UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

Interazione Persona-Calcolatore

Design pattern di HCI

Prof.ssa Daniela Fogli
Dipartimento di Ingegneria dell'Informazione

Perché un linguaggio di design pattern di HCI

- Per aiutare i *singoli progettisti* a costruire interfacce migliori:
  - Ricorrendo all'**esperienza collettiva** degli altri progettisti
  - Permettendo di **pensare "outside the toolkit"**
  - Esprimendo degli **invarianti di progetto**
- Per aiutare *la comunità* di HCI:
  - Con un **vocabolario comune** per parlare delle caratteristiche che rendono un'interfaccia "vincente" o meno
  - A **isolare le qualità** che fanno il successo di certe metafore, di certi widget, …
  - A produrre una **solida base su cui costruire** nuovi strumenti e concetti

Differenza fra design pattern di SE e di HCI

- Pattern di progetto nel software engineering (SE):

  "A ***problem*** is stated in terms of desiderable qualities of the internal structure of behavior of software, and the ***solution*** is stated in terms of suggested code structures"

- Pattern di progetto di HCI:

  "A ***problem*** is stated in the domain of human-computer interaction issues, and the ***solution*** is stated in terms of suggested perceivable interaction behavior"

[Dearden & Finlay 2006]

Clear Entry Points

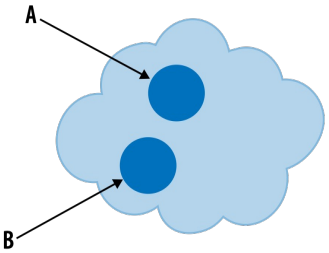


Figure 3-11. *Clear Entry Points schematic*

soluzione

What

Present only a few main entry points into the interface; make them task-oriented and descriptive. Use clear calls to action.

Use when

You're designing a site or app that has a lot of first-time or infrequent users. Most of these users would be best served by reading a certain piece of introductory text, doing an initial task, or choosing from a very small number of frequently used options.

However, if the purpose is clear to basically everyone who starts it, and if most users might be irritated by one more navigation step than is necessary (like applications designed for intermediate-to-expert users), this may not be the best design choice.

problema

Why

Some applications and websites, when opened, present the user with what looks like a morass of information and structure: lots of tiled panels, unfamiliar terms and phrases, irrelevant ads, or toolbars that just sit there disabled. They don't give the hesitant user any clear guidance on what to do first. "OK, here I am. Now what?"

For the sake of these users, list a few options for getting started. If those options match a user's expectations, he can confidently choose one and begin working—this contributes to immediate gratification. If not, at least he knows now what the site or app actually does, because you've defined the important tasks or categories up front. You've made the application more self-explanatory.

Varietà di sistemi interattivi

- Applicazioni d'ufficio
- Siti web
- Software gestionale
- Applicazioni per smartphone o tablet
- Interfacce operatore per controllo di processi industriali
- Video game
- Software multimediale per e-learning
- Strumenti per le arti visuali e la grafica
- Software per data visualization e visual analytics
- …

Variano per scopo, tipologia di utenti, grado di interattività, stili di interazione, contesti di uso etc., ma il loro progetto ha aspetti comuni

Aspetti comuni

- Organizzazione dei **contenuti** (Architettura dell'informazione):
  - Strutture o categorie per organizzare contenuti e funzionalità
  - Ad alto livello (tutta l'applicazione) e a basso livello (pagine, finestre, etc.)
  - Presentazione standardizzata con template e layout consistenti
- **Navigazione** nello spazio di lavoro
  - Differenti modi per navigare fra le informazioni
  - Strumenti di ricerca, esplorazione, filtraggio per trovare le informazioni
- Svolgere dei **compiti**
  - Flussi di lavoro o processi multi-step per svolgere i compiti
  - Fare azioni, dare comandi, inserire dati

I design pattern di Jenifer Tidwell

- Si rifanno ad Alexander: ***ambiente reale*** (architettura) vs. ***ambiente virtuale*** (sistema interattivo)
- Tidwell propone il suo primo linguaggio di design pattern di HCI nel 1999 come sito web: http://www.mit.edu/~jtidwell/common_ground.html
- Poi 3 edizioni (2005, 2010 e 2020) del libro:
  - **Designing Interfaces – Patterns for Effective Interaction Design, O'Reilly**
- I design pattern del libro costituiscono una **collezione** e non un vero e proprio linguaggio

