



Arkaitz Garro

---

# Responsive Web Design



## **Responsive Web Design**

**Publication date:** 14/06/2013

This book was published with *easybook v5.0-DEV*, a free and open-source book publishing application developed by Javier Eguiluz (<http://javiereguiluz.com>) using several Symfony components (<http://components.symfony.com>) .



Esta obra se publica bajo la licencia *Creative Commons Reconocimiento - No Comercial - Compartir Igual 3.0*, cuyos detalles puedes consultar en <http://creativecommons.org/licenses/by-nc-sa/3.0/es/>.

Puedes copiar, distribuir y comunicar públicamente la obra, incluso transformándola, siempre que cumplas todas las condiciones siguientes:

- Reconocimiento: debes reconocer siempre la autoría de la obra original, indicando tanto el nombre del autor (Arkaitz Garro) como el nombre del sitio donde se publicó originalmente [www.arkaitzgarro.com](http://www.arkaitzgarro.com). Este reconocimiento no debe hacerse de una manera que sugiera que el autor o el sitio apoyan el uso que haces de su obra.
- No comercial: no puedes utilizar esta obra con fines comerciales de ningún tipo. Entre otros, no puedes vender esta obra bajo ningún concepto y tampoco puedes publicar estos contenidos en sitios web que incluyan publicidad de cualquier tipo.
- Compartir igual: si alteras o transformas esta obra o si realizas una obra derivada, debes compartir tu trabajo obligatoriamente bajo esta misma licencia.



<b>Capítulo 1 Introducción.....</b>	<b>9</b>
1.1 Ventajas .....	10
1.2 Ejemplos de sitios creados aplicando RWD.....	10
1.3 Los ingredientes .....	12
<b>Capítulo 2 Estructura de rejilla flexible .....</b>	<b>13</b>
2.1 Crear una rejilla flexible.....	14
<b>Capítulo 3 Imágenes flexibles .....</b>	<b>21</b>
3.1 Imágenes fluidas.....	23
3.2 Imágenes de fondo .....	25
<b>Capítulo 4 Media Queries .....</b>	<b>29</b>
4.1 Sintaxis .....	30
4.2 Viewport .....	34
<b>Capítulo 5 Desarrollo adaptativo .....</b>	<b>35</b>
5.1 La importancia del contexto .....	35
5.2 Hacia un flujo de trabajo adaptativo.....	36
5.3 Puntos de ruptura .....	37
5.4 Desarrollo iterativo .....	38
5.5 Desarrollo adaptativo, de manera responsable .....	39
<b>Capítulo 6 Ejercicio Responsive Web Design.....</b>	<b>45</b>



Responsive Web Design (o diseño web adaptativo) se trata de una técnica de diseño y desarrollo web por el que se consigue que un único sitio se adapte perfectamente a todos los dispositivos que puedan consumirlo, desde ordenadores de escritorio a netbooks, tablets, teléfonos móviles, televisores, etc. En definitiva, se trata de construir una única web para que se vea correctamente y aproveche las particularidades de todo dispositivo que hoy exista, o pueda existir en el futuro, independientemente de la pantalla en la que se muestre.

Tanto la idea y como el propósito del diseño web adaptativo fueron previamente discutidos y descritos por el consorcio W3C en julio de 2008 en su recomendación *Mobile Web Best Practices* (<http://arborwebsolutions.com/articles/responsive-design-is-an-official-w3c-recommendation>) bajo el subtítulo One Web. Dicha recomendación, aunque específica para dispositivos móviles, puntuiza que está hecha en el contexto de "One Web" (<http://www.w3.org/TR/mobile-bp/#OneWeb>), y que por lo tanto engloba no solo la experiencia de navegación en dispositivos móviles sino también en dispositivos de mayor resolución de pantalla como dispositivos de sobremesa. El concepto de "One Web" hace referencia a la idea de construir una Web para todos (Web for All) y accesible desde cualquier tipo de dispositivo (Web on Everything). Hoy en día, la variedad de dispositivos existente en el mercado ha provocado que la información disponible no sea accesible desde todos los dispositivos, o bien es accesible pero la experiencia de navegación es muy pobre.

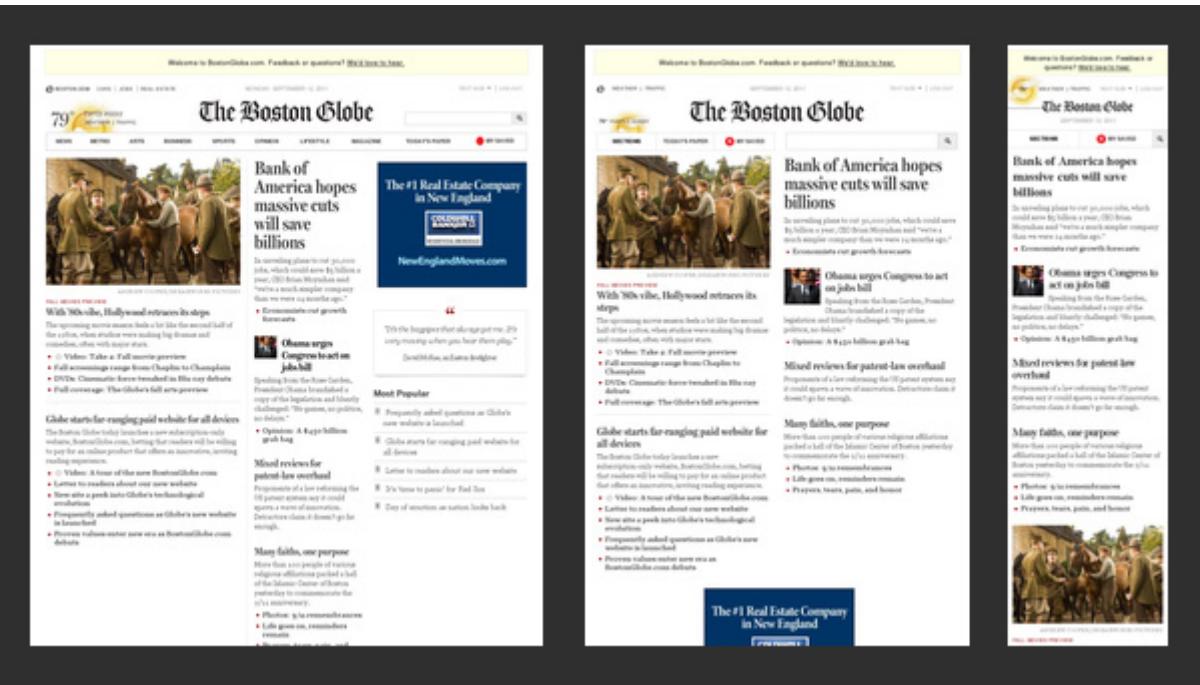
---

Las ventajas de utilizar un diseño web adaptativo son más que evidentes:

- Con **una sola versión** en HTML y CSS se cubren todas las resoluciones de pantalla, es decir, el sitio web creado estará optimizado para todo tipo de dispositivos: PCs, tabletas, teléfonos móviles, etc. Esto mejora la experiencia de usuario a diferencia de lo que ocurre, por ejemplo, con sitios web de ancho fijo cuando se acceden desde dispositivos móviles.
- **Reducción de costos.** Se reducen los costes ya que hasta hoy se debe hacer un portal para la Web y otro para dispositivos móviles. Esto origina mayores costos de creación y mantenimiento de la información.
- **Eficiencia en la actualización.** El sitio solo se debe actualizar una vez y se ve reflejada en todas las plataformas. Cuando tenemos los portales independientes para Web y Mobile se debe realizar la actualización dos veces lo que crea la necesidad de mayor cantidad de recursos y posibilidad de error.
- Desde el punto de vista de la **optimización de motores de búsqueda**, sólo aparecería una URL en los resultados de búsqueda, con lo cual se ahorran redirecciones y los fallos que se derivan de éstas. También se evitarían errores al acceder al sitio web en concreto desde los llamados *social links*, es decir, desde enlaces que los usuarios comparten en medios sociales tales como Facebook, Twitter, etc y que pueden acabar en error dependiendo de qué enlace se copió (desde qué dispositivo se copió) y desde qué dispositivo se accede.

