet and mobile to the testing process will give you a lot of useful feedback about the different devices in an early stage.

Testing tool

http://labs.adobe.com/technologies/shadow/

With Adobe Shadow a (responsive) website can be inspected, tested and debugged remotely (from a host PC) at various and multiple mobile devices simultaneously.

Test websites

Below are a number of test sites shown with which a responsive website can be tested by entering the URL to the website (local as well as online):

- http://mattkersley.com/responsive/ A tool to test in a variety of (common) device sizes.
- http://quirktools.com/screenfly/ A tool to test in a variety of (common) device sizes and resolutions.
- http://responsive.is/ A tool to test in common device sizes.
- http://www.responsinator.com/ A tool to test in a variety of (common) device sizes.
- http://responsivepx.com/ A tool with which you can adjust the width and height of the viewport to find exact breakpoint widths (in pixels) to use this information in your media queries.
- http://www.benjaminkeen.com/misc/bricss/ On this website a bookmarklet can be generated with a number of desired device sizes with which a responsive website can be tested.
- http://seesparkbox.com/foundry/media_query_bookmarklet On this website there is a bookmarklet available that can be dragged to the bookmarks bar which shows the media queries of a responsive website. It works however only in Chrome (9+) and Safari (5+).
- http://codeinchaos.github.com/mobile-screens/ A tool to test on a few common smartphones and tablets.

Mobile test application

- http://itunes.apple.com/nl/app/aptus/id510487565?mt=12
- Aptus is a dedicated browser that lets you preview, edit and screenshot your responsive site at any size from mobile through to large desktop.
- http://www.keynotedeviceanywhere.com/mobile-application-testing.html
- An application with which a responsive site can be tested on many different devices, both (very) old and most recent devices, such as an iPad 3.

Note:

Both are paid applications.

Device wall interactive

Last but not least, the device wall of the interactive department can be used to test a responsive website. As well as the other test devices the IT team has at their disposal.

Useful websites

Below are listed some websites which can be useful during the responsive web design project:

Grid system

- http://elliotjaystocks.com/blog/a-better-photoshop-grid-for-responsive-web-design/ An article about a 1000px grid system.
- http://978.gs/ 978 Grid system for Web Design
- http://960.gs/ 960 Grid system
- http://cssgrid.net/ The 1140px CSS grid system, fluid down to mobile
- http://www.tinyfluidgrid.com/ The happy & awesome way to build fluid grid based websites
- http://goldengridsystem.com/ A folding grid for responsive design
- http://framelessgrid.com/ A fixed-width grid. Use media queries to adapt your design as more columns become available. Since you'll be adapting column by column instead of pixel by pixel, you can choose exactly when your layout should and shouldn't adapt.
- http://lessframework.com/ A CSS grid system for designing adaptive web-sites
- http://kflorence.github.com/flurid/ A cross-browser CSS grid framework that doesn't hide pixels in margins
- http://csswizardry.com/fluid-grids/ A fluid grids calculator
- http://www.gridsystemgenerator.com/ A grid system generator

Responsive patterns

- http://bradfrostweb.com/blog/web/responsive-nav-patterns/#hide Responsive navigation patterns
- http://www.lukew.com/ff/entry.asp?1514 Multi-device layout patterns

Style Tiles

- http://styletil.es/ Style Tiles; a Visual Web Design Process for Clients & the Responsive Web

JavaScript for CSS3 browser support

- http://modernizr.com/ Modernizr is an open-source JavaScript library that detects the availability of native implementations for next-generation web technologies, i.e. features that stem from the HTML5 and CSS3 specifications.
- http://selectivizr.com/ Selectivizr is a JavaScript utility that emulates CSS3 pseudo-classes and attribute selectors in Internet Explorer 6-8.
- http://ecsstender.org/ eCSStender is a JavaScript tool that enables you to implement complicated CSS3 properties with very little code.

Boilerplate

A boilerplate is a basic HTML/CSS/JS template for a fast, robust and future-safe site.

- http://www.initializr.com/ A HTML5 template generator based on HTML5 boilerplate
- http://getskeleton.com/ A beautiful boilerplate for responsive, mobile-friendly development
- http://stuffandnonsense.co.uk/projects/320andup/ The 'tiny screen first' boilerplate extension prevents mobile devices from downloading desktop assets by using a tiny screen's stylesheet as its starting point
- http://html5boilerplate.com/ A rock-solid default for HTML5 awesome
- http://html5boilerplate.com/mobile Mobile boilerplate

CSS frameworks

A CSS framework is a library of CSS files that is used to make development of standards-based HTML and CSS web pages quick and painless. It typically provides CSS styles for typography, layout and browser resets.