Struttura dei design pattern del 1999

- Ogni pattern ha un **Nome** e contiene:
  - **Esempi**: sia buoni che cattivi (cioè che mostrano usi inappropriati del pattern)
  - Un **contesto** d'uso
  - Un **problema** che il progettista ha bisogno di risolvere
  - Un insieme di **"forze"** che spingono il progettista in diverse direzioni
  - Una **soluzione** o regola primaria su come tali forze possono essere tenute in conto per risolvere il problema
  - A volte alcune regole secondarie addizionali
  - A volte un **disegno stilizzato** della soluzione
  - Dei **riferimenti** ad altri pattern

Go Back One Step

Documenti ipertestuali

Examples:

- The "Back" button on a Web browser
- The "Back" button on a wizard
- Turning back a page in a physical book or magazine
- The "Undo" feature on some computer applications

Context: The artifact allows a user to move through spaces (as in Navigable Spaces), or steps (as in Step-by-Step Instructions), or a linear Narrative, or discrete states.

Problem: How can the artifact make navigation easy, convenient, and psychologically safe for the user?

Forces:

- Users tend to explore a navigable artifact in a tree-like fashion, going down paths that look interesting, then back out of them, then down another path.
- The user may want to temporarily look back at the previous space or state they were in.
- If the user gets into a space or a state that they don't want to be in, they will want to get out of it in a safe and predictable way.
- The user is more likely to explore an artifact if they are assured that they can easily get out of an undesired state or space; that assurance engenders a feeling of security.

Solution: **Provide a way to step backwards to the previous space or state.** If possible, let the user step backwards multiple times in a row, thus allowing them to backtrack as far as they want.

Resulting Context: Having a "back" function implies having a "forward" function; it's more of a convenience than a distinct pattern, but Web browsers have set up this expectation, so your users may be unpleasantly surprised if it's not there. Also, Go Back to a Safe Place is a logical pattern to use in addition to this one.

La struttura dei design pattern del libro

- Ogni pattern ha un **Nome** e contiene:
  - **What**: breve descrizione della soluzione (definizione del design pattern)
  - **Use when**: presenta i casi in cui va bene applicare il pattern
  - **Why**: descrive le ragioni per cui è bene applicare tale pattern (obiettivi e benefici)
  - **How**: spiega la **soluzione** in dettaglio, con indicazioni su come implementare il pattern. A volte contiene **riferimenti** ad altri pattern
  - **Examples**: mostra screenshot da sistemi noti

One-Window Drilldown

What

Show a list of items in a single screen or window. When the user selects an item from the list, show the details or contents of that item in the screen, replacing the list, as in the example in Figure 7-5.

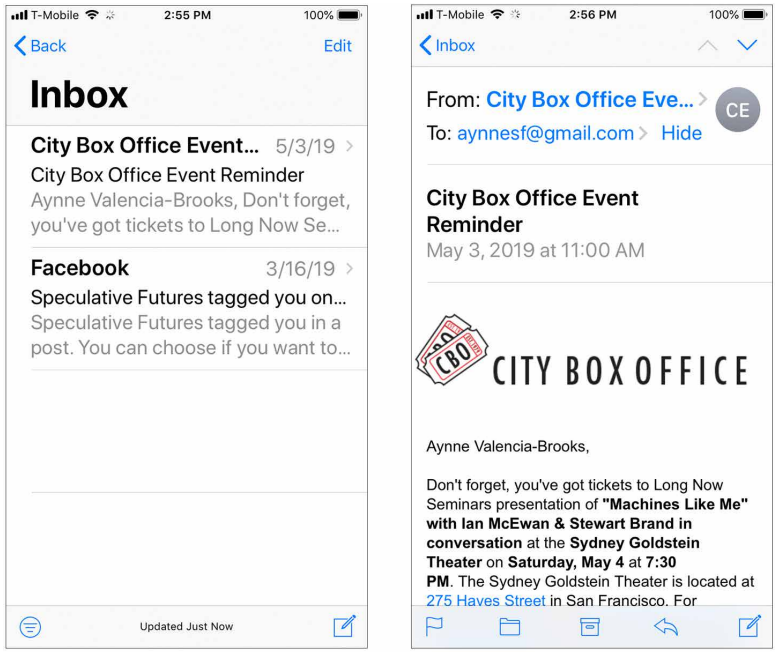


Figure 7-5. *Mac Mail on iPhone*

Use when

You are designing for mobile applications or websites. You have a list of items to show. Each item has interesting content associated with it, such as the text of an email message, a long article, a full-size image, or details about a file's size or date.