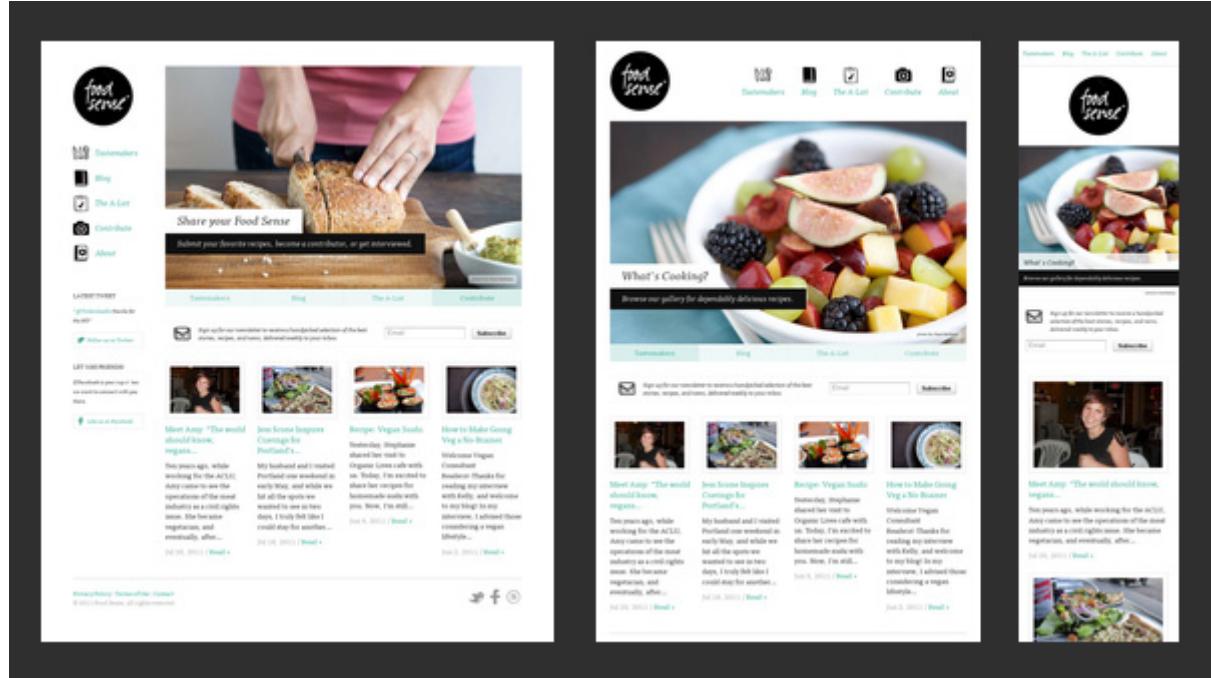
En un artículo llamado: Responsive Web Design: 50 Examples and Best Practices (<http://designmodo.com/responsive-design-examples/>) muestra excelentes ejemplos de la aplicación de esta tecnología. Aquí una pequeña muestra de alguno de ellos:

Boston Globe (<http://www.bostonglobe.com/>)



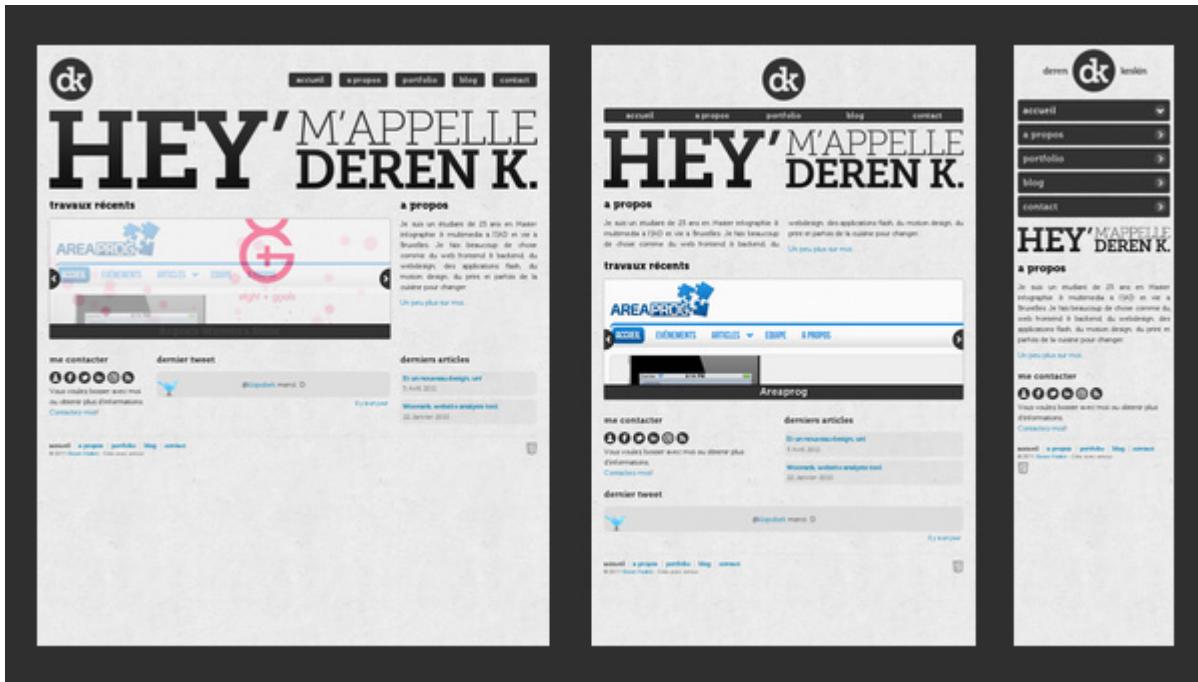
**Figura 1.1** Diario Boston Globe utilizando un diseño adaptativo

## Food Sense (<http://foodsense.is/>)



**Figura 1.2** La web Food Sense utilizando un diseño adaptativo

## Deren keskin (<http://www.deren.me/>)



**Figura 1.3** La web Deren keskin utilizando un diseño adaptativo

Los elementos básicos necesarios para construir un diseño adaptativo, son al menos los tres siguientes:

- Una estructura flexible, basada en rejilla.
- Imágenes y multimedia flexible.
- *Media queries*, como parte de la especificación de CSS 3.

En los próximos capítulos crearemos un diseño adaptativo que se comporte de manera diferente según el dispositivo a través del cual se accede a la web, y las *necesidades* del usuario.

Los diseños basados en rejilla han sido utilizados ampliamente en sectores como el diseño gráfico, pero tardaron en ser adoptados por la web. Estos diseños consisten en una serie de columnas que permiten ordenar y estructurar los contenidos (texto, imágenes, media...) de una manera ordenada y uniforme. Una estructura clásica de rejilla es la que se muestra en la siguiente imagen:



**Figura 2.1** Estructura de rejilla utilizada para tipografía

---

Se pueden observar claramente las líneas verticales y horizontales utilizadas para posicionar los elementos en el lienzo, así como los márgenes que separan los distintos elementos. De esta manera tan simple, es posible crear diseños con un aspecto visual uniforme.

Mientras que los `media query` ofrecen la potencia real para desarrollar un sitio web adaptativo, es posible ahorrar trabajo y código utilizando una aproximación en base a rejillas flexibles. Utilizando este tipo de rejillas, podemos asegurar que el sitio web se redimensiona en función del espacio disponible, sin la necesidad de utilizar `media query`. Posteriormente es posible utilizar `media query` si es necesario realizar cambios significativos en la estructura de la web.

Vamos a ver cinco componentes que nos permitirán construir una rejilla flexible, y cómo utilizarlas:

- Fuentes flexibles
- Contenedores flexibles
- Márgenes flexibles
- Rellenos flexibles
- Imágenes flexibles

El método tradicional utilizado para dar estilo a las fuentes es utilizar pixeles. Esto da un control exacto a los diseñadores a la hora de definir su tamaño, pero lo ideal es comenzar a pesar de manera proporcional. Utiliza valores proporcionales a la hora de definir los tamaños del texto, nos ahorrará mucho tiempo y líneas de código en nuestros `media query`. Si deseamos aumentar o reducir el tamaño de nuestro texto, con indicar el tamaño de letra del elemento padre sería suficiente, ya que el resto de elementos se verían afectados de manera inmediata.

