Responsive Web Design

G3 = Filip Krstanovic, Thomas Schlager, Igor Skoric

Survey Report
Part of *Information Architecture and Web Usability* Course 2012,
Institute for Information Systems and Computer Media (IICM),
Graz University of Technology
A-8010 Graz, Austria

03 Dec 2012

Abstract

This survey explores the term responsive web design and its components. It tries to give an overview of core technologies related to the topic. We introduce the concept of media queries, a necessity to modern responsive designs. A short field research analyses and interprets current responsive design trends. It finds several answers to how and why designers use certain visual design patterns. We can conclude that it will be hard for developers to avoid responsive design concepts in the future as it targets multiple problems of today's web architecture concerning the evolving device market (desktops, tablets and phones).

Further on we investigate possibilities to cope with specific multimedia elements like images and videos as part of the web and their role in responsive design. In the end we introduce techniques that will further enhance our user experience of the web: feature specific responsive design. A concept that adapts website content by the visiting device's supported features.

Contents

Chapter 1

Introduction

This survey is part of our contribution to the *Information Architecture and Web Usability* course held in winter term 2012 at *Graz University of Technology*. It explores and explains several aspects of the responsive web design trend. Its goal is to deliver basic understanding of underlying fundamentals, trend insights based on field research and selected core features.

Chapter 2 introduces the technological premises necessary to implement many responsive features. Chapter 3 shows results of a field research on several heavily responsive websites which may explain certain phenomena. Chapter 4 takes a narrower look at media elements (pictures, videos, ...) and available methods to apply responsive design to them. Chapter 5 focusses on responsive website content relying on features supported by the rendering devices such as a smartphone.

In his book Marcotte [2011, p.9] defines the term *Responsive Web Design* as a composite of three ingredients:

- A flexible, grid-based layout
- Flexible images and media
- Media queries

While we do not completely reject this definition it seems to be a rather practical and unscientific explanation. In the context of this survey we rely on a more general understanding. A website contains responsive elements if they meet one of the following criteria:

- Elements dynamically reorganise based on available screen space to provide an optimised viewing experience on any visiting device
- Resources like images and videos are subject of context sensitive rollout adapting to the visiting device's needs
- Specific elements are displayed or changed in respect to the visiting device's supported feature set

In his presentation Polacek [2012] uses a short and elegant phrase to explain responsive web design: "Responsive websites respond to their environment". The verb **respond** corresponds to several techniques like offering multiple fixed width layouts or fluid grid layouts. Further on he mentions another important aspect. The responsive website design approach focusses on developing a single site, adapting to multiple devices - not on multiple sites for multiple devices.

Chapter 2

Media Queries

With the number of screen resolutions in use nowadays and the rising demands of both clients and users, web designers face the ultimate challenge: how to enable website browsing on all those different screen resolutions without degrading user experience.

Problems such as navigation issues, scaling/cropping of images, data tables, constriction of content, with which designers used to struggle on a daily basis, became easier to solve thanks to media queries and other responsive web design techniques. [Marcotte, 2011]

2.1 Media types

2.1.1 Definition

The first attempt of the W3C (World Wide Web Consortium) to solve the problems and issues that web designers face, were the media types, part of the CSS 2 specification.

Media types were created so that we could better design for each type of browser or device, by conditionally loading CSS tailored for each one of them. So a screen-based device would ignore CSS loaded with the print media type, and vice versa. An "all" super-group was created for style rules meant to apply to all devices.

```
<link rel="stylesheet" href="global.css" media="all" />
<link rel="stylesheet" href="main.css" media="screen" />
<link rel="stylesheet" href="paper.css" media="print" />
```

As the media types specify a type of media to target, in our example, global.css would be loaded for all media types, main.css only for the screen media type and paper.css for the print media type.

A @media rule specifies the target media types (separated by commas) of a set of statements (delimited by curly braces). Invalid statements must be ignored. The @media construct allows style sheet rules for various media in the same style sheet:

```
@media print { body { font-size: 10pt } }
@media screen { body { font-size: 13px } }
@media screen, print { body { line-height: 1.2 } }
```

2.1.2 Types

The names picked for CSS media types reflect target devices for which the relevant properties make sense. Recognized media types are: All, Braille, Embossed, Handheld, Print, Projection, Screen, Speech, Tty, Tv. [W3C, 2012b]

2.1.3 Miscast types

When smartphones with small-screen browsers and tablets, arrived on the market, several problems with media types became evident. Designers could have targeted them by creating a stylesheet for the handheld media type:

```
<link rel=" stylesheet " href="tiny . css" media="handheld"/>
```

The problem with this approach was that early mobile devices were ignored, because they didn't have sufficiently capable browsers, therefor designers were choosing to design instead compelling screen or print specific stylesheets. When finally capable small-screen browsers appeared, there weren't a lot of handheld CSS files in use on the web, resulting in making screen-based stylesheets default by many mobile browser makers.

One of the problems that became obvious is that a handheld stylesheet wasn't suited to ad¬dress the designing challenges for different phones, for example, an iPhone and a five year-old feature phone.

2.2 Media queries

2.2.1 Definition

Realizing some of the drawbacks of the media types, the W3C used their work on CSS3 to take another attempt to solve previously addressed problems. The result was media queries, which extend and improve the idea of media types.

Rather than looking for a device type, media queries look at the capability of the device by inspecting the physical characteristics of the devices and browsers that render our content.

They are an incredibly robust mechanism for identifying not only media types, but for actually detecting different media features, such as:

- Width and height of the browser window ("viewport")
- Device width and height
- Orientation, i.e. is a phone or a tablet in landscape or portrait mode?
- Resolution [JavaScriptKit, 2012]

If the user has a browser that supports media queries, we can then write CSS specifically for certain situations. For example, detecting that the user has a small device like a smart phone of some description and giving them a specific layout.

2.2.2 Viewport

Constraints such as "min-" or "max-" can be used to prefix most media features, therefor avoiding the "<" and ">" symbols, which would conflict with HTML and XML.

A new meta-tag was developed due to mobile device inconsistencies with "min / max-width" & "min / max-device-width" and has now been integrated into the W3C's recommendations for Mobile Web Application Best Practices. Viewport tells the mobile device to interpret the browser width as the physical width of the mobile device in use. [1] The viewport meta-tag needs to be placed in between the head tag "<head></head>":

```
<meta name="viewport" content="width=device−width, initial −scale=1.0">
```

2.2.3 Syntax

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

Every media query consists of: [Marcotte, 2011]

- A media type: drawn from the CSS2.1 specification's list of approved media types (see previous chapter), in our example the media type is "screen".

[1] http://www.w3.org/TR/mwabp/#bp-viewporttext

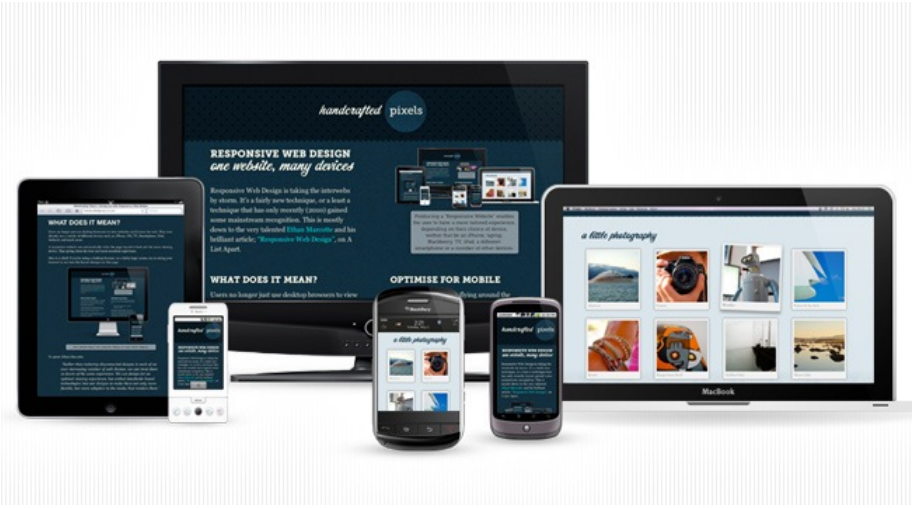


Figure 2.1: Displaying a website on various devices in different resolutions using media queries

- Zero or more expressions that check the conditions of media features. The query itself is wrapped in pa¬rentheses: (min-width: 1024px) and consists of two components: the name of a feature (min-width) and a corresponding value (1024px).