Alternatively, the list items and contents might just be large. You might need the entire screen or window to show the list, and again to show the contents of an item. Online forums tend to work this way, requiring the whole width of the screen to list conversation topics and a separate scrolled screen to show the conversations themselves.

Why

In a very constrained space, this might be the only reasonable option for presenting a list and item details. It gives each view the entire available space to "spread out" on the screen.

The shallow hierarchy of the one window drilldown pattern helps your user to not get too deep in the UI and makes it easy to return to the list where they started.

How

Create the list using whatever layout or format you find best—simple text names, cards, rows, trees, or outlines all work fine with *Thumbnail Grid*, as do other formats. Vertically scroll it if necessary, to fit it into the available space.

When the user clicks, taps, or otherwise selects one of the list items, replace the list display with a display of the item details or contents. On it, place a Back or Cancel button that brings the user back to the list screen (unless the platform supplies hardware buttons for such).

The item screen can offer additional navigational possibilities, such as drilling down further into the item details, stepping down into an item contained within that item (as in a hierarchy), or going "sideways" to the previous or next item in the list (as discussed in the next paragraph). In each case, replace the previous screen with the new one, and make sure the user can easily step back to the previous screen.

One disadvantage of this pattern is that to go from item to item, the user must "pogo-stick" between the list screen and the item screen. It takes a lot of clicks or taps to see more than a few items, and the user certainly can't flick between them quickly (as with *Two-Panel Selector*) or compare them easily (as with *List Inlay*). You can mitigate this problem by using Back and Next links to connect the user directly to the previous and next items in the list.

Examples

Examples abound in mobile design. Contrast the mobile version of a mail client shown in Figure 7-5 with its desktop counterpart. For instance, the *One-Window Drilldown* approach requires more text to be shown in the list, so the user has enough context to identify messages and triage them.

In the example in Figure 7-6 a reader can scroll through all the comments on a post in Reddit, and by clicking the back arrow in the header, easily return back to the Topic to view other threads.

...

Behavioral Patterns

- Nel libro, i design pattern sono preceduti da **Patterns in human behavior, perception, and thinking**
- **Non sono design pattern di HCI** ma servono per mettersi nei panni dell'utente
  - Safe Exploration – *"Let me explore without getting lost or getting into trouble"*
  - Instant Gratification – *"I want to accomplish something now, not later"*
  - Satisficing – *"This is good enough. I don't want to spend more time learning to do better"*
  - Changes in Midstream – *"I changed my mind about what I was doing"*
  - Deferred Choices – *"I don't want to answer that now, just let me finish!"*
  - Incremental Construction – *"Let me change this. That doesn't look right; let me change it again. That's better"*

Behavioral Patterns (cont.)

- Habituation – "*That gesture works everywhere else; why doesn't it work here, too?"*
- Microbreaks – *"I'm waiting for the train. Let me do something useful for 2 minutes"*
- Spatial Memory – *"I swear that button was here a minute ago. Where did it go?"*
- Prospective Memory - *"I'm putting this here to remind myself to deal with it later"*
- Streamlined Repetition – *"I have to repeat this how many times?"*
- Keyboard Only – *"Please don't make me use the mouse"*
- Social Media, Social Proof, and Collaboration – *"What did everyone else say about this?"*

Classificazione dei design pattern

2010

- Organizing the Content
- Getting Around
- Organizing the Page
- List of Things
- Doing Things
- Showing Complex Data
- Getting Input from Users
- Using Social Media
- Going Mobile
- Making It Look Good

2020

- Organizing the Content
- Getting Around
- Layout of Screen Elements
- Visual Style and Aesthetics
  (ma senza design patterns!
  Vedi dopo “Progettazione fisica”)
- Mobile Interfaces
- List of Things
- Doing Things
- Showing Complex Data
- Getting Input from Users

Alcuni esempi dal libro di Tidwell (3rd Ed.)

Design Pattern	Categoria
Feature, Search, and Browse	Organizing the content
Wizard	Organizing the content
Many workspaces	Organizing the content
Breadcrumbs	Getting around
Visual Framework	Organizing the page
Center Stage	Organizing the page
Grid of Equals	Organizing the page
Two-Panel Selector	Lists of Things
Button groups	Doing Things
Datatips	Showing Complex Data
Input Prompt	Getting input from users

Feature, Search, and Browse

What

A three-element combination on the main page of the site or app: a featured item, article, or product; a search box (expanded by default, or collapsed); and a list of items or categories that can be browsed.