La manera correcta de conseguir este comportamiento es utilizar medidas basadas en `em`. Además, debemos aplicar una regla muy sencilla para calcular el tamaño de letra concreto para cada caso:

$$\text{target} \div \text{context} = \text{result}$$

---

Donde **target** es el tamaño de fuente que queremos mostrar y **context** es el tamaño de fuente base de nuestro navegador (generalmente son 16px para la mayoría de navegadores).

Así pues, si deseamos mostrar un texto con un tamaño de 32px, su tamaño en **em** se calcularía de la siguiente manera:

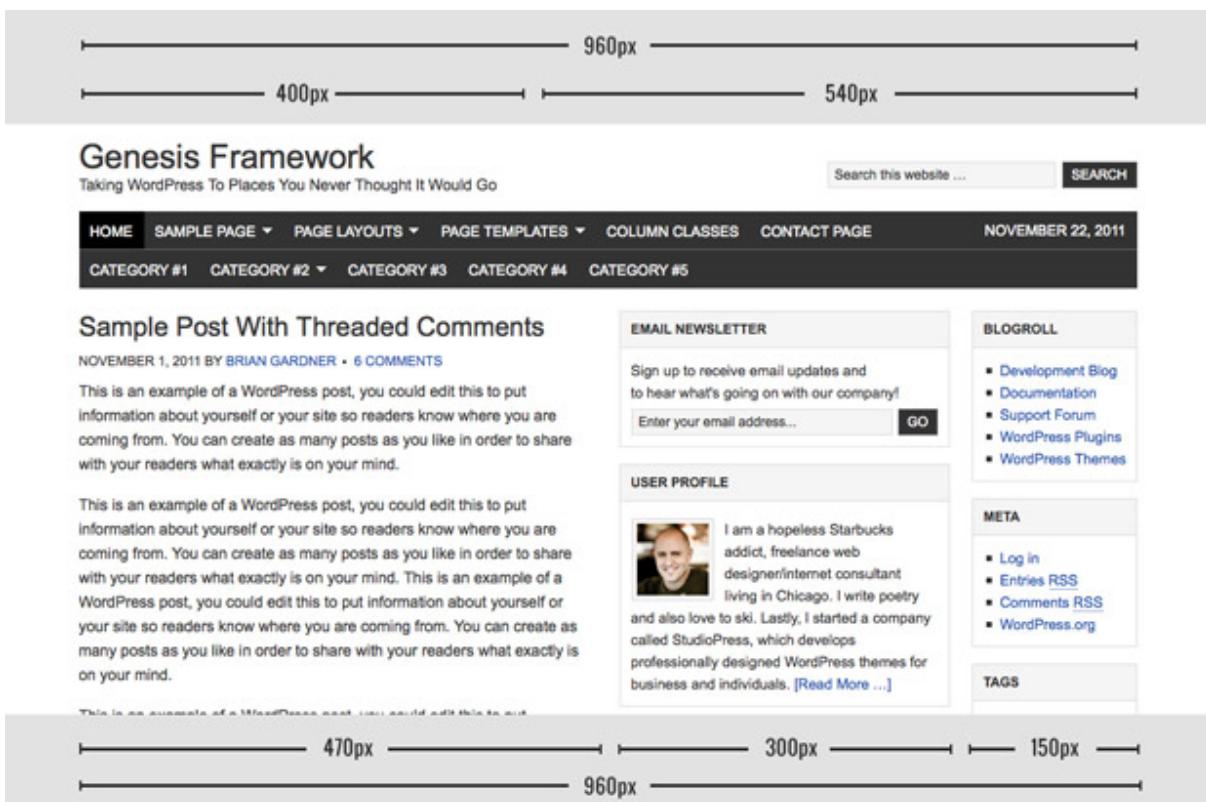
$$32\text{px} \div 16\text{px} = 2$$

Al utilizar esta fórmula, lo normal es obtener un número con mucho decimales. Lo ideal es mantener estos decimales, ya que el navegador es capaz de interpretarlos y siempre será mucho más preciso que no disponer de ellos. Además, es recomendable indicar el cálculo realizado junto a la regla de CSS, como referencia a futuro y en caso de necesitar cambiarlo.

```
| font-size: 2em /* 32px/16px */
```

La magia de la fórmula anterior es que no sólo se puede aplicar a tamaños de fuente, sino que funciona correctamente para crear la estructura de una web a través de etiquetas **div**.

Utilicemos como ejemplo el tema Genesis (<http://my.studiopress.com/themes/genesis/>) de Wordpress. En la siguiente imagen se puede ver que se ha optado por un diseño de 960px de ancho, dividido en una cabecera con dos contenedores y un cuerpo con tres contenedores, ambos separados por un espacio de 20px.



**Figura 2.2** Estructura del tema Genesis

Este es el HTML correspondiente a la estructura:

```
<div id="wrap">
  <div id="header">
    <div id="title-area"></div>
    <div class="widget-area"></div>
  </div>
  <div id="inner">
    <div id="content-sidebar-wrap">
      <div id="content"></div>
      <div id="sidebar"></div>
      <div id="sidebar-alt">
      </div>
    </div>
  </div>
```

Y la el código CSS base de la estructura:

```
#wrap {
  width: 960px;
}

#header {
  width: 960px;
```

```
}

#title-area {
    width: 400px;
}

#header .widget-area {
    width: 540px;
}

#inner {
    width: 960px;
}

#content-sidebar-wrap {
    width: 790px;
}

#content {
    width: 470px;
}

#sidebar {
    width: 300px;
}

#sidebar-alt {
    width: 150px;
}
```

Conociendo los valores objetivo, la primera tarea es establecer nuestro **ancho base**. Esta medida hace referencia al ancho que hemos definido a nuestro layout sin aplicar ningún diseño adaptativo. Como medida general, suele adoptarse como ancho base un 90% del área visible de la pantalla, lo que permite seguir manteniendo toda la estructura y contenidos de la web siempre visibles.

A la hora de aplicar la fórmula, la única diferencia es que en este caso no vamos a hacer uso de la unida de medida `em`, sino de porcentajes, por lo que debemos multiplicar el resultados por 100. Si aplicamos la fórmula a la estructura anterior, el resultado sería el siguiente:

```
#wrap {
    width: 90%;
```