A media query is a logical expression that is either true or false. A media query is true if the media type of the media query matches the media type of the device where the user agent is running and all expressions in the media query are true. If using a media feature without specifying a value, the expression resolves to true if the feature's value is non-zero.

In our example this media query would be true only for media type screen with width larger than 1024px, otherwise it would be false and ignored by the browser.

Furthermore, it is possible to construct complex media queries using logical operators, including not, and, and only, as well as combining multiple media queries in a comma-separated list.

The not keyword negates the result of the query; for example "all and (not color)" is true for monochrome devices regardless of media type. The only keyword hides style sheets from older browsers that don't support media queries:

```
<link rel=" stylesheet " media="only screen and (color)" href="example.css" />
```

2.2.4 Example

Defining max and min (browser) screen sizes for different devices:
i.e. 768px - portrait width of tablets

```
@media screen and (max-width: 768px){ css goes here}
```

i.e. 320px - portrait width of smartphones

```
@media screen and (max-width: 520px){ css goes here}
```

i.e. 1024px - landscape width of typical netbook resolutions as well as various desktop monitor resolutions

```
@media screen and (min-width: 1200px){ css goes here}
```

2.2.5 Why use media queries?

The growth of the number of smartphones and tablets produced in the last years and the diversity of screen sizes available, made it practically unimaginable to ignore users that browse on those devices. Furthermore, building and maintaining versions to fit each type of screen or device has become impossible. Each user deserves the best website browsing experience possible and most of the clients today want their website to be mobile compatible, however, that would not be even possible without the concept of "Responsive Webdesign". [Sender 11, 2011]

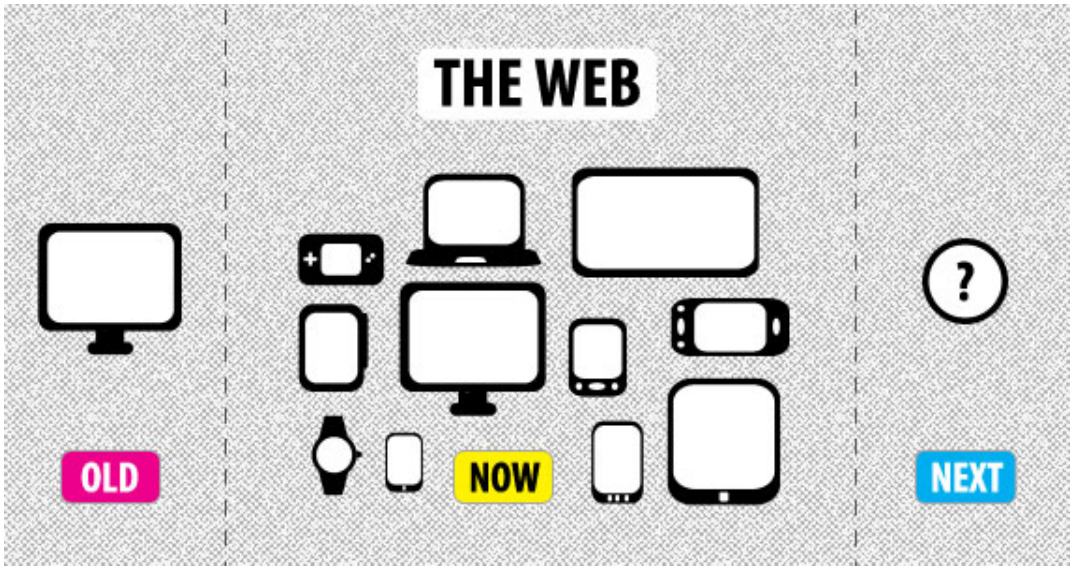


Figure 2.2: Web designing now and then
[artisttechnewmedia.com, 2012]

Only in the mobile market, there have been recorded 21 browsers, 15 OSs and 14 device vendors so far and dozens of different screen resolutions in use. [Koch, 2012]

Moreover, if the user uses a desktop browser doesn't necessary mean that he maximizes the browser window. A poll done by the 456bereastreet website on a sample of 1070 test subjects has shown that only 50.4% of the respondents maximize their browser windows, from which only 20% on Mac and 65% on Windows. [Johansson, 2007]

2.2.6 Browser support

The browser support for media queries is decent. Media Queries enjoy broad support in all modern desktop browsers. Safari, Chrome, Opera, Mozilla support media queries from early versions, Internet Explorer just from the latest version 9.

Things are looking good for media query support beyond the desktop browsers as well. WebKit-based mobile browsers, such as Mobile Safari, HP's webOS, and Android's browser all support media queries, as well as Opera Mobile and Opera Mini.

Unfortunately there are still browsers that have non-native media query support, such as Internet Explorer 8 and earlier versions, IE Mobile and older BlackBerry browsers. However, there are solutions available for these problems, mostly in a form of javascript libraries such as: respond.js library, which patches support for min-width and max-width queries in older browsers and css3-mediaqueries.js library, which makes IE5+, Firefox 1+ and Safari 2 and other older browsers transparently parse, test and apply media queries. [Koch, 2011]

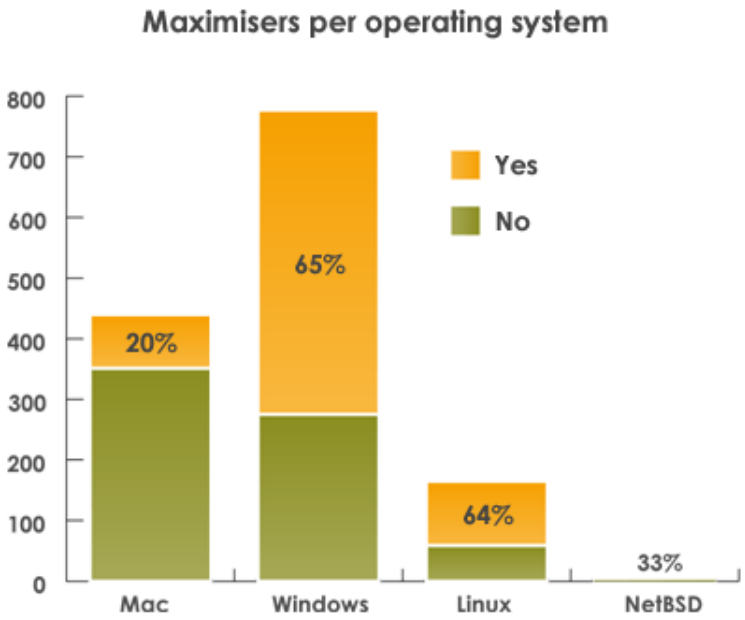


Figure 2.3: Poll results in numbers [Johansson, 2007]

Chapter 3

Responsive Design Patterns of Common Website Areas

To support our understanding of responsive web design we analyse several websites heavily using responsive techniques. All sites included in this field research are sourced at Uggedal [2012] who provides a comprehensive list of up to date responsive sites.

It is necessary to clarify the conditions of this practical survey. Our primary goal is to discover certain patterns (of design decisions) to find quantitative testimony on how today's websites implement responsive features (ignoring any aspects of usability, accessibility or performance) Further on we try to relate specific design choices to common website areas. It is rather difficult to find a clear scientific definition of *common website areas or elements*. One could take a look at elements of web design guidelines as e.g. listed in Bowlby [2008]. But it seems reasonable to reduce our view to structural elements as identified by the Australian Web Advice and Policy Team [2012]:

- Header
- Navigation
- Content
- Footer

Structural elements are not limited to certain parts of a whole website[1]. In many cases they sustain in a constant state on the entire site. This reduces our analysis to the front page. All structural elements contain subelements. In our research we assume that all subelements of a website are exclusively contained within one of the previously mentioned structural areas[2]. Thus the header *may* contain elements like a website or company logo, search fields or social integration. Content areas usually reside below header and navigation containing atomic pieces of informational content. Footers are located below the content containing meta information.

3.1 Analysis Conditions

The analysis is performed on two devices: a desktop computer and a tablet (the latter to check touch compatibility) - both supporting media queries and JavaScript in their browsers. To analyse the structural elements' behaviour we start with the largest possible view of a website[3] and fluently decrease the browser's window width until it reaches 400pixels. By doing this we identify major visual break points as the website degrades its elements (not to be confused with the graceful degradation process [Viklund, 2011; Marcotte, 2011]). 22 sites have been analysed this way.