Use when

Your site offers users long lists of items—articles, products, videos, and so on—that can be browsed and searched. You want to engage incoming users immediately by giving them something interesting to read or watch.

Alternately, your site focuses on enabling searching or transacting. In this case, search is the dominant element on the screen. Featured content and browsing have secondary importance.

Why

These three elements are found together on many sites. Searching and browsing go hand in hand to find desired items. Some people will know what they're looking for and use the search box, whereas others will do more open-ended browsing through the lists and categories you show them.

Featured items are how you engage the user. They're far more interesting than just category lists and search boxes, especially when you use appealing images and headlines. A user who lands on your page now has something to read or experiment with, without doing any additional work at all—and they may find it more interesting than whatever they originally came for.

How

Place a Search box in a prominent location, such as an upper corner, or in a banner across the middle top of the site. Demarcate it well from the rest of the site—use whitespace to set it off, and use a different surrounding background color if necessary.

Alternatively, display Search in a collapsed or compacted state. It still needs to be easy to see and access, but it can be an icon or the label "Search." Selecting this opens the full search field. This pattern saves space on smaller screens.

Set aside *Center Stage* (Chapter 4) for the featured article, product, or video. Very near it, and still at the top of the page, place an area for browsing the rest of the site's content. Most sites show a list of stories, cards, topics, or product categories. These might be links to pages devoted to those categories.

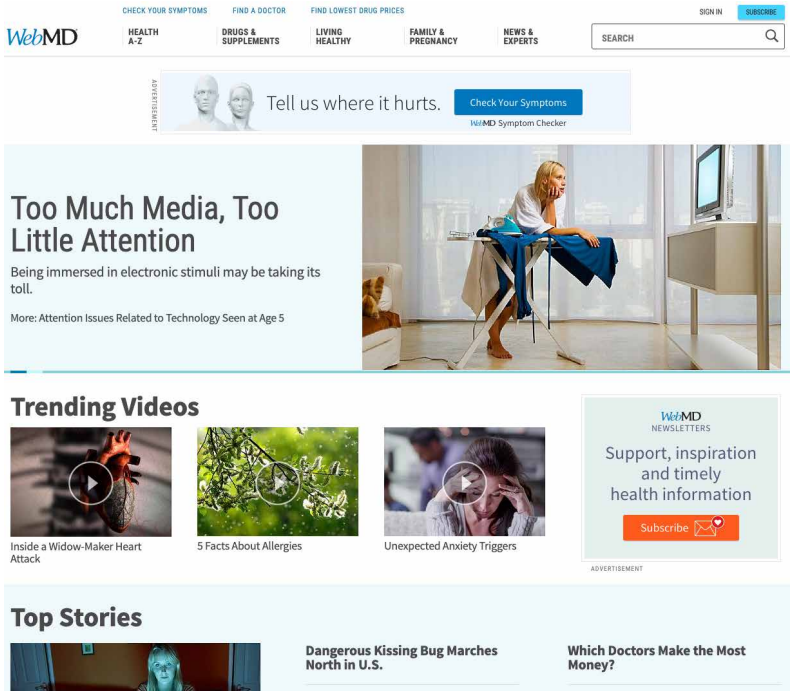
If the category labels open in place to show subcategories, the list behaves like a tree. Some sites, such as Amazon, turn the category labels into menus: when the pointer rolls over the label, a menu of subcategories appears.

Choose the features well. Features are a good way to sell items, advertise specials, and call attention to breaking news. However, they are the front door and also define what your site is about. What will they want to know about? What will capture their attention and hold them at your site?

As the user browses through categories and subcategories, help them "stay found" with the *Breadcrumbs* pattern (Chapter 3).

Examples

Content-centric websites. The following three examples demonstrate the classic pattern of *Feature, Search, Browse*. WebMD (Figure 2-2), Yahoo! (Figure 2-3), and Sheknows (Figure 2-4) are news- and content-centric digital publishers. WebMD and Yahoo! have Search at the top as a single large feature. Sheknows offers a variation: two features above a prominent search input.



Wizard

What

A feature or component that leads the user through the interface step by step to do tasks in a prescribed order.

Use when

You are designing a UI for a task that is long or complicated, and that will usually be novel for users—not something that they do often or want much fine-grained control over (such as the installation of a software package). You're reasonably certain that the designer of the UI will know more than the user does about how best to get the task done.

Tasks that seem well suited for this approach tend to be either branched or very long and tedious—they consist of a series of user-made decisions that affect downstream choices.