---

```
    max-width: 960px;
}

#header {
    width: 100% /* 960px/960 */;
}

#title-area {
    width: 41.666667% /* 400px/960 */;
}

#header .widget-area {
    width: 56.25% /* 540px/960 */;
}

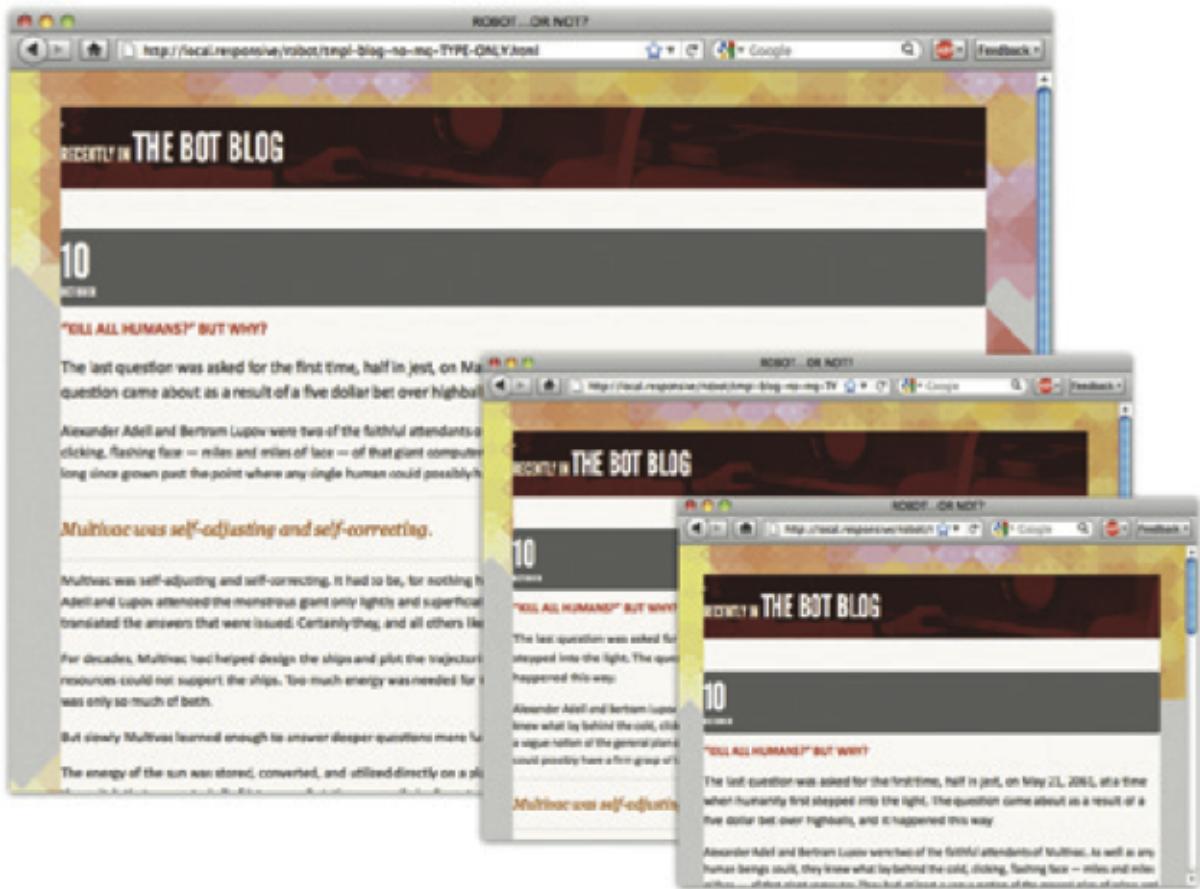
#inner {
    width: 100% /* 960px/960 */;
}

#content-sidebar-wrap {
    width: 82.291667% /* 790px/960 */;
}

#content {
    width: 48.958333% /* 470px/960 */;
}

#sidebar {
    width: 31.25% /* 300px/960 */;
}

#sidebar-alt {
    width: 15.625% /* 150px/960 */;
}
```



**Figura 2.3** Resultado de aplicar contenedores flexibles

Ahora que tenemos nuestros contenedores flexibles, el siguiente paso es obtener el mismo comportamiento para los márgenes y rellenos.

Para determinar la medida de los márgenes, vamos a utilizar la anchura del elemento contenedor. Tomemos como ejemplo la barra lateral. En este caso, el margen lateral izquierdo está definido a 25px, mientras que el ancho del contenedor es de 150px. Por lo tanto, la manera de calcular el nuevo margen es dividir el 25px (target) entre 150px (context), dando como resultado 16.666667%

```
.widget-area ul {
    margin: 10px 0 0 25px;
}

.widget-area ul {
    margin: 10px 0 0 16.666667%;
```

---

En este caso, el comportamiento es exactamente igual al utilizado para calcular la media de los márgenes. Dividimos el relleno actual, 10px, entre la anchura del elemento contenedor 300px, dando como resultado un relleno del 3.33333%.

```
.enews p {  
    padding: 5px 10px 0;  
}  
  
.enews p {  
    padding: 5px 3.33333% 0;  
}
```

El último punto a tener en cuenta es el comportamiento de las imágenes. Por suerte, este es el caso más sencillo. Es suficiente con definir el ancho máximo de las imágenes al 100% (todo el ancho de su contenedor).

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

Al finalizar el capítulo anterior, hemos dado una solución genérica para que las imágenes se muestren de una manera adaptativa, pero no hemos identificado el problema anterior a la solución.

Teniendo en cuenta este pequeño bloque HTML:

```
<div class="figure">
  <p>
    
    <b class="figcaption">Lo, the robot walks</b>
  </p>
</div>

.figure {
  float: right;
  margin-bottom: 0.5em;
  margin-left: 2.53164557%; /* 12px / 474px */
  width: 48.7341772%; /* 231px / 474px */
}
```

El código anterior crea un bloque flotante, de ancho variable que contiene una imagen y una descripción de la misma. El problema surge cuando la imagen proporcionada supera las medidas del bloque que la contiene. La estructura no se ve afectada, y las proporciones de la columna siguen intactas, pero la imagen excede el tamaño del contenedor, produciéndose el siguiente efecto.



# GENESIS FRAMEWORK

Supports HTML5 and Mobile Responsive Design

PAGE LAYOUTS

PAGE TEMPLATES

CONTACT PAGE

CATEGORY #2

CATEGORY #3

CATEGORY #4

POST FORMATS

## Post With Threaded Comments

[jardner](#) — 6 Comments



In a WordPress post, you could edit this to put information about your site so readers know where you are coming from. You can also add as you like in order to share with your readers what is on

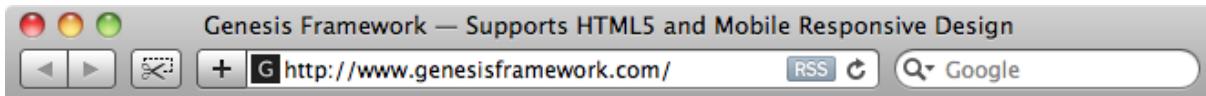
---

### **Figura 3.1** La imagen excede el tamaño de su contenedor

La solución al problema anterior, la vimos en el capítulo anterior y es tan sencillo como aplicar la siguiente restricción a las imágenes:

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

A partir de ahora, todas las imágenes del documento ocuparán como máximo el mismo tamaño que su contenedor, independientemente de su tamaño original.



# Sample Post With Threaded Comments

April 1, 2013 by [Brian Gardner](#) — [6 Comments](#)



This is an example of a WordPress post, you could edit this to put information about yourself or your site so readers know where you are coming from. You can create as many posts as you like in order to share with your readers what is on your mind.

This is an example of a WordPress post, you could edit this to put information about yourself or your site so readers know where you are coming from. You can

---

### Figura 3.2 La imagen encaja perfectamente dentro de su contenedor

Otra buena noticia es que el navegador redimensionará nuestra imagen, manteniendo las proporciones, en la medida que su contenedor cambie de tamaño. Además, esta regla puede ser aplicada al resto de elementos multimedia:

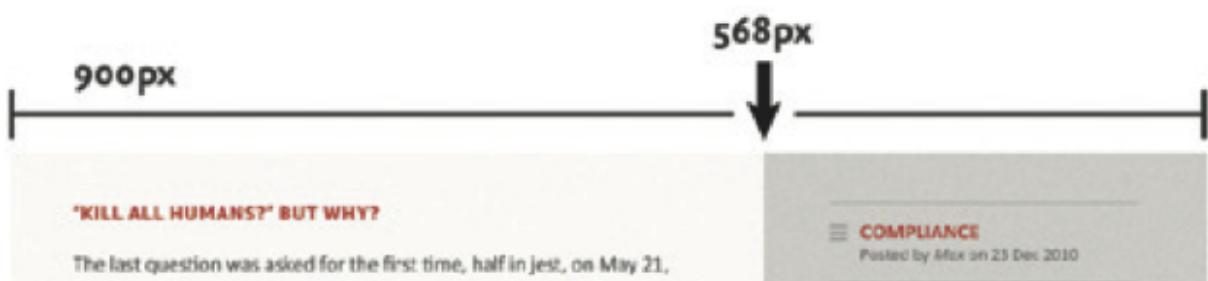
```
img,  
embed,  
object,  
video {  
    max-width: 100%;  
}
```

En 2004, Dan Cederholm <http://bkaprt.com/rwd/18/>, publicó un magnífico artículo para conseguir un efecto de dos columnas a partir de imágenes de fondo. La técnica es realmente muy sencilla, crear una imagen con colores diferenciados que simulasen crear columnas de un mismo alto. Este efecto se consigue con una simple regla de CSS:

```
.blog {  
    background: #F8F5F2 url("blog-bg.png") repeat-y 50% 0;  
}
```

Esta técnica funciona perfectamente, pero está pensada para estructuras de ancho fijo, por lo tanto debemos adaptarla para que funcione correctamente en nuestros diseños flexibles. De nuevo, la manera para conseguirlo es aplicar la fórmula que hemos venido utilizando hasta ahora:  $\text{target} \div \text{context} = \text{result}$ .

