blink

By Heidi Adkisson

In the year 2000, there were two neatly divided user experience worlds: the world of the web and the world of desktop applications.

- The desktop world was about performing work: word processing, image editing, crunching numbers with a spreadsheet. In this environment, operating system vendors (primarily Microsoft and Apple) worked hard to enforce standards that helped create a more uniform user experience across applications created by third parties.
- The web experience was primarily about finding/sharing information or completing a sales transaction. It was the “wild west” -lacking the standards and conventions of the desktop world.

Now these two worlds are converging. Desktop applications may seamlessly incorporate content from the Web – and use web conventions such as hyperlinks. Web-based applications provide functionality previously only found on the desktop. As these two worlds converge, new types of usability risks emerge.

In this essay, I discuss some common usability risks that we’ve seen with bringing desktop-line functionality to web applications.

Risk 1: Assuming desktop interactions can be translated directly to the web

One barrier to translating familiar interactions from the desktop to the web is the different expectations users bring to the web. Consider the case of one client who created a web

application by putting the commands (such as save, edit, and delete) in fly-out menus at the top of the page. Their logic was clear: after all, many sites (such as Circuit City) use fly-out menus. And users are familiar with command menus because the approach is standard in most desktop applications.

The problem was that on the web, users associate fly-out menus with navigation. They expect commands to appear as buttons or link on the page. In this particular case, users were clearly frustrated by the system as evidenced by the high volume of "how do I…? calls" received by client's call center. The solution here was to use a "button bar" – putting key commands directly on the page as buttons. Less frequently used commands were contained within a "More Actions" menu. This is an approach that a number of web-based applications have adopted, including Gmail and Yahoo Mail.

Dragging and dropping is an interaction that more web applications have adopted. However, as we mention in our 2006 Staff Picks of Usability Issues, our studies indicate that many users do not expect drag-and-drop to be available on the web. Drag-and-drop can also be quite mouse-intensive, particularly for frequently-performed tasks. For this reason, most desktop applications provide a command-based way to move objects from one location to another. Where possible, we recommend the same approach for web-based applications.

Risk 2: The return of interface "mystery meat" – this time for commands

Many early Web 1.0 sites were built by graphic designers, who placed a high value on site aesthetics, sometimes at the expense of site usability. One technique – using icon-only navigation schemes that revealed text labels only on mouse-over – was appropriately derided as "mystery meat navigation."

The Web 2.0 design aesthetic – with its emphasis on simplicity – has spawned a related technique where actions for an item on the page are only revealed on rollover. For example, early versions of Live.com only displayed Edit and Remove actions when the user rolled over the top bar of the module.

Figure 1

Figure 1: In early versions of Live.com the Edit and Remove actions were not visible unless the user rolled their mouse over the upper right of the module.

There are two risks with this interaction: 1) discoverability of the actions available 2) the two step movement that is required (first display the links, then click on the desired link). Thankfully, Microsoft subsequently decided on displaying these links persistently on each module—no rollover required.

Figure 2

Figure 2: Edit and remove actions are now more explicitly available for Live.com modules.

This is not to say that actions hidden until rollover are always a bad idea. As with dynamic navigation menus, the implementation can have a huge impact on usability. If the entire object is "hot" and creates a large target area for displaying the actions, discoverability is improved.

The additional benefit of displaying actions on rollover is that it creates a clear visual relationship between an object and its actions without unnecessarily cluttering the interface by persistently showing every action for every object. 37 Signals' Backpack ToDo lists provide an example of a well-done implementation.

Figure 3

Figure 3: With Backpack ToDo Lists, rolling over any part of a list item displays the Edit and Remove actions.

Risk 3: Inadequate transactional feedback

With increased page-level interaction, another key factor in creating a usable application is feedback. On the web, people are accustom to having a completed function take them to a different page in response -or at least re-load the current page. From a user experience perspective this is slow and clunky. However, people have become accustom to this and may miss feedback that occurs more quickly.

It is important that feedback be clear and bold. The consequences of missed feedback are potentially high. Where feedback isn't adequate we've seen people erroneously conclude that "nothing happened." This leads them to re-do a transaction unnecessarily, which produces the same unclear outcome, and ultimately leads the user to believe that the system isn't working.

Risk 4: Where desktop-like interactivity occurs users expect desktop-like performance

An important aspect of making dynamic displays usable is responsiveness. Sluggish performance can result in users blowing right by important functionality. Consider Google Suggests, which uses type-ahead functionality to streamline search term entry. Responsiveness is key to its usefulness. We've seen users completely miss similar functionality because the performance was sluggish. They were never aware the feature was available because they had finished their query before the entry suggestions displayed.

Scrolling is another function for which users have high expectations. For example, think about how an image library scrolls in the desktop environment, so that the images are visible at all points during the action. This is critical "way-finding" feedback. In our lab we've seen web-based scrolling where the page displays lags significantly behind the scrolling action. Essentially, this lets users perform functions that the system can't adequately support. It's very frustrating for users to try to find the "sweet spot" -the point at which they can perform the task as quickly as possible without breaking the functionality.

Conclusions

When designing interactions for web applications, think in terms of how you can make a function require less time, less physical effort, and less cognitive effort. Of course you should always be thinking this, but when adding interactivity that might not be expected on the web, it's important to double-check the benefits and risks.

Finally, don't discount the emotional appeal of a richer experience. Always playing it safe is a recipe for mediocrity. If you feel an approach is risky, make sure it gets on the "we need to test this list."