[1] We apply to the common agreement which uses the term website as a collection of single web pages [Thomas, 2009].

[2] Although in some situations the navigation area can be seen as a subelement of the header we ignore this by now.

[3] Limited by our display's pixel width of 2560 pixels.

3.2 Header

Patterns found in the header can be seen in figure 3.1. It shows one significant strong pattern: *degrade media elements*. The simple reason behind this is the fact that headers normally contain the websites or companies' graphic logo which designers take huge care of. It can be interpreted as it being the most cared about single element on the website.

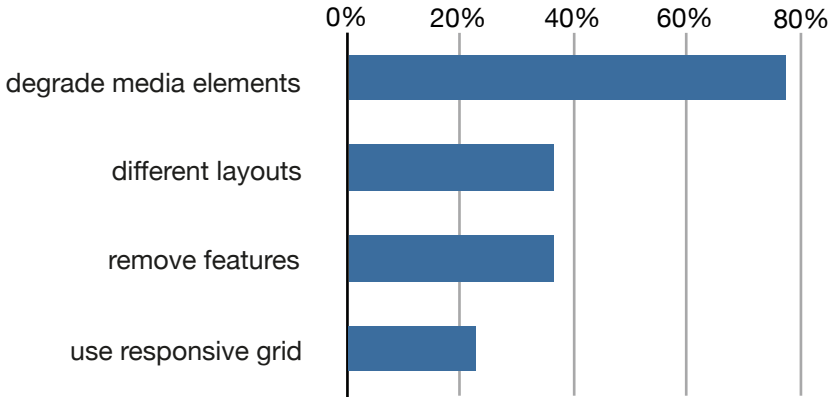


Figure 3.1: Common Design Decisions in Website Headers

Another interesting result is that more than one out of three websites completely rearrange the header layout during the degradation process. This means that designers not only shrink spaces, reduce font sizes or symbolise texts into images but also they freely reorder an move around elements to better fit to one another. Also more than one out of three sites remove features from the header which is a surprisingly low value when you take into account that the a typical degraded site is forced to use only one third or less of the original space available. A typical element to drop out is *social integration*.

3.3 Navigation

Website navigation is a sensible topic challenged by several aspects including usability and accessibility. There are many ways to accomplish navigation and many ways to make errors [Web Page Mistakes, 2008]. Our survey does not further investigate into any of these aspects, as it focusses on design decisions themselves. But it shall be noted that during the research we found all websites to be touch compatible[4] - which is a necessity today and closely bound to responsive web design.

Most websites project its major web pages or sub websites as a list into its navigation system (when speaking of non degraded websites it is a horizontal list in most cases). As pointed out in figure 3.2 we can see that more than 50% of all analysed sites converted a horizontal navigation list into vertical list at some break point. Converting horizontal elements into vertical elements is an obvious choice if horizontal space is narrowing.

The second significant pattern in navigation systems is to collapse the major sections into a single menu button (which will pop up the navigation if activated) which saves large parts of the space consumed. This pattern is not dependent on the previous but often combined with is. Modern touch compatible popup menus (with transitions) can be designed in pure CSS3 using *:active* and *:focus* pseudo classes [Lazaris, 2012].

Certain websites try to avoid the hassles of popup mechanisms by using fluid boxes for buttons or rearranging them in grid fashion. It is clear that this can only be applied to simple menus without deep hierarchical elements. Since navigation is important to keep the site properly functioning, less than 10% of all sites remove elements from it.

[4] Usable on any touchscreen only tablet or smartphone without restrictions.

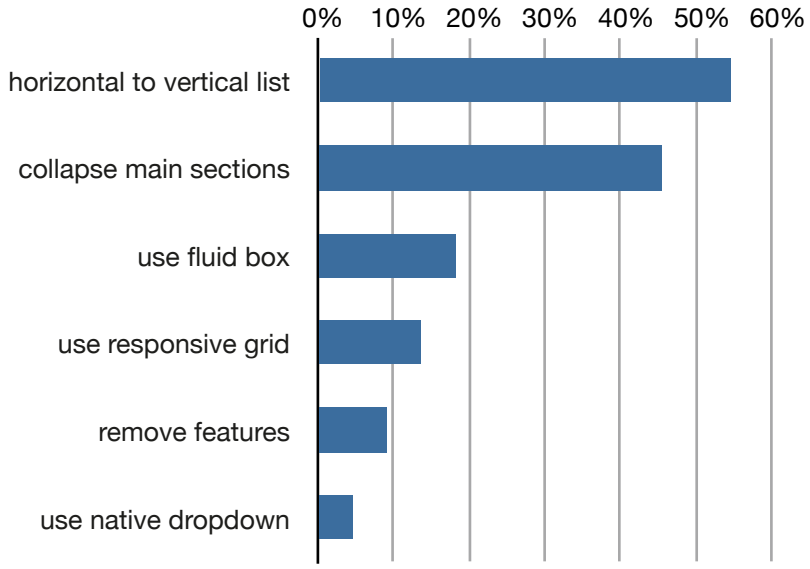


Figure 3.2: Common Design Decisions in Website Navigation

3.4 Content

The content part of a site is altering at any time you change to a different page or section. Therefor it has to be flexible enough to keep up with different types of content. The common answer to this problem today is the flexible grid as described in Marcotte [2011, Chapter 2]. Our research result (as seen in figure 3.3) clearly shows: if you are doing responsive web design and your content is complex enough you use a grid technology.

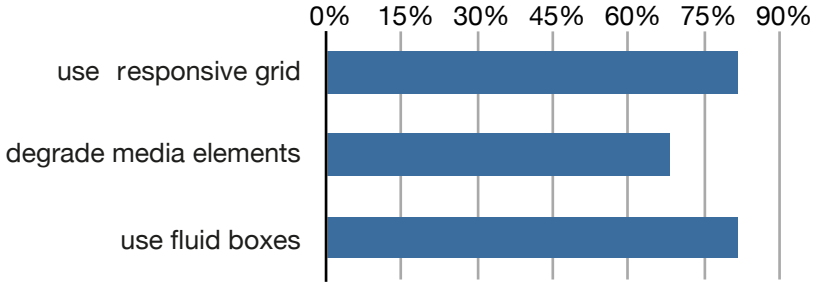


Figure 3.3: Common Design Decisions in Website Content

We can also see a high penetration of fluid boxes which further improves the content's flexibility for subelement arrangement. Browsers have rather weak implementations of text justification (compared to typesetting engines like TEX). Since fluid grid boxes become narrow at times we do not see a lot of justified text these days (only one out of 22 in our test set).

Degrading media content is more difficult than in the header area. Two thirds of all tested sites implemented some means to cope with this topic. Without any further investigation we assume: the more dynamic the content is (and the more contributors you have) the more difficult it becomes to manage high quality degradation of images or videos. E.g static front pages (of companies) have advances over sites relying on author contributions (like magazine sites). To seriously test sites on their ability to cope with content media elements it would be a good idea to divide them up into several content categories and compare them independently.

3.5 Footer

Analysing the footer part of websites partially led to a rather unexpected results. Prior we assumed: if space is shrinking the footer would be a good place to remove several elements. But as shown in figure 3.4 we can see that actually only a small portion of all web sites do this.

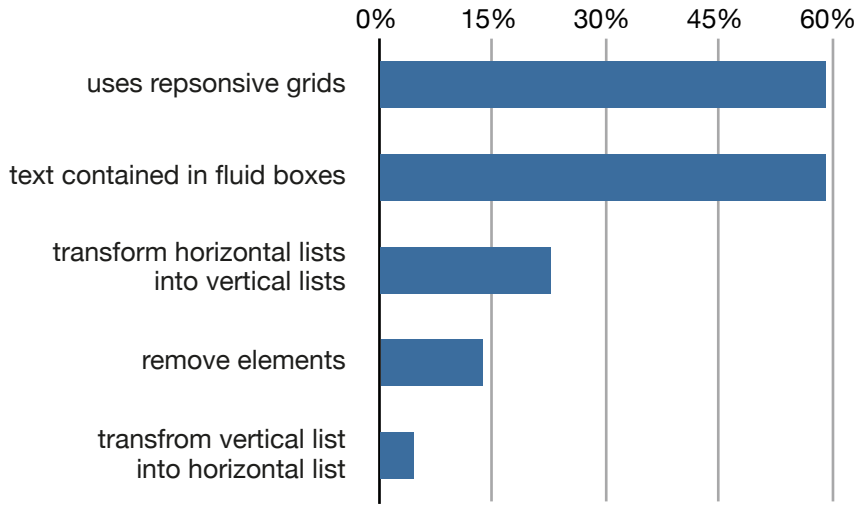


Figure 3.4: Common Design Decisions in Website Footer

The usage of responsive grid ordering is another surprise. Compared with the content area (dynamic content, flexible grids) the footer is almost as static as the website header. A possible explanation for this disparity between header and footer is the on page location. The top spot of a website is carefully organised to manage the available space above the content. In contrast at the bottom of a website it seems rather unimportant how much space an element takes. This idea is also supported by the previous pattern of this section.