El primer paso es buscar el punto de transición en nuestro fondo, el pixel exacto donde se simula la transición entre las columnas. Vamos a suponer que el ancho inicial de nuestra columna es de 900px, y que la separación entre las columnas se produce en el pixel 568.



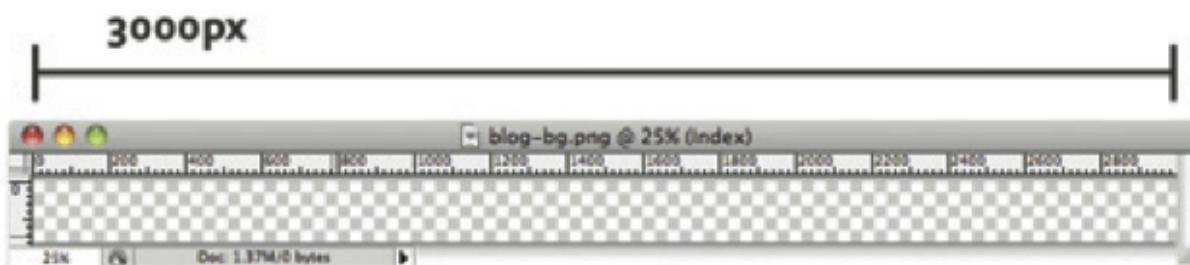
### Figura 3.3 El punto de transición entre las columnas

Con estos simples datos, podemos convertir nuestro fondo fijo para que se muestre de manera adaptativa. Para ello, aplicamos la fórmula que conocemos para calcular el nuevo punto de manera relativa:

$$568 \div 900 = 0.6311111111111111$$

Así pues, nuestro valor objetivo es 63.1111111111111%.

Es hora de obtener nuestra nueva imagen de fondo. Para ello, creamos una nueva imagen con un tamaño lo suficientemente ancho como para cubrir cualquier tamaño de ventana, por ejemplo 3000px. Esto nos asegurará que el fondo se mostrará correctamente aunque el ancho de la ventana sea muy amplio.

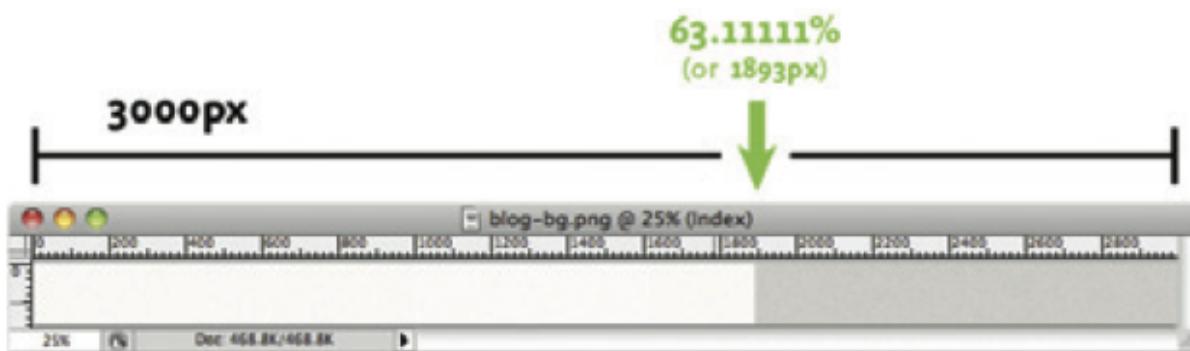


### Figura 3.4 Nuestro lienzo base para crear la imagen de fondo

Debemos calcular el nuevo punto de transición entre las columnas. Como hemos definido el ancho de nuestra imagen a 3000px, el nuevo punto de transición es:

$$3000 \times 0.6311111111111111 = 1893.333333333333$$

Creamos las texturas necesarias para este caso, hasta y desde el punto de transición que hemos obtenido:



### Figura 3.5 Creamos las texturas de nuestro fondo

---

Ahora que ya tenemos preparada nuestra nueva imagen de fondo, ya solo queda aplicar los estilos necesarios para mostrarla correctamente:

```
.blog {  
    background: #F8F5F2 url("blog-bg.png")  
        repeat-y 63.111111111111% 0; /* 568px / 900px */  
}
```

Al igual que en la solución original, estamos posicionando el gráfico en la parte superior del blog y la repetimos verticalmente. La diferencia está en la nueva posición de la imagen, que hará que la imagen se muestre en su posición correcta aunque se redimensione la ventana del navegador.

Realmente, la solución anterior no es del todo flexible, ya que no estamos redimensionando la imagen sino trasladándola para mostrarla correctamente. Esta solución no es válida si necesitamos una imagen de fondo que deba redimensionarse, como un logo o sprites que hemos utilizado en nuestros enlaces.

La solución a este problema viene dada por la propiedad `background-size` de CSS 3, la cual nos permite realmente redimensionar las imágenes de fondo.



Desde la especificación de CSS 2.1, ha sido posible modificar el aspecto de los documentos HTML en función del **tipo de dispositivo** (<http://www.w3.org/TR/CSS2/media.html>) en el que se mostraban. El caso más común es el de crear una hoja de estilos que se aplica al imprimir los documentos:

```
<link rel="stylesheet" type="text/css" href="core.css" media="screen" />
<link rel="stylesheet" type="text/css" href="print.css" media="print" />
```

Como no sólo vamos a formatear nuestros documentos para que se muestren de manera correcta, la especificación CSS ofrece una buena cantidad de tipos de medios (*media type*) para los que podemos aplicar un diseño específico, concretamente los siguientes: all, braille, embossed, handheld, print, projection, screen, speech, tty, y tv.. El problema es existe una extensa variedad de dispositivos que comparten el mismo tipo de medio, pero son completamente diferentes entre sí.

Por suerte, la W3C introdujo los *Media Query* como parte de la especificación de CSS 3, mejorando de manera notable el objetivo de los *Media Type*. Un *Media Query* no sólo nos permite seleccionar el tipo de medio, sino consultar otras características sobre el dispositivo que esta mostrando la página.

Un simple ejemplo de *Media Query*:

```
<link rel="stylesheet" type="text/css"
      media="screen and (max-device-width: 480px)"
      href="shetland.css" />
```

Esta consulta contiene dos componentes:

- Un tipo de medio (screen)
- y una consulta concreta sobre la característica del medio (max-device-width) y el valor objetivo (480px).

En otras palabras, estamos preguntando al dispositivo si su resolución horizontal (max-device-width) es igual o menor a 480px. Si se cumple la condición, el dispositivo cargará la hoja de estilos shetland.css. De otra manera, el link será ignorado.

Los *Media Query* también pueden ser definidos dentro de la propia hoja de estilos:

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

O incluso utilizando la sentencia @import:

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

No importa la manera en la que se defina el *Media Query*, el resultado debe ser el mismo: si el navegador cumple con el tipo de medio y las condiciones indicadas, se aplicarán las reglas CSS definidas.

Los *Media Queries* pueden contener una o más expresiones, expresadas como funciones multimedia, que se resuelven en *true* o *false*. El resultado del *query* o consulta devuelve *true* si el *media type* especificado en el *Media Query* coincide con el tipo de dispositivo en que el documento está siendo mostrado y todas las expresiones en el *Media Query* devuelven *true*.

Cuando un *Media Query* devuelve *true*, la correspondiente hoja de estilo es aplicada, siguiendo las reglas habituales de CSS. Las hojas de estilo con *Media Queries* adjuntos a los *tags* `<link>` seguirán descargándose,

---

incluso si sus *Media Queries* resultan *false* (sin embargo, no se aplicarán).

### ***logical operators***

Se pueden crear *Media Queries* complejos utilizando *logical operators*, incluyendo *not*, *and* y *only*. El operador *and* es usado para combinar múltiples *media features* en un sólo *Media Query*, requiriendo que cada función devuelva *true* para que el *Query* también lo sea. El operador *not* se utiliza para negar un *Media Query* completo y el operador *only* se usa para aplicar un estilo sólo si el *Query* completo es correcto.