The catch is that the user must be willing to surrender control over what happens and when. In many contexts, that works out fine, because making decisions is an unwelcome burden for people doing certain things: "Don't make me think, just tell me what to do next." Think about moving through an unfamiliar airport—it's often easier to follow a series of signs than it is to figure out the airport's overall structure. You don't get to learn much about how the airport is designed, but you don't care about that.

But in other contexts, it backfires. Expert users often find a *Wizard* frustratingly rigid and limiting. This is particularly true for software that supports creative processes such as writing, art, or coding. It's also true for users who actually do want to learn the software; *Wizard* doesn't show users what their actions really do or what application state gets changed as choices are made. That can be infuriating to some people.

Why

Divide and conquer. By splitting up the task into a sequence of chunks, each of which can be dealt with in a discrete "mental space" by the user, you effectively simplify the task. You have put together a preplanned road map through the task, thus sparing the user the effort of figuring out the task's structure—all they need to do is address each step in turn, trusting that if they follow the instructions, things will turn out OK.

But the very need for a Wizard indicates that a task might be too complicated. If you can simplify a task to the point where a short form or a few button clicks can do the trick instead, that's a better solution. (Keep in mind, too, that a Wizard approach is considered a bit patronizing.)

How

"Chunking" the task. Break up the operations constituting the task into a series of chunks, or groups of operations. You might need to present these groups in a strict sequence, or not; sometimes there is value in breaking up a task into steps 1, 2, 3, and 4 just for convenience.

A thematic breakdown for an online purchase can include screens for product selection, payment information, a billing address, and a shipping address. The presentation order doesn't much matter because later choices don't depend on earlier choices. Putting related choices together just simplifies things for people filling out those forms.

You might decide to split up the task at decision points so that choices made by the user can change the downstream steps dynamically. In a software installation *Wizard*, for example, the user might choose to install optional packages that require yet more choices; if they choose not to do a custom installation, those steps are skipped. Dynamic UIs are good at presenting branched tasks such as this because the user never needs to see anything that's irrelevant to the choices they made.

In either case, the difficult part of designing this kind of UI is striking a balance between the sizes of the chunks and the number of them. It's silly to have a two-step *Wizard*, whereas one comprising 15 steps is tedious. On the other hand, each chunk shouldn't be overwhelmingly large, or you've lost some benefits of this pattern.

Physical structure. *Wizard*s that present each step in a separate page, usually navigated with Back and Next buttons, are the most obvious and well-known implementation of this pattern. They're not always the right choice, though, because now each step is an isolated UI space that shows no context—the user can't see what went before or what comes next. But an advantage of such a *Wizard* is that they can devote each page to that step completely, including illustrations and explanations.

If you do this, allow the user to move back and forth at will through the task sequence. Offer a way for the user to step backward or to otherwise change their mind about an earlier choice. Additionally, many UIs show a selectable map or overview of all the steps, getting some of the benefits of a *Two-Panel Selector*. (In contrast to that pattern, a *Wizard* implies a prescribed order—even if it's merely suggested—as opposed to completely random access.)

If you instead choose to keep all the steps on one page, you could use one of several patterns from Chapter 4:

- *Titled Sections*, with prominent numbers in the titles. This is most useful for tasks that aren't heavily branched because all steps can be visible at once.
- *Responsive Enabling*, in which all the steps are present on the page, but each one

Examples

The Microsoft Office designers have done away with many of its *Wizards*, but a few remain—and for good reason. Importing data into Excel is a potentially bewildering task. The Import Wizard (Figure 2-40) is an old-school, traditional application *Wizard* that guides the user step by step through the import process. It uses Back/Next buttons, branching, and no sequence map, but it works. Each screen lets you focus on the step at hand, without worrying about what comes next.