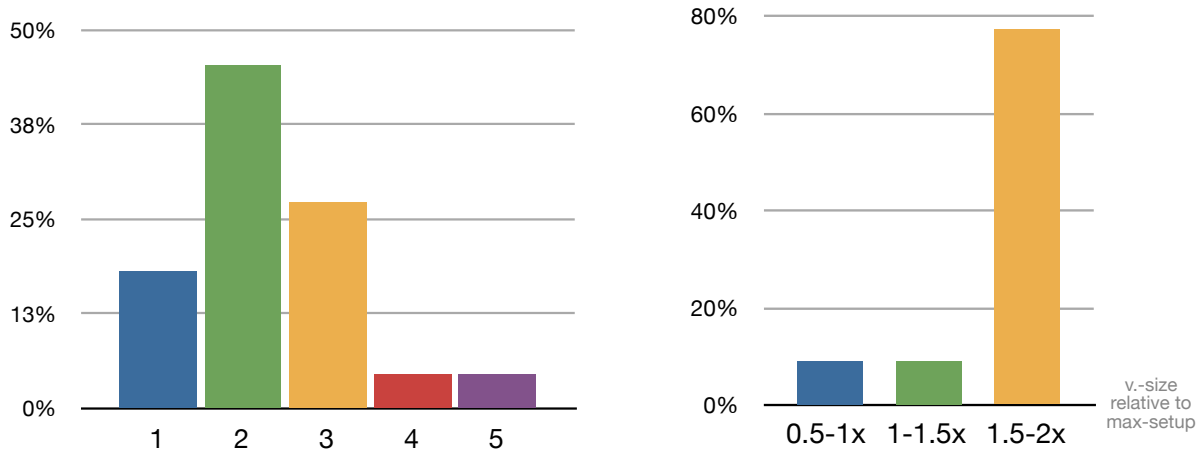
3.6 Break Points and Vertical Scroll Magnitude

Our investigation shows that there is a sweet spot of two major break points used on responsive websites (see figure 3.5a). This number is a pure technical value representing the granularity of degradation or enhancement the developer saw fit. The sweet spot of two may be seen as a reflection of today's device landscape. Desktop computers, tablet computers and smartphones demand three different sizes and need at least two break points.

If you constantly resize browser windows (like we did during testing) an increasing number of break points makes it harder to follow and identify changing elements, especially on complex layouts. But this has less or no effect on real world use cases where screen space is either static or window size isn't changed more than once to fit use.

Another consideration (in the context of responsive design) is a website's vertical scroll magnitude comparing full scale sites and their degraded counterparts:

- A website when rendered at its maximum width setting (usually manually set in a style sheet between 1000 and 1400 pixels) has a vertical magnitude (which can e.g. be measured in pixels or relative to the current window size).
- The same website degraded to a reasonable smartphone width setting (in our test case 400 pixels) has a different vertical magnitude (horizontal elements transform to vertical ones, fluid boxes narrow down and expand vertically, . . .).
- Comparing these two magnitudes results in a *transformation factor* (without dimension). A transformation factor of 1.3 means: a full size web site increases 1.3 times in vertical magnitude when degraded to



(a) Number of Major Break Points

(b) Transformation Factor

Figure 3.5: Break Points and Vertical Magnitude

smartphone size of 400 pixels. We want to clearly state that this is by no means a standard testing procedure or common website feature. We just found it interesting enough to note down the height values to see if there is any significant outcome.

Figure 3.5b shows a breakdown in three categories of vertical transformation factors:

- Sites which shrink $(0.5 - 1.0)$
- Sites which slightly grow $(1.0 - 1.5)$
- Sites which significantly grow $(1.5 - 2.0)$

We can see that most of the websites in our test set significantly grow vertically when being degraded. While an increase between 1.5 and 2.0 is not a horrifying number we think there is room for improvement concerning this topic. We saw sites when rendered on a smartphone take up ten or more vertical screen sizes which makes it harder to locate content subelements.

All relevant recorded data can be found in Appendix A. It contains:

- List of analysed websites
- Found patterns
- Percentage calculations

Chapter 4

Responsive Media Elements

4.1 Images

Images in websites are as common as text - standard. Developing responsive website designs mostly focusses on how to effectively use available space and degrade or enhance specific contained website elements (structure and content). While images themselves can either be scaled or cropped there are many technical ways to achieve these effects. But available space is not always the only concern. When talking about smartphones and mobile connectivity, bandwidth usage and download speeds are increasingly important. As Marcotte [2011, p. 44-49] points out, the trivial approach to flexible images is using the basic CSS property *max-width*:

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

A big problem arises when looking at the results of CSS scaled images. Certain browser do not support or are weak at scaling images this way which can lead to serious artefacts in high resolution images (mostly on earlier versions of Internet Explorer). This problem can partially be conquered by using workarounds with browser specific stylesheets and code snippets. In general working with calculated percentage values can deliver some design benefit but remains with limitations. [Marcotte, 2011]

4.1.1 Alternative Resources

The *Responsive Images Community Group* is currently working on an unofficial draft of an HTML extension called **Picture**. The idea behind this new tag is to provide all means to define alternative sources for images which can then be controlled by pure CSS and combined with media queries. The syntax looks like this:

```
<picture>
  <source media="(min-width: 45em)" srcset="large.jpg">
  <source media="(min-width: 18em)" srcset="medium.jpg">
  <source srcset="small.jpg">
</picture>
```

Right now using the picture tag is rather unappealing because it is only supported by the Chromium browser. In addition it does not take the previously mentioned need for download efficiency into account. Being completely independent from HTTP requests it has no effect on wether a resource is actually downloaded. [rICG, 2012]

Another (yet unsupported) approach loosely connected to the picture tag is presented by Gallagher [2011]. It uses the CSS3 extension *attr()* to combine media queries with multiple data source attributes in the original *img* tag. It suffers the same weaknesses.

Roberts [2011] provides a working solution to swap different sizes of images using CSS only. He uses a combination of background images and *img* tags. Based on a media query it is possible move in smaller and hide larger images when degrading a website. Sample pseudo code:

```
<div class="r-img" style="background:url(large.png); width:..; height:..;">
    <img src="small.png" alt="" />
</div>
```

With CSS properties:

```
.r-img img{
    /* Hide image off-screen on larger devices */
    position:absolute;
    left:-9999px;
}

@media(max-width:480px){
.r-img{
    /* Remove styling from the div */
    background:none!important;
    width:auto!important;
    height:auto!important;
}
.r-img img{
    /* Bring smaller image back into view */
    position:static;
    max-width:100%;
}
}
```

4.1.2 JavaScript Based Rollout

Currently only JavaScript provides all means to cope with both the demands for dynamic sizing and download efficiency. This paragraph shortly describes the method introduced by Jehl [2010] at the *Filament Group*. A JavaScript embedded at the top of the delivered html content rewrites the BASE element and redirects all further requests (stylesheets, images, ...) into a fictional subdirectory. Using a special *.htaccess* file the web server identifies responsive image requests (which have a predefined string in their name) and redirects them to a transparent single pixel image. Requests without the special string are redirected to their original location outside the fictional directory. After the DOM creation has completed, the JavaScript changes the BASE element back to it's original setting and re-requests responsive images based on device resolution. This method has two big benefits:

- On small devices it only requests smaller versions of images
- It has no effect on browsers not supporting it

A similar way is presented by Russel [2010]. It relies on the same 1-pixel-request principle. All image requests are handled by a single PHP script. At DOM creation time the html code invokes browser requests to the PHP file which without further parameters will return a single pixel sized image. After DOM creation is finished a JavaScript calculates images fluid sizes and rewrites their src attributes to include a GET parameter for resolution. At the new request the PHP script will serve a suitable image. This method will not show images when JavaScript is unavailable.