Además, se pueden combinar múltiples *Media Queries* separados por comas en una lista; si alguno de los *Queries* devuelve *true*, todo el *\*media statement* devolverá *true*. Esto es equivalente a un operador *or*.

El keyword *and* se usa para combinar múltiples *media features*, así como combinar éstos con *media types*. Un *Media Query* básico sería:

```
| @media (min-width: 700px) { ... }
```

Sin embargo, si quisiéramos que esto sólo se aplicara si la pantalla está en modo *landscape*, se usaría el operador *and*:

```
| @media (min-width: 700px) and (orientation: landscape) { ... }
```

Si además, sólo quisiéramos que esto se aplicara si el dispositivo fuera un *media type TV*:

```
| @media tv and (min-width: 700px) and (orientation: landscape) { ... }
```

Cuando se utilizan las listas separadas por comas en los *Media Queries*, si algunas de las *Media Queries* devuelven *true*, los estilos se aplican. Cada *Media Query* separado por comas en la lista se trata como un *Query* individual, y cualquier operador aplicado a un *Media Query* no afecta a los demás. Esto significa que los *Media Queries* separados por comas puede dirigirse a diferentes *media features*, *types* o *states*.

Por ejemplo, si quisiéramos aplicar un conjunto de estilos si el dispositivo de visualización tienen un mínimo de 700px o está en modo *landscape*:

```
| @media (min-width: 700px), handheld and (orientation: landscape) { ... }
```

---

La keyword `not` se aplica al *Media Query* en su totalidad y devuelve *true* si el *Media Query* devuelve *false* (como `monochrome` en una pantalla a color). Este keyword no se puede utilizar para negar un *individual feature query*, solamente un *entire media query*. Por ejemplo:

```
| @media not all and (monochrome) { ... }
```

Esto significa que el *Query* es evaluado de esta manera:

```
| @media not (all and (monochrome)) { ... }
```

en vez de así:

```
| @media (not all) and (monochrome) { ... }
```

Por ejemplo, este otro *Media Query*:

```
| @media not screen and (color), print and (color)
```

Se evalúa así:

```
| @media (not (screen and (color))), print and (color)
```

El keyword `only` previene a los navegadores que no soportan *Media Queries*:

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

La especificación de los *Media Query* incluye una larga lista de características que podemos consultar sobre el dispositivo. En este especificación, se hace referencia a dos términos que hay que tener claros:

- *display area*: espacio disponible en la ventana del navegador para mostrar el contenido de la página web.
- *rendering surface*: hace referencia al espacio total disponible en el dispositivo.

El listado completo de características es el siguiente:

		min- max-
width	El ancho del área de visualización (display area)	Sí
height	El alto del área de visualización (display area)	Sí
device-width	El ancho total del dispositivo (rendering surface)	Sí
device-height	El alto total del dispositivo (rendering surface)	Sí
orientation	Acepta los valores portrait o landscape	No
aspect-ratio	Relación de aspecto entre el ancho y alto del área de visualización	Sí
device-aspect-ratio	Relación de aspecto entre el ancho y alto del dispositivo	Sí
color	El número de bits de profundidad de color	Sí
color-index	El número de entradas en la tabla de colores del dispositivo	Sí
monocrome	El número de bits de profundidad de color, en dispositivos monocromáticos	Sí
resolution	Densidad de pixels en el dispositivo, medido en dpi	Sí

Una de las ventajas es que podemos encadenar múltiples condiciones en el media query, utilizando la palabra reservada `and`:

```
@media screen and (min-device-width: 480px) and (orientation:landscape)
{
  ...
}
```

---

La etiqueta `meta` para el `viewport` fue introducida por Apple en Safari para móviles en el año 2007, para ayudar a los desarrolladores a mejorar la presentación de sus aplicaciones web en un iPhone.

La etiqueta `viewport` nos permite definir el ancho, alto y escala del área usada por el navegador para mostrar contenido. Al fijar el ancho o alto del `viewport`, podemos usar un número fijo de pixeles (ej: 320px, 480px, etc) o usar dos constantes, `device-width` y `device-height` respectivamente. Se considera una buena práctica configurar el `viewport` con algunas de estas dos constantes, en lugar de utilizar un ancho o alto fijo. Muchos desarrolladores caen en el error de configurar el `viewport` con un `width` de 320px, siendo este el ancho del iPhone.

A continuación la configuración más común de una etiqueta `meta`:

```
<meta name="viewport" content="width=device-width, initial-scale=1,  
maximum-scale=1">
```

Las propiedades a tener en cuenta son las siguientes:

- `width`: controla el ancho del área de visualización. Puede ser inicializado a un número concreto de pixels, como `width=600` o con el valor especial `device-width`, que indica el ancho total del dispositivo, en pixels en una escala del 100%.
- `initial-scale`: controla el nivel de zoom inicial al cargarse la página. Las propiedades `maximum-scale`, `minimum-scale`, y `user-scalable` controlan como el usuario puede realizar zoom sobre la página.

A pesar de haber sido introducido por Apple, esta propiedad ha sido ampliamente adoptada por el resto de navegadores móviles, convirtiéndose en un estándar *de facto*.

Hasta ahora, hemos definido todas las herramientas necesarias para crear estructuras adaptativas: diseños basados en rejillas flexibles, estrategias para incorporar elementos multimedia y la potencia de los *Media Query* para adaptar el contenido a nuestras necesidades. Vemos como incorporar estas técnicas a la hora de desarrollar un sitio web adaptativo.

Un diseño adaptativo, implementado de una manera correcta, puede dar a los usuarios de la web, un alto nivel de continuidad entre los diferentes *contextos*. Esto es así, porque de la manera más simple, un diseño adaptativo es capaz de mostrar un documento HTML correctamente en una infinidad de dispositivos, gracias a una estructura flexible y los *Media Query* que aseguran un diseño portable y accesible en la manera de lo posible.

De alguna manera, es posible identificar el *contexto* en el que se visita una web, a partir del dispositivo utilizado. De este contexto, podemos definir un tipo de usuario y sus objetivos. En otras palabras, un usuario móvil quiere un acceso rápido a la información y realizar diferentes tareas a las que realizaría sentado en su sofá con su portátil. En este caso, el tiempo y el ancho de banda están en extremos totalmente opuestos.

Por otra parte, si las prioridades y los objetivos del usuario varían según el *contexto*, entonces puede que disponer de un único documento HTML pueda suponer un problema, dependiendo de la manera en la que se encuentre estructurada la información.

De todas formas, es complicado suponer el *contexto* del usuario únicamente por el tipo de dispositivo, ya que efectivamente, es solamente eso: **una suposición**. ¿Cómo diferenciar, si mi navegación *móvil* se realiza desde la entrada del metro o desde el sofá de mi casa? ¿Es posible por tanto suponer un *contexto*?

Por lo tanto, no podemos inferir el *contexto* del usuario en base a su dispositivo. Así mismo, las palabras *mobile* o *desktop* no definen el comportamiento en la que los usuarios acceden a la web: un portátil puede ser un dispositivo móvil (por ejemplo en un tren), al igual que un *smartphone* o *tablet* puede estar fijo en un lugar. El desarrollo adaptativo no pretende ser un reemplazo de los actuales sitios móviles, sino que forma parte de una estrategia de desarrollo, donde se pretende evaluar si efectivamente es necesario separar totalmente la experiencia móvil o tiene más sentido mostrar la información de una manera adaptativa.



**Figura 5.1** ¿Es posible realizar una versión adaptativa del New York Times?

Una de las primeras preguntas (si no lo primer) que debemos hacernos, a la hora de plantearnos un diseño adaptativo es la siguiente: **¿En qué me-**

---

**dida este contenido o funcionalidad beneficia o aporta valor a nuestros usuarios?** De hecho, esto es algo que deberíamos preguntar **siempre** para cualquier tipo de proyecto, sea web o no.