- http://csswizardry.com/inuitcss/ CSS framework; inuit.css is built to work on smaller screens (such as tablets) and tiny screens (such as phones) straight out of the box with minimal effort
- http://www.stuffandnonsense.co.uk/blog/about/hardboiled_css3_media_queries/ A template for media queries for standard devices
- http://blueprintcss.org/ CSS framework

Polyfills

A polyfill is a browser fallback, made in JavaScript, that allows functionality you expect to work in modern browsers to work in older browsers, i.e. to support CSS3 media queries or HTML5 features like 'canvas' in older browsers.

- https://github.com/scottjehl/Respond A fast & lightweight polyfill for min/max-width CSS3 Media Queries (for IE 6-8, and more)
- http://code.google.com/p/css3-mediaqueries-js/ A JavaScript library to make IE 5+, Firefox 1+ and Safari 2 transparently parse, test and apply CSS3 Media Queries

Image scaling

- https://github.com/adamdbradley/foresight.js Foresight gives webpages the ability to tell if the user's device is capable of viewing high-resolution images (such as the 3rd generation iPad) before the image is requested from the server. Additionally, it judges if the user's device currently has a fast enough network connection for high-resolution images. Depending on device display and network connectivity, foresight.js will request the appropriate image for the webpage.
- https://github.com/danmillar/jquery-anystretch Anystretch is a jQuery plugin that allows you to add a dynamically-resized background image to any page or block level element. The image will stretch to fit the page/element, and will automatically resize as the window size changes
- https://github.com/filamentgroup/responsive-images An Experiment with Mobile-First Images that Scale Responsively & Responsibly
- http://srobbin.com/jquery-plugins/backstretch/ A simple jQuery plugin that allows you to add a dynamically-resized background image to any page
- http://retinajs.com/ An open source script that makes it easy to serve high-resolution images to devices with retina displays

Video scaling

- http://fitvidsjs.com/ A lightweight, easy-to-use jQuery plugin for fluid width video embeds
- http://css-tricks.com/NetMag/FluidWidthVideo/Article-FluidWidthVideo.php A 'tutorial' on fluid with video

Text scaling

http://fittextjs.com/ A jQuery plugin for inflating web type

SVG scaling

http://raphaeljs.com/ A small JavaScript library that should simplify your work with vector graphics on the web

Resources

1. Marcotte, E. (2011). *Responsive Web Design*. New York: Happy Cog.
2. http://www.slideshare.net/livefront/responsive-design-7877412
3. http://www.alistapart.com/articles/responsive-web-design
4. http://www.designlunatic.com/2011/11/pros-and-cons-of-responsive-web-design/
5. http://thepam.blogspot.com/2011/08/pros-and-cons-of-responsive-web-design.html
6. http://www.slideshare.net/jameswillweb/css3-media-queries-mobile-elixir-or-css-snake-oil
7. http://caniuse.com/css-mediaqueries
8. http://blogs.creative-jar.com/post/Mobile-support-for-Responsive-Web-Design.aspx
9. http://www.lullabot.com/articles/responsive-adaptive-web-design
10. http://www.zeldman.com/2011/07/06/responsive-design-i-dont-think-that-word-means-what-you-think-it-means/
11. http://blog.easy-designs.net/archives/2011/11/16/on-adaptive-vs-responsive-web-design/
12. Wroblewski, L. (2011). *Mobile First*. New York: Happy Cog.
13. http://bradfrostweb.com/blog/web/mobile-first-responsive-web-design/
14. http://andreasviklund.com/learn/graceful-degradation-vs-progressive-enhancement-part-1/
15. http://gravitydept.com/blog/responsive-ecommerce/
16. http://www.slideshare.net/yiibu/pragmatic-responsive-design
17. http://verekia.com/initializr/responsive-template
18. http://www.rippleffect.com/news-and-views/2012/02/01/user-experience-responsive-web-design/
19. http://www.uxbooth.com/blog/how-to-design-a-mobile-responsive-website/
20. http://www.alistapart.com/articles/dive-into-responsive-prototyping-with-foundation/
21. http://trentwalton.com/2011/07/14/content-choreography/
22. http://couchable.co/blog/post/a-responsive-design-workflow
23. http://webdesign.tutsplus.com/articles/design-theory/designing-for-a-responsive-web/
24. http://www.webdesignerdepot.com/2012/02/designing-for-responsiveness/
25. http://www.lukew.com/ff/entry.asp?1514
26. http://bradfrostweb.com/blog/web/responsive-nav-patterns
27. http://mobilewebbestpractices.com
28. http://elliotjaystocks.com/blog/a-better-photoshop-grid-for-responsive-web-design/
29. http://www.webdesignshock.com/responsive-web-design/
30. http://dev.opera.com/articles/view/responsive-web-design-a-project-management- perspective/
31. http://uxdesign.smashingmagazine.com/2012/05/30/design-process-responsive-age/