4.1.3 Image Cropping

CSS3 specification includes a proper *crop* property as described on CSS-Portal [2012]. Sadly as of today there is no browser support for this property. The related and similar property *clip* (described on W3Schools.com [2012]) in contrast is well supported. But clipping an image does not solve any space saving problems. Marcotte [2011] introduces a weak cropping mechanism using the CSS *overflow* property which makes images look

cropped by hiding the overflown parts of background images in div containers. But this way the cropping window's top left corner is limited to start at coordinates (0,0) at any time.

By using certain CSS tricks it is still possible to achieve cropping behaviour today. When combining property *overflow* with negative margins, *div* contained images can be properly cropped. Example:

```
<style>
  .crop { width: 200px; height: 150px; overflow: hidden; }
  .crop img { width: 400px; height: 300px; margin: -75px 0 0 -100px; }
</style>
...
<div class="crop">
    <img src="..." alt="..." />
</div>
```

The principle relies on the simple fact that CSS margins can be negative (see [Imbong, 2009]). In that way an outer *div* element can define the cropping window and the margins of the contained *img* tag move it relative to the clip area - creating a crop effect. [O'Rourke, 2009]

4.2 Audio Elements

The HTML5 specification offers an <audio> tag with an automatic fallback mechanism. A simple html code looks like the example in listing 4.1. The user agent is expected to go trough the list of source elements and use the first one that it supports.

```
<audio controls="controls">
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>
```

Listing 4.1: HTML5 Audio Element Example[*W3Schools Audio Tag*]

Supported formats so far are mp3, wav and ogg. Combining mp3 and either ogg or wav provides full browser support coverage of major browsers [*W3Schools Audio Tag*].

By including another fallback mechanism in the body of the <audio> tag but outside and after the source tags it is possible to fallback again if the default fallback runs trough without success. This fallback can be a flash player or any other mechanism like a pure javascript player. you can see an example of this in listing 4.2.

```
<audio id="audio_with_controls" controls>
  <source src="test.mp3" type="audio/mpeg" />

  <object class="playerpreview" type="application/x-shockwave-flash"
          data="player_mp3_mini.swf" width="200" height="20">
    <param name="movie" value="player_mp3_mini.swf" />
    <param name="bgcolor" value="#085c68" />
    <param name="FlashVars" value="mp3=test.mp3" />
    <embed href="player_mp3_mini.swf" bgcolor="#085c68" width="200"
           height="20" name="movie" align=""
           type="application/x-shockwave-flash" flashvars="mp3=test.mp3">
    </embed>
  </object>

</audio>
```

```
<div id="player_fallback"></div>
<script>
  if (document.createElement('audio').canPlayType) {
    if (!document.createElement('audio').canPlayType('audio/mpeg')) {
      ... SWFObject script line here ...
      document.getElementsById('audio_with_controls').style.display = 'none';
    }
  }
</script>
```

Listing 4.2: HTML5 Audio Element Advanced Example[*HTML5Rocks Audio Element Guide*]

Audio elements have default controls and style, but can be replaced entirely and restyled completely while preserving compatibility and function. There are JavaScript Libraries that offer custom controls and automatic fallback handling to flash. One example is the jPlayer JS library which is completely free and open source [*jPlayer*].

4.3 Videos

4.3.1 Responsive Videos

Videos are becoming an important marketing tool on many websites, therefore the need to incorporate responsive videos into designs is growing dramatically. Responsive videos are elastic and are especially favored where web pages will be viewed on different screen sizes using a variety of browsers. Until now the playing of a video was either with a Flash player or with Quick Time, of which both relied on third party plugins. Since most of the web is based on video sharing and third party plugins always slow down the page, therefore making browsers slower, the new "video" tag in HTML5 proved to be quite useful. [Developer Drive, 2012]

*When handling video embed code that uses iframes and objects tags, using the HTML5 video element, however is not enough, therefore it will not work for video found on most video sharing sites like YouTube and Vimeo. In order to tackle this, it is needed to embed the code with a container and specify the child elements with absolute positions. In this case, give 100% to both width and height, thereby forcing the embed elements to automatically expand full width. [Web Designer Wall, 2011]

```
video {max-width: 100%;height: auto;}
```

4.3.2 Html5 video players

There are 24 different html5 video players known so far, with each having different features, specialties and approaches. Knowing the differences between them is important to be able to choose the best HTML5 based video player possible in accordance to the needs of a project. Some of the popular among them are: [VideoSWS, 2012]

Video for Everybody

Video for Everybody is HTML code that embeds a video into a website using the HTML5 <video> element, falling back to Flash automatically without the use of JavaScript or browser-sniffing. Case they both fail, a placeholder image is shown and the user can download the video using the links provided.
Everything is completely done without JavaScript and requires two video encodes, one Ogg file, and one MP4 file, however if the user does not have Flash they are not prompted to install it. [CamenDesign, CD2010]

Video.js

Video.js is a JavaScript and CSS library which makes easier to work with and build on HTML5 video. Video.js provides common controls built in HTML/CSS, fixes cross-browser inconsistencies, adds additional features

like fullscreen and subtitles, manages the fallback to Flash or other playback technologies when HTML5 video isn't supported, and also provides a consistent JavaScript API for interacting with the video. [VideoJS, 2012]

MediaElement.js

Provides the same user interface on any browser using HTML5 audio and video players in pure HTML and CSS as well as Custom Flash and Silverlight players that mimic the HTML5 MediaElement API for older browsers giving the user consistent experience regardless of what codecs and plugins their browser supports. [MediaElements.js, 2012]

HTML5 Video

HTML5video.org is an online community of web developers and the home of the Kaltura open source video player javascript library. HTML5 Enables Web Pages to playback and manipulate video and audio across platforms and devices - powering amazing rich-media applications that work everywhere. [Html5Video.org, 2012]

Chapter 5

Feature Specific Responsive Content

Usually the main focus area of responsive design is the width and height of the Viewport (Browser Window) and the problem of fitting and reordering the content to fill it appropriately. Worded differently it is the handling of lack or abundance of space or the shape of it. But Viewport space properties are only one aspect of the display device capabilities which may impact the experience of the user. There are other features which may be worth considering when designing the appearance or usability of your website.

5.1 Physical Input Devices

One important device feature which may influence your design decisions are the capabilities of input controls. The W3C has proposed new queries for the fourth edition of the Media Queries specification[*Media Queries Level 4 Draft*] to address the availability and properties of these.

5.1.1 Media Query: *pointer*

This Media Query allows to detect the presence and the accuracy of an attached pointer device. This Query can use three different values: 'none', 'coarse' and 'fine'.

> Both 'coarse' and 'fine' indicate the presence of a pointing device, but differ in accuracy. A pointing device with which it would be difficult or impossible to reliably pick one of several small adjacent targets would qualify as 'coarse'.
> Typical examples of a 'fine' pointing system are a mouse, a track-pad or a stylus-based touch screen. Finger-based touch screens would qualify as 'coarse'.

The specification contains a provision for the user agent to give a value of 'coarse' even if an input device is present which can be considered 'fine' but the user may have indicated that they have trouble controlling the input device accurately.

5.1.2 Media Query: *hover*

This media query allows for detection of a input device that allows hovering over elements to display information, navigational cues or other information or design changes. According to the W3C the user agent should always return the value which is indicative of the *least* capable device attached.

> If a device has multiple pointing devices, some of which support hovering and some of which not, it is recommended that the UA reports the hovering ability of the least capable of the primary pointing devices.

Additionaly, the designer may not assume that a device which returns '0' does not display the pseudoclass *:hover* but only that it requests a version of the website which is usable without the hover mechanic. Again, the UA may return '0' contrary to device capability to suit the accesibility needs of the user.

5.1.3 Zooming with Touch Gestures

One special point that needs to be considered when thinking about input devices is that touch devices do not only have limitations but often in contrast allow the user to do a few special actions with your page which they can not do with a traditional PC device. The most prominent of those is the possibility to *zoom* into your page without actually changing the viewport or font scaling.

One functionality which is often called *pinch to zoom* may provide the user of a close-up view of web page elements that are not designed to be viewed that way. This is one of the cases in which vector images have a clear advantage because of the way they behave when scaled by the client.

Another functionality is called *double-tap zoom* and aligns the zoom level to the size of the *tapped* element. Sizing your website elements properly to take advantage of this might make your website more usable.