Si partimos de un planteamiento *mobile first*, hay que asegurarse que la información que mostramos, y las funcionalidades que implementamos sean importantes para el usuario, ya que no hay espacio suficiente para todo. Hay que darse cuenta de lo que realmente importa. Diseñar partiendo de este paradigma, nos obliga a concentrarnos en lo realmente importante.

If you design mobile first, you create agreement on what matters most. You can then apply the same rationale to the desktop/laptop version of the web site. We agreed that this was the most important set of features and content for our customers and business; why should that change with more screen space? *Luke Wroblewski*

En otras palabras, diseñar desde un inicio pensando en dispositivos móviles puede enriquecer la experiencia de los usuarios, proporcionando un elemento que normalmente se pasa por alto: enfocarnos en lo realmente importante. Esto no quiere decir que los diseños sean simples, faltos de contenidos o funcionalidades, sino que debemos concentrarnos en lo realmente importante.

El siguiente paso es identificar el número de diseños diferentes que vamos crear, para acomodarnos a los distintos tamaños de dispositivos. La siguiente tabla muestra los anchos más comunes a la hora de identificar los puntos de ruptura:

320 pixels	Para dispositivos pequeños como teléfonos, en disposición vertical
480 pixels	Para dispositivos pequeños como teléfonos, en disposición horizontal
600 pixels	Para tabletas de menor tamaño, como Amazon Kindle (600×800), en disposición vertical

768 pixels	Para tabletas de unas 10", como el iPad (768x1024), en disposición vertical
1024 pixels	Para tabletas de unas 10", como el iPad (768x1024), y pequeños portátiles o <i>netbooks</i> , en disposición horizontal
1200 pixels	Para pantallas panorámicas, en portátiles o dispositivos de escritorio
1600 pixels	Para pantallas panorámicas, en portátiles o dispositivos de escritorio

De manera tradicional, el desarrollo web (y en general, el desarrollo de software) ha seguido las siguientes fases para la construcción de un sitio web:

- Una fase de planteamiento inicial, donde se capturan los requisitos que debe cumplir la solución final, se plantea la estructura de contenidos y se realizan *mockups* que serán la base del diseño.
- En la fase de diseño, los *mockups* se convierten en pantallas utilizando una herramienta del tipo *Photoshop* o *Fireworks*. Estas pantallas deben ser aprobadas por el cliente antes de seguir adelante.
- Finalmente, una vez que los diseños se encuentran aprobados, pasan al equipos de desarrollo para crear las páginas estáticas en HTML.

En un desarrollo adaptativo, este proceso puede resultar muy pesado y costoso. Cuando diseñamos una página, lo ideal sería diseñar también el aspecto de esta página en los distintos dispositivos, lo que supondría realizar tantos diseños como puntos de ruptura existan. Si tenemos que repetir este proceso con quince o cincuenta páginas, simplemente deja de ser viable.

En este punto, una posible mejora puede ser combinar el equipo de diseño y el de desarrollo, para que, partiendo de un diseño fijo y único, discutir de manera conjunta cómo el diseño se va a acomodar a las distintas resoluciones, y las implicaciones que puede tener a la hora de implemen-

---

tarlo. De esta manera, rápidamente surgen preguntas del tipo: ¿Cómo va a funcionar esta galería de imágenes en un dispositivo táctil? ¿De qué manera se va a mostrar este elemento emergente? ¿Qué pasa si no disponemos de JavaScript en el dispositivo?

De esta manera podemos comenzar a construir prototipos con un diseño flexible, escalable en los distintos dispositivos, de una manera muy ágil, implicando a ambos equipos.

Durante el ciclo de diseño/desarrollo, las páginas son constantemente refinadas mientras las construimos, con el objetivo de hacerlas totalmente adaptativas. Hemos convertido una página estática, diseñada para mostrarse de la misma manera en todos los dispositivos, en un diseño fluido al que hemos aplicado *Media Queries* para mostrarse correctamente en los diferentes dispositivos. En este último caso, tenemos la posibilidad de incluir en nuestra página un *polyfill* para hacer funcionar correctamente esto *Media Query* en navegadores que no lo soportan, como Internet Explorer 7 u 8: [respond.js](https://github.com/scottjehl/Respond) (<https://github.com/scottjehl/Respond>)

¿Qué ocurre si el navegador no soporta @media y no tiene activado JavaScript? Pues que simplemente se mostrará la página con su diseño original, a tamaño completo.

Otro problema añadido, al aplicar un diseño adaptativo, es el que se muestra a continuación:

```
.blog {  
    background: #F8F5F2 url("img/blog-bg.png") repeat-y;  
}  
  
@media screen and (max-width: 768px) {  
    .blog {  
        background: #F8F5F2 url("img/noise.gif");  
    }  
}
```

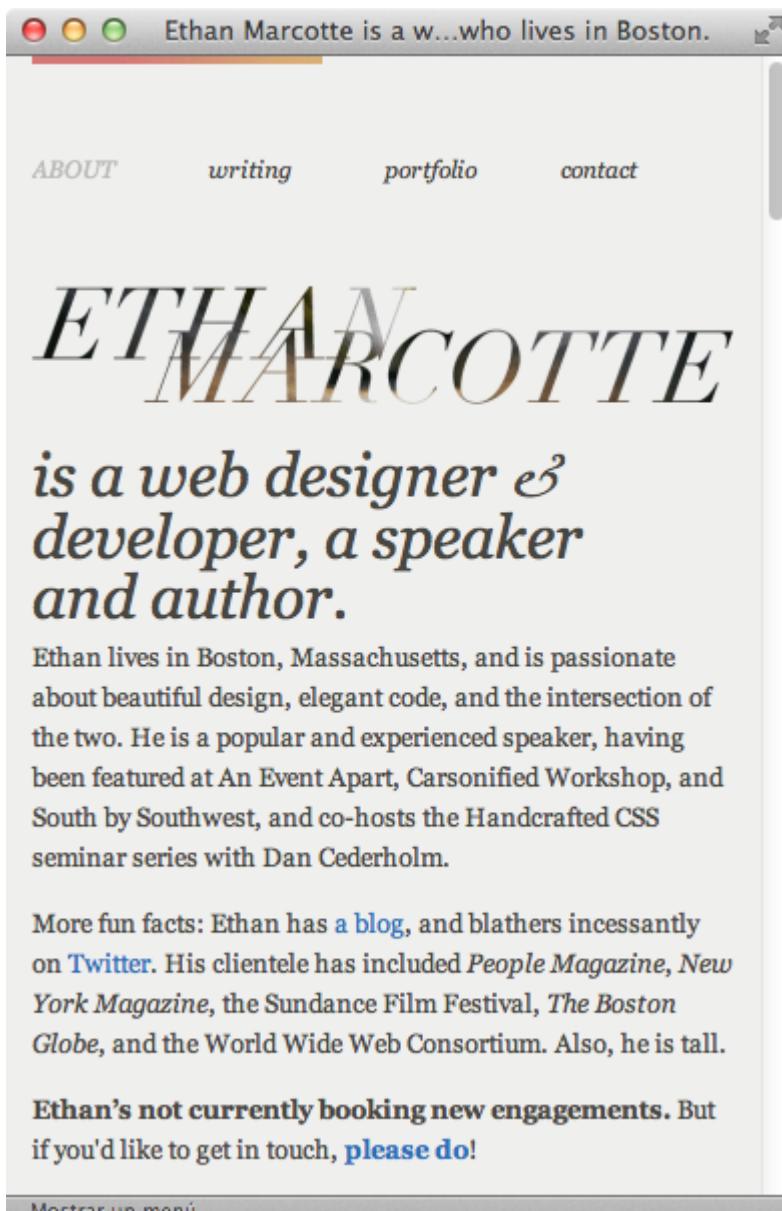
En este caso, primero aplicamos una imagen de fondo blog-bg.png al elemento .blog. Después, para dispositivos con una pantalla más pequeña, reemplazamos esa imagen por una simple imagen GIF. El problema, es que en este tipo de dispositivos se descargan ambas imágenes, afectan-

---

do directamente a la cantidad de información descargada a través de la red.