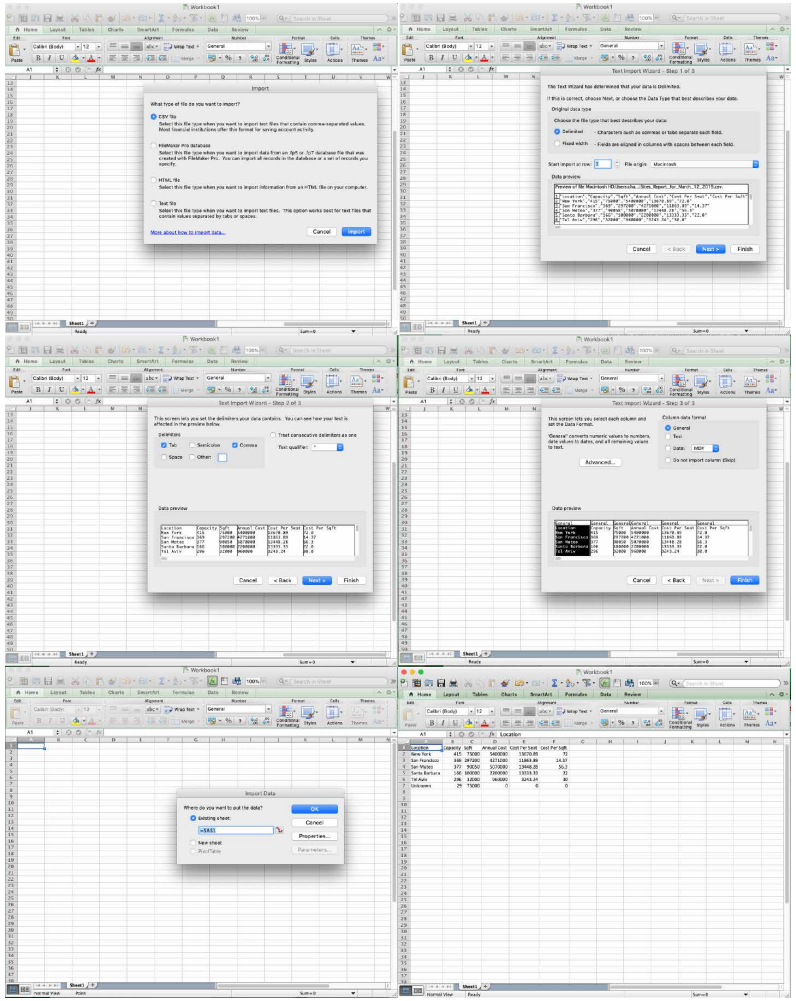


Figure 2-40. *The Microsoft Excel import wizard*

Many Workspaces

What

An interface where users can view more than one page, project, file, or context at a time. It can consist of multiple top-level tabs, tab groups, streams/feeds, panels, or windows. Users might have the option to place these workspaces side by side.

Use when

You're building an application that views or edits any type of content—websites, documents, images, or entire projects that include many files. A major aspect of choosing this pattern is the need to have different views or task "modes" available at the same time. For example, people often keep many browser tabs open at the same time so that they can switch between various websites, or compare them. Application developers and media creators often need to see and adjust code or controls in an editor window, and at the same time see the output of their work to see whether they are getting the desired outcomes—either as compiled and executed code, or as a rendered media object.

Designers of conventional websites don't generally need to think about this. All of the major browsers supply perfectly good implementations of this pattern, using tabs and browser windows. Spreadsheet applications such as from Microsoft or Google offer tabbed workspaces to separate a complicated workbook into individual calculation sheets.

Applications whose central organizing structure is a personal news stream might not need *Many Workspaces*, either. Email clients, personal Facebook pages, and so forth show only the one news stream that matters to the user; multiple windows don't add much value. That being said, email clients often let a user launch multiple email messages in different windows. Some Twitter applications can show several filtered streams side by side; for instance, they might show a search-based feed, then a feed from a custom list, and then a feed of popular retweets. (See the TweetDeck example in Figure 2-52.)

Why

People sometimes need to switch rapidly between different tasks in the same project or file, or monitor activity across a large number of real-time feeds.

People multitask. They go off on tangents, abandon trains of thought, stop working on task A to switch to task B, and eventually come back to something they left hanging. You might as well support it directly with a well-designed interface for multitasking.

Side-by-side comparisons between two or more items can help people learn and gain insight. Let users pull up pages or documents side-by-side without having to laboriously switch context from one to another.

This pattern directly supports some Chapter 1 patterns, such as *Prospective Memory* (a user might leave a window open as a self-reminder to finish something) and *Safe Exploration* (because there's no cost in opening up an additional workspace while leaving the original one where it is).

How

Choose one or more ways to show multiple workspaces. Many well-known applications use the following:

- Tabs
- Separate OS windows
- Columns or panels within a window
- Split windows, with the ability to adjust the splitters interactively

If you deal with fairly simple content in each workspace—such as text files, lists, or *Streams and Feeds*—split windows or panels work fine. More complex content might warrant entire tab pages or windows of their own so that a user can see a larger area at once.