5.2 Alternative Properties of Display Screens

Besides the default properties already well known and used like width, height, device-width, device-height and orientation there are few display device features which are worth mentioning and might affect user experience.

5.2.1 Media Query: *resolution* and Display Resolution (Pixel Density)

Using the case of the current introduction of *Retina* screens to the consumer PC product line of Apple Computers but also the ever-increasing pixel density on small devices like smartphones, one should consider the actual space that website elements take up on the display device. The W3C MQ4 Draft proposes the query *resolution* to aid in detecting this property.

While the iPhone 3GS had a PPI (Pixel per Inch, often incorrectly DPI, dots per inch) of 163 its successor, the iPhone 4 had 326 PPI. This means that an image or website element that displayed on 1 inch of actual surface on the iPhone 3 would only be $\frac{1}{4}$ of an inch big on the iPhone 4. Same is true for the laptop computers or home PCs with similar differences in PPI. On the other hand designers might want to use higher resolution images and icons because the device is actually capable of displaying them.

Website designers might consider making their breakpoints dependent on more than pure pixel widths and take into account the pixel density of the target device.

5.2.2 Media Queries: *monochrome* and *luminosity*

Another one of the features in the upcoming Media Queries 4 Draft is *luminosity* which overlaps somewhat with the already implemented query *monochrome*.

Monochrome allows for detection of devices with monochrome displays. Those are displays that are only capable of displaying shades of one color. A recent example are all kinds of eReader devices with e-Paper or e-Ink displays. Those displays offer great contrast but lack the ability to display color information.

The Query *luminosity* allows for communicating the brightness of the environment in which the display device operates. It does so in three levels called 'dim', 'normal' and 'washed' in increasing order of brightness. The W3C does however avoid defining exact brightness in lux as the value might change according to already present automatic brightness adjustment of the screen and other factors.

Both of these queries prompt the website to offer high-contrast versions that are either usable in environments or devices which are not hospitable to nuanced color design or might even be blindingly bright in a dimly lit room.

5.3 Providing Context for Input Fields

Everything up to this point was about *reacting* to the features and properties of the device which is displaying our website. But there is also a way in which a designer might help the user device provide additional flexibility

and usability by proactively declaring potential to do so. A very good way to do that is the usage of HTML5 input type declarations.

Many websites offer some functionality which prompts the user for input and expects a certain format of that input. By using HTML5 input types the user device can offer customized input controls, particularily on touchscreen devices with virtual keyboards which can be customized easily but also to some degree on other devices. Possible tags are seen in listing 5.1.

Additionaly, devices might offer automatic input format validation and aid the user in complying with the required format of the input field.

```
<input type="color">
<input type="date">
<input type="datetime">
<input type="datetime-local">
<input type="email">
<input type="month">
<input type="number">
<input type="range">
<input type="search">
<input type="tel">
<input type="time">
<input type="url">
<input type="week">
```

Listing 5.1: HTML5 input fields with type declaration[W3C, 2012a]

5.3.1 Custom Input Elements

One of the browsers providing custom input controls is Google Chrome. A few examples can be seen in figure 5.1 which also features a triggered form validation failure message. Although the same browser is used for displaying the date picker control in figure 5.2 the browser displays entirely different controls. For the desktop computer with a 'fine' accuracy pointer device the browser displays a calendar grid. For the touch device though it displays three rolling wheels, one for each continous part of the input value. This enhanced functionality does not require any additional effort from the website designer and integrates seamlessly into the device usability paradigms.

5.3.2 Adaptable Virtual Inputs

A different form of providing better inputs is the adaptation of virtual controls like keyboards by introducing subtle but important changes. Offering additional buttons or offering a whole new set of keys that are preselected based on the input type can greatly speed up the input and at the same time minimize frustration. In figure 5.3 you can see the different keyboard versions offered up by the iPhone when focusing input fields of different types.

5.3.3 Custom Browser Actions on Non-Input Elements

Input elements are not the only html elements that may provide ways for the display device to react in an appropriate way. One additional way is a custom protocol definition in an <a> tag href attribute as seen in listing 5.2. When the resulting link is clicked on a modern mobile phone with iOS or Android OS it automatically switches th the call application and copies the linked number, effectively enabling calling with two clicks.

```
<a href="tel:555-1212">Call</a>
```

Listing 5.2: Custom HREF link

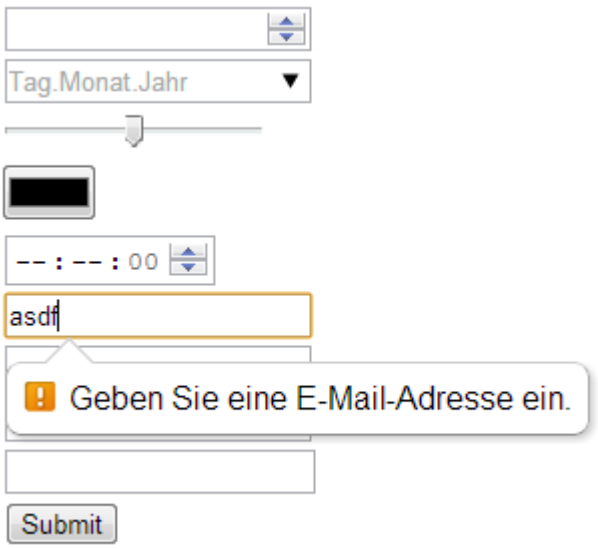


Figure 5.1: Custom Input Controls and Form Validation (Chrome, Windows 7)

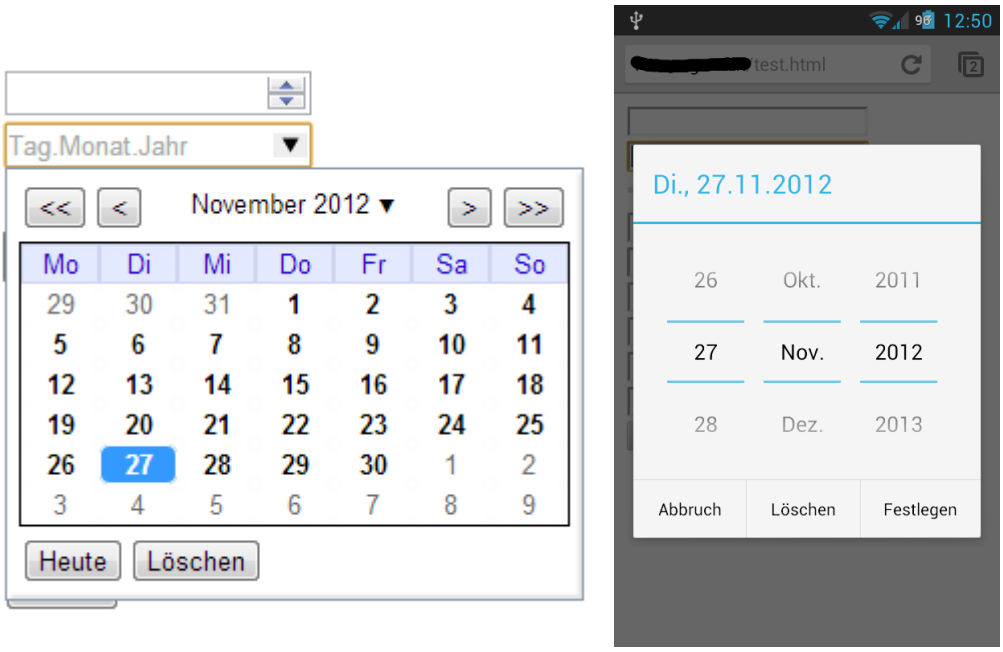


Figure 5.2: Date Input Controls (Chrome, Windows 7 and Android)

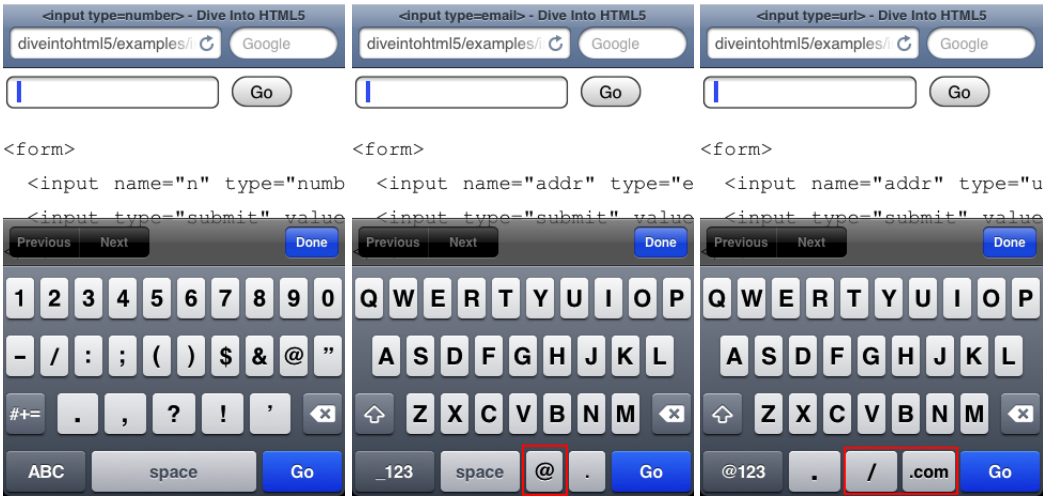


Figure 5.3: Keyboard Adaptation based on Input Field Type (Safari, iPhone)

5.4 HTML5 APIs

There are a number of HTML5 APIs that can be made available trough the browser and may offer interesting functionality to the website.

