Responsive Versus Adaptive Web Design

A development approach comparison

Authors
Sriram Ramanathan, Chief Technology Officer
Matthew Trevathan, Director of Product Platform Development
Amit Aghara, VP of Solution Engineering
Manojav Sridhar, Director, Platform Solution Engineering
Table of contents

Executive summary .................................................................................................................. 1
Responsive web design .......................................................................................................... 2
• Performance .......................................................................................................................... 4
• Testing .................................................................................................................................. 4
• Maintenance ........................................................................................................................... 5
Adaptive web design ............................................................................................................. 6
• Performance .......................................................................................................................... 7
• Testing and maintenance ...................................................................................................... 7
The Kony approach ............................................................................................................... 7
• Using Kony for responsive web design ............................................................................... 8
• Using Kony for adaptive web design .................................................................................. 8
Conclusion ............................................................................................................................... 10
### Executive summary

As the number of gadgets that connect to the web grows, so does debate among developers about the best way to design and develop for them. Today’s users are diverse, accessing the internet through applications, browsing the mobile web, and expecting uncompromised experiences no matter how their content is delivered. These patterns are global, and the diversity of use is only increasing (see table 1.)

<table>
<thead>
<tr>
<th>Area</th>
<th>SMS</th>
<th>Web browsing</th>
<th>Email</th>
<th>Social networking</th>
<th>Apps</th>
<th>Streaming music</th>
<th>Instant messaging</th>
<th>Video/ mobile TV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td>94%</td>
<td>60%</td>
<td>55%</td>
<td>58%</td>
<td>59%</td>
<td>21%</td>
<td>33%</td>
<td>19%</td>
</tr>
<tr>
<td>Brazil</td>
<td>85%</td>
<td>69%</td>
<td>66%</td>
<td>75%</td>
<td>74%</td>
<td>39%</td>
<td>57%</td>
<td>43%</td>
</tr>
<tr>
<td>China</td>
<td>84%</td>
<td>75%</td>
<td>58%</td>
<td>62%</td>
<td>71%</td>
<td>59%</td>
<td>67%</td>
<td>39%</td>
</tr>
<tr>
<td>India</td>
<td>45%</td>
<td>15%</td>
<td>17%</td>
<td>26%</td>
<td>13%</td>
<td>11%</td>
<td>15%</td>
<td>8%</td>
</tr>
<tr>
<td>Italy</td>
<td>89%</td>
<td>37%</td>
<td>51%</td>
<td>47%</td>
<td>49%</td>
<td>26%</td>
<td>35%</td>
<td>17%</td>
</tr>
<tr>
<td>Russia</td>
<td>95%</td>
<td>68%</td>
<td>55%</td>
<td>59%</td>
<td>64%</td>
<td>41%</td>
<td>34%</td>
<td>36%</td>
</tr>
<tr>
<td>S. Amer.</td>
<td>93%</td>
<td>80%</td>
<td>52%</td>
<td>55%</td>
<td>81%</td>
<td>40%</td>
<td>70%</td>
<td>44%</td>
</tr>
<tr>
<td>Turkey</td>
<td>78%</td>
<td>37%</td>
<td>33%</td>
<td>69%</td>
<td>38%</td>
<td>22%</td>
<td>50%</td>
<td>9%</td>
</tr>
<tr>
<td>UK</td>
<td>92%</td>
<td>66%</td>
<td>68%</td>
<td>63%</td>
<td>56%</td>
<td>20%</td>
<td>37%</td>
<td>19%</td>
</tr>
<tr>
<td>US</td>
<td>86%</td>
<td>82%</td>
<td>75%</td>
<td>63%</td>
<td>62%</td>
<td>38%</td>
<td>28%</td>
<td>28%</td>
</tr>
</tbody>
</table>

Source: © Nielsen, (February 2013) via © mobiThinking

### Table 1. Activities performed by smartphone users at least once a month, according to Nielsen.

So how can today’s businesses address these varied use cases? Facebook, Twitter, LinkedIn, ebay, Target, and many other companies have mobile strategies that leverage both native applications and mobile web channels. Strategies that target the Mobile and mainstream Web take two approaches to solve the same question: “How to I render content to the multitude of mobile devices?”