The most complicated cases involve development environments for entire coding projects. When a project is open, a user might be looking at several code files, stylesheets, command windows (where compilers and other tools are run), output or logfiles, or visual editors. This means that many, many windows or panels can be open at once.

When users close some web browsers, such as Chrome, the set of workspaces (all open web pages, in tabs and windows) are automatically saved for later use. Then, when the user restarts the browser, their entire set of previously opened web pages is restored, almost as they left it. This is especially nice when the browser or machine has crashed. Consider designing in this feature; it would be a kindness to your users.

Examples

Both TweetDeck (Figure 2-52) and Hootsuite (Figure 2-53) take a multipanel or multistream approach to managing social media feeds.

TweetDeck is a *Streams and Feeds*–type application that can show many streams at once: filtered Twitter feeds, non-Twitter sources, and so on. The example in Figure 2-52 shows several typical TweetDeck columns. This maintains the spirit of a news stream by keeping all the updates visible at once; had these columns been in dif-

Breadcrumbs

What

Breadcrumbs refers to a specific type of navigation that shows the path from the starting screen down through the navigational hierarchy, the content architecture of the site, to the selected screen. The *Breadcrumbs* navigation pattern can be thought of as a series of parent—child links that show the drilldown into the information architecture of the site. The breadcrumbs show where in the content hierarchy the current screen is. Target (Figure 3-55) shows a common use of breadcrumbs on sites with large product directories.

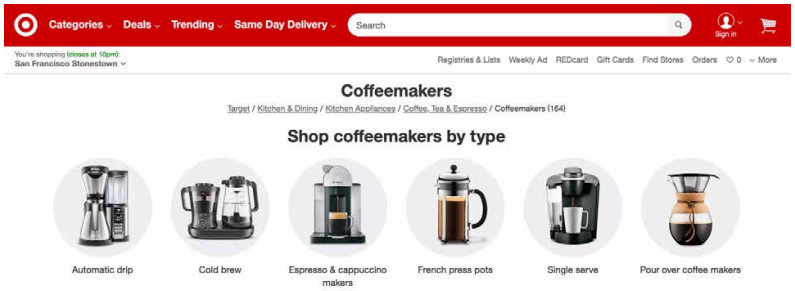


Figure 3-55. *Target breadcrumbs*

Use when

Your application or site has a hierarchical structure with two or more levels. Users move around via direct navigation, browsing, filtering, searching within the site, or deep-linking into it from elsewhere. Global navigation alone isn't sufficient to show a "You are here" signpost, because the hierarchy is too deep or large.

Alternatively, your site or app might have a set of browsing and filtering tools for a large dataset, such as products being sold online. The products are categorized in a hierarchy, but that categorization doesn't necessarily match the way people will look for those products.

Why

Breadcrumbs show each level of hierarchy leading to the current page, from the top of the application all the way down. In a sense, they show a single linear "slice" of the overall map of the site or app.

So, like a *Progress Indicator*, *Breadcrumbs* help a user to pinpoint where they are. This is especially handy if they've jumped abruptly to somewhere deep in the tree, as they would by following search results or a faceted browsing tool. Unlike a *Progress Indicator*, though, *Breadcrumbs* don't indicate to the user where they're headed next. They deal only with the present.

Some texts tell you that *Breadcrumbs*—so named for the Hansel and Gretel story, in which Hansel drops breadcrumbs on a forest trail to mark their way home—are most useful for telling the user how they got to where he is from the top of the site or app. But that's only true if the user has drilled straight down from the top, with no side-tracking, or following other branches, or dead-ends, or searching, or linking directly from other pages…not likely.

Instead, *Breadcrumbs* are best for telling you where you are relative to the rest of the app or site—it's about context, not just history. Look at the Target example in Figure 3-55. Faceted browsing—searching for items with certain characteristics—brought me to this page deep in the Target website. (A keyword search could have done the same.) But now that I'm here, I can see where I am in the product hierarchy and I know what else I can look at. I can use the *Breadcrumbs* to look at all of Target's stand mixers and do some comparison shopping.

Finally, *Breadcrumbs* are usually clickable links or buttons. This turns them into a navigational device in their own right.

How

On each page that is below a certain level in the navigational hierarchy—that is, it is deep in the content or screen or page architecture—show a list of all the parent pages, up to the main or home page. The goal is to see the parent-child relationships or what the "drill down" path is to get to the currently selected screen. Near the top of the page, put a line of text or icons indicating the current level of hierarchy. Start with the top level; to its right, put the next level and so on down to the current page. Between the levels, put a graphic or text character to indicate the parent—child relationship between them. This is usually a right-pointing arrow, triangle, greater-than sign (>), slash (/), or right angle quotes (»).