5.4.1 WebAudio API

This API offers Audio Services like Text-To-Speech over device loudspeakers and Speech-Recognition trough a device microphone. It also offers direct manipulation of audio streams.

5.4.2 Geolocation API

This API offers locating the users current position at any time trough usage of a devices built in GPS antenna and network provider supported triangulation information.

5.4.3 Rotation API

This API offers the ability to detect positional changes in a device's and acceleration values.

5.5 Detecting Available Features

Most of the features which are not yet covered my media queries (e.g. geolocation) can be detected by using JavaScript workarounds which work as follows:

1. Use API / Create Element
2. Check for Success
3. If success → load ressources / execute enhancement
4. If failure → load / execute fallback

Fortunately there are JavaSCript libraries which take care of all the details and offer a simple API for the end user. One very popular and well maintained library is Modernizr (`http://modernizr.com`) which can be used by including code as seen in listing 5.3. If the feature defined in 'test' is found to be available then the sourcecode defined in 'yep' is executed. If the feature is lacking, then the code found at 'nope' is executed instead.

```
Modernizr.load({
  test: Modernizr.geolocation,
  yep : 'geo.js',
  nope: 'geo-polyfill.js'
});
```

Listing 5.3: Modernizr Example[*Modernizr Documentation*]

Chapter 6

Conclusion

We can see that the idea of responsive web design is still in an early development state. Still we conclude it to be the best choice for platform independent and future proof web design now. There is a certain lack of golden rules to achieve proper web design. Today it is pretty much up to any single developer to decide how many break points or actual layouts to use.

Media queries are there and officially supported by many browsers. An increasing number of websites is relying on them. Still they are only a technological necessity to responsive web design and are just part of a bigger solution to classic web design issues in the context of mobile computing.

Our field research shows that there are several common responsive patterns evolving which even relate to specific website elements. An increased number of websites and better categorical analysis could deliver more distinct and comprehensive results. But this would have been out of reach of this survey.

Today there is a lack of officially supported and standardised ways to achieve a vital part of responsive web design: responsive multimedia content. As explained in chapter 4 working with images can be quite a hassle when relying on plain web standards HTML and CSS. In many situations only appropriate JavaScript modules and server side rollout can deliver good responsive concepts.

Chapter 5 introduces several elements that can not only improve basic web design but enhance our overall web experience by transforming classic rendered web contents into interactive applications with support for special input devices, geo location and other features.

Bibliography

artisttechnewmedia.com [2012]. *Responsive Web Design Image*. Last checked: December 3rd. 2012. http://www.artistechnewmedia.com/thenest/wp-content/uploads/2012/03/responsive-web-design.jpg (cited on page 5).

Australian Web Advice and Policy Team [2012]. *Common page elements*. Last checked: December 2nd. 2012. http://webguide.gov.au/finding-content/page-elements/ (cited on page 7).

Bowlby, Selene M. [2008]. *15 Key Elements All Top Web Sites Should Have*. Last checked: December 2nd. 2008. http://freelancefolder.com/15-top-site-elements/ (cited on page 7).

CamenDesign [CD2010]. *Video for Everybody!* Last checked: December 3rd. CD2010. http://camendesign.com/code/video_for_everybody#video-what (cited on page 15).

CSS-Portal [2012]. *CSS crop Property*. Last checked: December 2nd. 2012. http://www.cssportal.com/css-properties/crop.htm (cited on page 13).

Developer Drive [2012]. *Adding Responsive Videos to your Design*. Last checked: December 3rd. 2012. http://www.developerdrive.com/2012/07/adding-responsive-videos-to-your-design (cited on page 15).

Gallagher, Nicolas [2011]. *Responsive Images Using CSS3*. Last checked: December 2nd. 2011. http://nicolasgallagher.com/responsive-images-using-css3/ (cited on page 12).

HTML5Rocks Audio Element Guide. http://www.html5rocks.com/en/tutorials/audio/quick. http://www.html5rocks.com/en/tutorials/audio/quick (cited on page 15).

Html5Video.org [2012]. *HTML5 Video*. Last checked: December 3rd. 2012. http://html5video.org (cited on page 16).

. *jPlayer*. http://www.jplayer.org/. http://www.jplayer.org/ (cited on page 15).

http://www.w3schools.com. *W3Schools Audio Tag*. http://www.w3schools.com/html/html5_audio.asp. http://www.w3schools.com/html/html5_audio.asp (cited on page 14).

Imbong, John [2009]. *The Definitive Guide to Using Negative Margins*. Last checked: December 2nd. 2009. http://coding.smashingmagazine.com/2009/07/27/the-definitive-guide-to-using-negative-margins/ (cited on page 14).

JavaScriptKit [2012]. *Introduction to CSS Media Queries*. Last checked: December 3rd. 2012. http://www.javascriptkit.com/dhtmltutors/cssmediaqueries.shtml (cited on page 3).

Jehl, Scott [2010]. *Responsive Images: Experimenting with Context-Aware Image Sizing*. Last checked: December 2nd. 2010. http://filamentgroup.com/lab/responsive_images_experimenting_with_context_aware_image_sizing/ (cited on page 13).

Johansson, Roger [2007]. *Poll results: 50.4 percent of respondents maximise windows*. Last checked: December 3rd. 2007. http://www.456bereastreet.com/archive/200704/poll_results_504_of_%20respondents_maximise_windows (cited on pages 5, 6).

Koch, Peter-Paul [2011]. *Mobile Browsers*. Last checked: December 3rd. 2011. http://www.quirksmode.org/mobile/browsers.html (cited on page 5).

Koch, Peter-Paul [2012]. *Mobile market overview*. Last checked: December 3rd. 2012. http://www.quirksmode.org/mobile/mobilemarket.html (cited on page 5).

Lazaris, Louis [2012]. *CSS3 Transitions Without Using :hover*. Last checked: December 2nd. 2012. http://www.impressivewebs.com/css3-transitions-without-hover/ (cited on page 8).

Marcotte, Ethan [2011]. *Responsive Web Design*. Edited by Mandy Brown. Jeffrey Zeldman, 2011 (cited on pages 1–3, 7, 9, 12, 13).

MediaElements.js [2012]. *HTML5 video and audio made easy. One file. Any browser. Same UI.* Last checked: December 3rd. 2012. http://mediaelementjs.com/ (cited on page 16).

Modernizr Documentation. http://modernizr.com/docs/ (cited on page 22).

O'Rourke, Robert [2009]. *CSS Display an Image Resized and Cropped*. Last checked: December 2nd. 2009. http://stackoverflow.com/questions/493296/css-display-an-image-resized-and-cropped (cited on page 14).

Polacek, John [2012]. *What The Heck Is Responsive Web Design?* Last checked: December 2nd. 2012. http://johnpolacek.github.com/scrolldeck.js/decks/responsive/ (cited on page 1).

rICG [2012]. *The Picture Element - An HTML Extension for Adaptive Images*. Last checked: December 2nd. 2012. http://picture.responsiveimages.org (cited on page 12).

@rivoal.net>, Florian Rivoal <florian. *Media Queries Level 4 Draft*. http://dev.w3.org/csswg/mediaqueries4 (cited on page 17).