This paper discusses the two approaches that are being used to meet this challenge, responsive web design (RWD) and adaptive web design (AWD). It covers examples, pros and cons, and how Kony addresses today’s complex web requirements.
Responsive web design

Ethan Marcotte introduced the concept of responsive web design in his article “Responsive Web Design” on A List Apart in 2010. In essence, responsive web sites adapt the layout of the web page to the viewing parameters of the device. This may sound confusing unless you have ever tried to view a traditional website on your smartphone that doesn’t use adaptive or responsive design: the page renders to the screen, and some pictures may fill the screen while text on the page becomes unreadable without a magnifying glass. The responsive approach scales the site according to the screen size, delivering a manageable amount of information to the screen. Starbucks provides a great example of how responsive design works, shown in Figure 1.

Figure 1. Starbucks website as an example of responsive web design.

Figure 1 shows a screenshot of the Starbucks website using a maximized view of the browser. As expected, the screen displays a lot of information, providing subtitles in the header navigation and links to popular social network platforms such as Facebook and Twitter.

There doesn’t seem to be anything magical about responsive design until a user changes the screen size or tries to use the website on a smaller device, such as a tablet. Starbucks keeps a focus on what is important to Starbucks, as the advertisement is still center stage on the landing page, but some content on the page has been changed while other items found on the desktop landing page have disappeared altogether, shown in Figure 2 (following page).
Figure 2. Starbucks website in a portrait orientation resembling a tablet view.

“This concept of shifting the data at certain breakpoints while maintaining readable, proportional data is key to responsive design.”

Some readers may notice the page in figure 2 has changed in subtle ways. For instance, the top navigation in Figure 1 has descriptive subtitles that are removed in the tablet rendered version (figure 2), while the social networking links have been moved below the visible part of the page. Additionally, the search bar has been moved from the top of the page to the bottom, below the visible area. Most of the information is still on the page, but it now requires more scrolling.

This concept of shifting the data at certain breakpoints while maintaining readable, proportional data is key to responsive design. At specific points within the layout, data will shift to match a layout type, often taking columns of information and moving them under the columns in the new layout for the specific screen size. Most designers set breakpoints at three levels: desktop, tablet, and mobile, with mobile being the first screens they design while progressively enhancing the experience as the screen size changes. This becomes more evident on the Starbucks site when the screen width is shrunk even more.

The screenshots below show how the information has been collapsed into a single column to fit on the device. Starbucks keeps important information higher on the screen. Notice how the coupon is always important to Starbucks, and is consistent across the different screen views.
Figure 3. Top, middle, and bottom portions of mobile view of Starbucks website on a smartphone.

Although Starbucks gets plenty of information onto a single page, it requires a lot of scrolling to reach the data at the bottom of the page. That’s why it is imperative that designers understand what is important to the end user, given the screen size and expected use, so real estate can be prioritized. Starbucks also does a good job of illustrating how scalable graphics are important in this model to maintain a professional, crisp site across all the platforms. Overall, the Starbucks example shows how responsive design can leverage techniques within the browser to deliver content that is tailored to the display using a single HTML page.

Performance
Delivering optimal performance with responsive design takes extra thought and preparation. Common pitfalls include different bandwidth considerations for different devices, loading the same JavaScript you would load for a desktop on a mobile device, or loading large image sizes that are cumbersome for a mobile experience.

“Delivering optimal performance with responsive design takes extra thought and preparation”

Testing
Many development shops forget that a change to a responsive layout at one breakpoint requires testing across all breakpoints. If a developer changes the design of the layout for the tablet breakpoint, testing should not be isolated to that one channel. In a responsive layout, there may be aspects that a developer changes within the layout without realizing the consequences the change may have on other aspects of the layout. Developers must ensure that all channels are retested when a change is made using RWD.
Maintenance