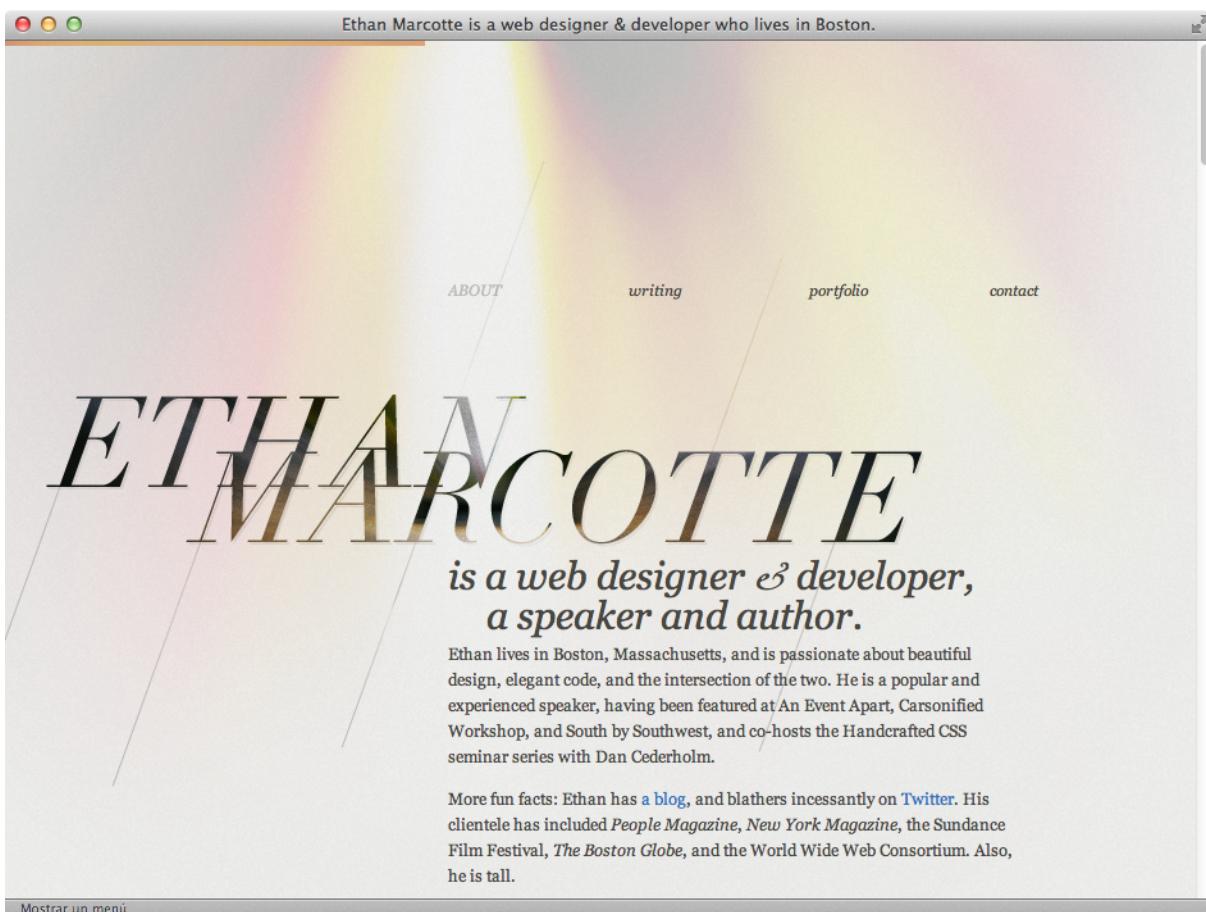
Hablando en términos generales, el diseño adaptativo hace referencia a comenzar con una *resolución de referencia*, y utilizar los *Media Query* para adaptar el diseño a otros contextos. Una estrategia más responsable para diseños adaptativos es pensar primero en dispositivos móviles, en lugar de un contexto de escritorio. Esto es, comenzamos definiendo una estructura para dispositivos más pequeños, y posteriormente utilizamos los *Media Query* para escalar el diseño a mayores resoluciones.

Un ejemplo de esta estrategia, es la web de Ethan Marcotte (<http://ethanmarkotte.com/>) . Por defecto, el contenido es mostrado de una manera lineal, pensado especialmente para dispositivos móviles.



**Figura 5.2** Aspecto de la web de Ethan Marcotte en dispositivos móviles

A medida que el espacio disponible aumenta, se muestra el diseño completo. La estructura se vuelve más compleja, con *assets* más pesados.



**Figura 5.3** Aspecto completo de la web de Ethan Marcotte

A pesar de esta estrategia, el diseño sigue siendo totalmente adaptativo, y utiliza las técnicas que hemos visto en anteriores capítulos: la estructura está basada en una rejilla flexible y las imágenes se muestran correctamente en cualquier resolución. Per ha diferencia de la estrategia tradicional, se utiliza la propiedad `min-width` para escalar y diseñar la web. La estructura básica de la hoja de estilos es la siguiente:

```
/* Default, linear layout */
.page {
    margin: 0 auto;
    max-width: 700px;
    width: 93%;
}

/* Small screen! */
@media screen and (min-width: 600px) { ... }

/* "Desktop" */
@media screen and (min-width: 860px) { ... }
```

```
/* IT'S OVER 9000 */
@media screen and (min-width: 1200px) { ... }
```

Pongamos como ejemplo que tenemos una galería de imágenes en nuestra web, que funciona como un carrusel utilizando JQuery. En este caso, si el navegador no dispone de Javascript, tendremos una lista de imágenes, una sobre otra. Si alguien visita la web con JavaScript desactivado, la experiencia será bastante negativa.

La solución a este problema es cargar únicamente el contenido extra de la página a través de JavaScript. Para ello, la primera tarea a realizar es eliminar todas las imágenes excepto la primera de ellas, que es la que siempre se mostrará:

```
<div class="slides">
  <div class="figure">
    <b></b>
    <div class="figcaption">...</div>
  </div><!-- /end .figure -->
</div><!-- /end .slides -->
```

El siguiente paso es crear una nueva página HTML que contenga las imágenes restantes:

```
<div class="figure">
  <b></b>
  <div class="figcaption">...</div>
</div><!-- /end .figure -->
<div class="figure">
  <b></b>
  <div class="figcaption">...</div>
  ...
</div><!-- /end .figure -->
```

Éste no es un documento HTML válido, ni pretende serlo, ya que vamos a cargarlo a través de JavaScript para insertarlo en nuestra página actual.

```
$(document).ready(function() {
  $.get("slides.html", function(data) {
    var sNav = [
      '<ul class="slide-nav">',
```

```
'<li><a class="prev" href= »
    "#welcome-slides">Previous</a></li>',
'<li><a class="next" href="#welcome-slides"> »
    Next</a></li>',
'</ul>
].join("");
$(".welcome .slides")
.append(data)
.wrapInner('<div class="slidewrap"> »
<div id="welcome-slides" class="slider"> »
</div></div>')
.find(".slidewrap")
}); });
});
```

Y esto es todo. Si el navegador soporta JavaScript, se cargarán el resto de las imágenes, y se creará el carrusel. En caso de no disponer de JavaScript, simplemente se mostrará la imagen principal.

Es posible restringir, desde JavaScript, las funcionalidades del documento en función del contexto, haciendo el script dependiente de la resolución.

```
if (screen.width > 480) {
    $(document).ready(function() { ... });
}
```

Esta instrucción en JavaScript equivale a un *media query* min-width: 480px: si el ancho de la pantalla es menor a 480px, el script no se ejecutará.

A partir del siguiente sitio web, y utilizando una estrategia "Mobile First", adaptarlo para que se visualice de manera correcta en un dispositivo *smartphone* y en una *tablet*. Se pide:

- Identificar la disposición de los elementos para cada una de las resoluciones.
- El diseño actual es una estructura de anchos fijos. El nuevo diseño se debe basar en una estructura de rejilla flexible.
- Las imágenes que acompañan a los post, deben mostrarse de manera correcta, sin exceder el tamaño de su contenedor.
- [Opcional] Aplicar propiedades CSS 3 para los fondos, bordes redondeados, tipografías, animaciones...

[Descargar ficheros fuente \(snippets/final/rwd.zip\)](#)