Roberts, Harry [2011]. *Responsive images right now*. Last checked: December 2nd. 2011. http://csswizardry.com/2011/07/responsive-images-right-now/ (cited on page 12).

Russel, Craig [2010]. *Responsive Images and Context Aware Image Sizing*. Last checked: December 2nd. 2010. http://craig-russell.co.uk/2011/01/22/responsive-images-and-context-aware-image-sizing.html#.ULvQt6U7018 (cited on page 13).

Sender 11 [2011]. *Mobile screen size trends*. Last checked: December 3rd. 2011. http://sender11.typepad.com/sender11/2008/04/mobile-screen-%20http://sender11.typepad.com/sender11/2008/04/mobile-screen-%20http://sender11.typepad.com/sender11/2008/04/mobile-screen-s.html (cited on page 4).

Thomas, John [2009]. *The Difference – Websites vs. Web Pages*. Last checked: December 2nd. 2009. http://innovationsimple.com/web-design/website-vs-webpage/ (cited on page 7).

Uggedal, Eivind, editor [2012]. *A collection of inspirational websites using media queries and responsive web design*. Last checked: December 2nd. 2012. http://mediaqueri.es (cited on page 7).

VideoJS [2012]. *HTML5 Video, Now Available Everywhere*. Last checked: December 3rd. 2012. http://videojs.com/ (cited on page 16).

VideoSWS [2012]. *HTML5 Video Player Comparison*. Last checked: December 3rd. 2012. http://praegnanz.de/html5video/ (cited on page 15).

Viklund, Andreas [2011]. *Graceful degradation vs. Progressive enhancement (part 1)*. Last checked: December 2nd. 2011. http://andreasviklund.com/learn/graceful-degradation-vs-progressive-enhancement-part-1/ (cited on page 7).

W3C [2012a]. *HTML5 Input Controls Working Draft*. http://www.w3.org/TR/html-markup/input.html. 2012 (cited on page 19).

W3C [2012b]. *Recognized media types*. Last checked: December 3rd. 2012. http://www.w3.org/TR/CSS21/media.html#media-types (cited on page 2).

W3Schools.com [2012]. *CSS clip Property*. Last checked: December 2nd. 2012. http://www.w3schools.com/cssref/pr_pos_clip.asp (cited on page 13).

Web Designer Wall [2011]. *CSS: Elastic Videos*. Last checked: December 3rd. 2011. http://webdesignerwall.com/tutorials/css-elastic-videos (cited on page 15).

Web Page Mistakes [2008]. *Website Navigation*. Last checked: December 2nd. 2008. http://www.webpagemistakes.ca/website-navigation/ (cited on page 8).

Appendix A
Pattern Analysis Data Tables

Navigation	%	#	Websites
horizontal list -> vertical list	0.5	12	sony.com, systemagic.co.uk, lottanieminen.com, dorigati.it, aids.gov, getdonedone.com, microsoft.com, bigyouth.fr, wentworthmansion.com, unitedutilities.com, wm.edu, hsgac.senate.gov
collapse main section into single menu	0.5	10	sony.com, systemagic.co.uk, dorigati.it, aids.gov, getdonedone.com, microsoft.com, bigyouth.fr, wentworthmansion.com, wm.edu, hsgac.senate.gov
contained in fluid box	0.2	4	paidtoexist.com, systemagic.co.uk, vittoriovittori.com, wentworthmansion.com
2D grid	0.1	3	tuj.ac.jp, , wm.edu, atlantaballet.com
remove features	0.1	2	vittoriovittori.com, atlantaballet.com
horizontal list -> native dropdown list	0.0	1	iso.org

Content		#	Websites
responsive grid	0.8182	18	sony.com, iso.org, evening-edition.com, systemagic.co.uk, vittoriovittori.com, lottanieminen.com, dorigati.it, aids.gov, microsoft.com, keynesforkids.com, responsiveprocess.com, bigyouth.fr, wentworthmansion.com, tuj.ac.jp, introducingthenovel.com, wm.edu, hsgac.senate.gov, atlantaballet.com
degrade media elements	0.6818	15	paidtoexist.com, lottanieminen.com, dorigati.it, aids.gov, getdonedone.com, microsoft.com, keynesforkids.com, responsiveprocess.com, bigyouth.fr, wentworthmansion.com, tuj.ac.jp, introducingthenovel.com, wm.edu, hsgac.senate.gov, atlantaballet.com
fluid boxes	0.8182	18	iso.org, paidtoexist.com, evening-edition.com, systemagic.co.uk, vittoriovittori.com, dorigati.it, aids.gov, getdonedone.com microsoft.com, keynesforkids.com, responsiveprocess.com, bigyouth.fr, wentworthmansion.com, unitedutilities.com, introducingthenovel.com, wm.edu, hsgac.senate.gov, atlantaballet.com

Header		#	Websites
degrade media elements	0.7727	17	sony.com, iso.org, evening-edition.com, systemagic.co.uk, dorigati.it, aids.gov, getdonedone.com, oxideinteractive.com.au, microsoft.com, keynesforkids.com, responsiveprocess.com, bigyouth.fr, wentworthmansion.com, tuj.ac.jp, introducingthenovel.com, hsgac.senate.gov, atlantaballet.com
completely rearrange items	0.3636	8	sony.com, iso.org, systemagic.co.uk, dorigati.it, aids.gov, microsoft.com, unitedutilities.com, atlantaballet.com
remove features	0.3636	8	iso.org, dorigati.it, aids.gov, oxideinteractive.com.au, bigyouth.fr, unitedutilities.com, wm.edu, hsgac.senate.gov
responsive grid	0.2273	5	paidtoexist.com, vittoriovittori.com, wentworthmansion.com, tuj.ac.jp, wm.edu

Footer		#	Websites
responsive grid	0.5909	13	systemagic.co.uk, vittoriovittori.com, dorigati.it, aids.gov, getdonedone.com, microsoft.com, keynesforkids.com, bigyouth.fr, wentworthmansion.com, tuj.ac.jp, wm.edu, hsgac.senate.gov, atlantaballet.com
fluid boxes	0.5909	13	iso.org, paidtoexist.com, evening-edition.com, systemagic.co.uk, vittoriovittori.com, dorigati.it, aids.gov, microsoft.com, keynesforkids.com, bigyouth.fr, wentworthmansion.com, tuj.ac.jp, introducingthenovel.com
horizontal lists -> vertical lists	0.2273	5	sony.com, getdonedone.com, wentworthmansion.com, unitedutilities.com, atlantaballet.com
remove featuers	0.1364	3	systemagic.co.uk, dorigati.it, hsgac.senate.gov
vertical lists -> horizontal lists	0.0455	1	aids.gov

Major Breakpoints		#	Websites
1	0.1818	4	evening-edition.com, getdonedone.com, keynesforkids.com, unitedutilities.com
2	0.4545	10	sony.com, lottanieminen.com, dorigati.it, oxideinteractive.com.au, responsiveprocess.com, wentworthmansion.com, bigyouth.fr, tuj.ac.jp, introducingthenovel.com, wm.edu
3	0.2727	6	paidtoexist.com, vittoriovittori.com, aids.gov, microsoft.com, hsgac.senate.gov, atlantaballet.com
4	0.0455	1	iso.org
5	0.0455	1	systemagic.co.uk

Websites	#	
All	22	sony.com, iso.org, systemagic.co.uk, vittoriovittory.com, paidtoexist.com, evening-edition.com, lottanieminen.com, dorigati.it, aids.gov, getdonedone.com, oxideinteractive.com.au, microsoft.com, keynesforkids.com, responsiveprocess.com, bigyouth.fr, wentworthmansion.com, unitedutilities.com, tuj.ac.jp, introducingthenovel.com, wm.edu, hsgac.senate.gov, atlantaballet.com
Uses block typesetting	1	introducingthenovel.com

Vertical Size Factor Max width -> 400px (only if there is a maximum width within 2560px)	#	
0,5-1	2	oxideinteractive.com.au, unitedutilities.com
1-1.5	2	systemagic.com, paidtoexist.com
1.5-2	17	wm.edu, sony.com, iso.org, vittoriovittori.com, evening-edition.com, dorigati.it, aids.gov, getdonedone.com, microsoft.com, keynesforkids.com, responsiveprocess.com, bigyouth.fr, wentworthmansion.com, tuj.ac.jp, introducingthenovel.com, hsgac.senate.gov, atlantaballet.com