Let’s consider an example from a financial institution that uses RWD for its landing or home page. To deal with every browser incompatibility and spec implementation difference, this code (shown below in figure 4) needs to ensure a constant theme with a “write once” code base. This is just for one page (see figure 5); every different style of page and use of widget will have their own inconsistencies. The various combinations of style and widget use have to be addressed and maintained many times over for complex websites that can be as large as 300 separate flows, as with an intricate retail banking application.

```javascript
/*
1. Avoid the WebKit bug in Android 4.0.* where (2) destroys native `audio`
and `video` controls.
2. Corrects inability to style clickable `input` types in iOS.
3. Improves usability and consistency of cursor style between image-type
`input` and others.
4. Removes inner spacing in IE 7 without affecting normal text inputs.

Known issue: inner spacing remains in IE 6.
*/

button,
input[type="button"], /* 1 */
input[type="reset"],
input[type="submit"] {
-webkit-appearance: button; /* 2 */
cursor: pointer; /* 3 */
*overflow: visible; /* 4 */
}

/*
Removes inner padding and search cancel button in Safari 5 and Chrome
* on OS X.
*/

input[type="search"]::-webkit-search-cancel-button,
input[type="search"]::-webkit-search-decoration {
-webkit-appearance: none;
}
```

Figure 4. Example of code created to deal with browser incompatibility.

```javascript
/*
1. Addresses box sizing set to content-box in IE 8/9.
* Known issue: excess padding remains in IE 6.
*/

input[type="checkbox"],
input[type="radio"] {
box-sizing: border-box; /* 1 */
padding: 0; /* 2 */
*height: 13px; /* 3 */
*width: 13px; /* 3 */
}

/*
1. Addresses `appearance` set to `searchField` in Safari 5 and Chrome.
2. Addresses `box-sizing` set to `border-box` in Safari 5 and Chrome
* (include `-box` to future-proof).
*/

input[type="search"] {
-webkit-appearance: textfield; /* 1 */
-box-box-sizing: content-box;
-webkit-box-sizing: content-box; /* 2 */
box-sizing: content-box;
}
```

Figure 5. Additional example of code needed to deal with browser incompatibility.
Adaptive web design

Adaptive web design (AWD) sends a tailored web experience to a device by determining which device type has requested the content. AWD often uses a set of predefined layouts based on screen size, along with CSS and JavaScript, to deliver a user experience tailored to the device. AWD can also allow a developer to tailor content to a specific device, like an iPhone, if there are particular visual design components they want to use for the device.

Perhaps a developer wants to deliver high quality images to a Retina® display while using standard images for a regular iPad. AWD doesn’t limit the developer from using the same JavaScript across devices or reusing CSS, but it does provide channel separation, allowing a developer to make changes for a specific device or set of devices without having to retest the entire mobile web experience. Additionally, the device-targeted experience is lightweight and only sends the code that the device needs, increasing performance of sites built using adaptive web design.

The primary difference between AWD and RWD is how the content is served up. AWD can deliver a more native experience over the web because an AWD site detects the type of device and delivers a view that has device-specific functionality – iPhone buttons or Android navigation bars, for instance, depending on the actual device. RWD relies on screen size, not on the device type, to render content, so content developed within RWD frameworks creates buttons, menus, and layouts based on screen dimensions only.

The CNN website illustrates a good example of adaptive web design. Figure 6 below shows how the layout, look, and feel of the CNN website differs on two different devices. The first thing to note is that the CNN site instantly recognizes that an android device has connected to it and offers the native version of the site to the end user (left). It is also easy to see that the CNN site has a more desktop-like view when rendered on a Nook HD (right).

Figure 6. CNN website displayed on an Android device.
Studies show that people use mobile devices differently than they use desktop computers. Adaptive web design makes it easier to tailor the web content and design based on how the user will access the content, rather than attempting to deliver the exact same content to the end user regardless of the viewing device. In the example above, CNN can deliver content to both devices, but users don’t get confused on their desktop if they resize their browser, as with a responsive site. The theming for the mobile experience looks tailored with different fonts and table headlines that differ from the main site. The key here is that menu bar, buttons, and other objects may be tailored to the device itself, not just the screen size.

**Performance**

AWD sites are built to match specific devices or device groups, allowing developers to tailor the code to the device. This often reduces the size of the code downloaded by the device and minimizes the need for downloading and executing unused JavaScript. Although RWD has made significant improvements in performance, there are many times when tailoring code for device performance is important and AWD is the right solution.

**Testing and maintenance**

Development shops should consider the types of tools they are using for development and how much traffic a target device is receiving to help determine if AWD is the right fit. The advantage of AWD is that a developer can deploy code for a specific channel without having to run regression testing on other channels; however, without the right tool set, this approach can lead teams to maintain multiple code bases for each device.

**The Kony approach**

A good mobile experience is different than a good desktop experience. It is important to think of how the target audience is going to use the site. Thinking mobile isn’t just thinking about how the layout will differ. The designer needs to think about how the target audience may use their smartphones differently than they use their desktops. There are times when a tailored, concise mobile experience is more important than a responsive web experience. *Thinking about what a user needs from the experience for a specific device is key.*

That’s why Kony offers the flexibility to choose the approach that makes the most sense for each business and its users. Whether the development shop chooses responsive or adaptive web design, the Kony framework supports it.

*“Thinking mobile isn’t just thinking about how the layout will differ.”*
Using Kony for responsive web design
Responsive web designers tout the fact that the RWD approach promotes using a single code base for delivering web-based content. The Kony develop once, deploy everywhere approach extends the idea of using a single code base beyond the web, allowing developers to leverage JavaScript for both native and web-based applications.

Kony’s dynamic HTML forms allow a developer to create responsive pages that can be interlaced with Kony native or hybrid pages to create a mobile experience that seamlessly blends responsive content into the mobile experience. This approach is often used when content changes on a regular basis, but the base mobile application does not.

![Figure 7. Kony HTML forms interlace native and hybrid pages.](image)

For instance, a company may have a list of frequent activities that the business publishes daily. The activities need to be created by one team, approved by another team, and published by a third team. As shown in figure 7, Kony dynamic HTML lends itself to leveraging a web content engine, such as Drupal, where content workflow is an important part of the process, while using the services offered by the Kony Cloud in a single Kony application.

Using Kony for adaptive web design
The Kony approach makes AWD seamless, empowering developers to deploy native code and web-based applications using the same code base. Developers are already creating a tailored native experience. Kony allows them to create a mobile web experience using the same code base, tailored to specific devices, to deliver an optimized HTML5-based experience. The Kony Multi-channel Experience Platform detects the type of device making a request and delivers a highly custom look and feel that is created from the same code base as the native code.

“Kony makes AWD seamless since developers can deploy their native code and web-based application using the same code base.”
The Kony approach promotes reuse of business and view logic across both native and web-based applications while allowing developers to make changes for a specific device without affecting other devices. When a developer creates a native application for an iPhone, Android, iPad, or other device, the same code can be deployed.

Consider the following example: building a simple movie listing application. Figure 8 shows how by using the same code base with different layouts and reusing the business logic and data mapping, Kony delivers a web experience that is responsive to the device on which it is consumed.

![Figure 8](image)

**Figure 8.** Kony Studio showing three different layouts for the different web channels (left). Kony Studio showing layout for the web on a smartphone (middle). Corresponding smartphone display (right).

Figure 9 is a layout for tablet devices that leverages the extra tablet real estate to produce a two-column view that shows a film and its corresponding review.

![Figure 9](image)

**Figure 9.** A layout for tablet devices that leverages the extra tablet real estate to produce a two-column view that shows a film and its corresponding review.
And for desktop devices (figure 10), the additional real estate allows the developer to produce a grid layout that displays all the information at the user’s fingertips.

![Figure 10. Shows the same code base optimized for desktop display.](image)

All of these outputs are single-page applications (SPA) that leverage true HTML5 and CSS3 for output. Kony AWD also allows developers to deploy new features or abilities to a new web channel without having to retest the other channels that would be required when using responsive design. Additionally, the Kony ability to display dynamic web content helps developers quickly ingest web content engines like Drupal to easily deliver responsive content to devices.

**Conclusion**

It’s not easy to come up with a “one size fits all” approach to achieving true responsiveness for each of your users’ screen sizes or device types – regardless of the use case or UI/UX.

While responsive web design is a promising approach to addressing key objectives, the use of CSS media queries to create a single application output for all screen sizes results in some performance and maintenance issues. Additionally, you really don’t have a lot of flexibility around how to reorganize your UI layout for a specific screen size without developing a new layout.

Adaptive web design offers flexibility and choice on how you create a responsive application for each device/screen size group. And while it does require you to create multiple UI layouts that could create a larger upfront development effort, it ultimately results in an optimized set of outputs that achieves responsiveness in its own way.
Both responsive web design and adaptive web design have their pros and cons. At Kony, we believe the right approach is up to you. Depending on your use cases, you can adopt either approach – or a combination – offering you the flexibility and choice to best address your business needs.

Footnotes