The labels for each page should be the page titles. Users should recognize them if they've been to those pages already; if not, the titles should at least be self-explanatory enough to tell the user what those pages are about. The labels should be links to those pages.

Some *Breadcrumbs* show the current page as the last item in the chain; some don't. If yours do, make them visually different from the rest of the items because they're not links.

Visual Framework

What

Across an entire app or site, all screen templates share common characteristics to maintain a consistent layout and style. Each page might use the same basic layout, margin, header and gutter size, colors, and stylistic elements, but the design gives enough flexibility to handle varying page content.

Use when

You're building a website with multiple pages or a UI with multiple windows—in other words, almost any complex software. You want it to "hang together" and look like one thing, deliberately designed; you want it to be easy to use and navigate.

Why

When a UI uses consistent color, font, layout, and when titles and navigational aids—signposts—are in the same place every time, users know where they are and where to find things. They don't need to figure out a new layout each time they switch context from one page or window to another.

A strong visual framework, repeated on each page, helps the page content stand out more. That which is constant fades into the background of the user's awareness; that which changes is noticed. Furthermore, adding enough character to the design of the visual framework helps with the branding of your website or product—the pages become recognizable as yours.

How

Draw up an overall look-and-feel that will be shared among all pages or windows. Home pages and main windows are "special" and are usually laid out differently from inner pages, but they should still share certain characteristics with the rest of the site, such as:

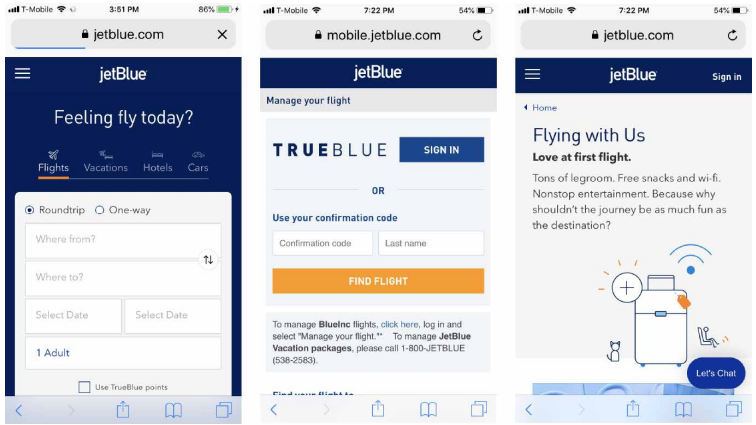
Color
: Backgrounds, text colors, accent colors, and other colors

Fonts
: For titles, subtitles, ordinary text, callout text, and minor text

Writing style and grammar
: Titles, names, content, short descriptions, any long blocks of text, and anything else that uses language.

In a *Visual Framework* design like that of JetBlue, shown in Figure 4-24, all pages or windows share the following, as appropriate:

- "You are here" signposts, such as titles, logos, *Breadcrumbs* trails, global navigation with indicators of the current page, and *Module Tabs*
- Navigational devices, including global and utility navigation, OK/Cancel buttons, Back buttons, Quit or Exit buttons, and navigational patterns such as *Progress Indicator* and *Breadcrumbs* (Chapter 3)
- Techniques used to define *Titled Sections*
- Spacing and alignment, including page margins, line spacing, the gaps between labels and their associated controls, and text and label justification
- The overall layout, or the placement of things on the page, in columns and/or rows, taking into account the margins and spacing issues listed previously



Implementation of a *Visual Framework* should force you to separate stylistic aspects of the UI from the content. This isn't a bad thing. If you define the framework in only one place—such as a CSS stylesheet, a Java class, or a visual system library—it lets you change the framework independently from the content, which means that you can more easily tweak and adjust it to get it as you want it. (It's also good software engineering practice.)

Center Stage

What

The task at hand is placed front and center at most times in the user experience. This type of layout puts the most important part of the UI into the largest subsection of the page or window, clustering secondary tools and content around it in smaller panels.

Use when

The screen's primary job is to show a single unit of coherent information to the user, let them edit a document, or enable them to perform a certain task. Other content and functions are secondary to this one. Many types of interfaces can use a *Center Stage*—tables and spreadsheets, forms, and graphical editors all qualify. So do web pages that show single articles, images, or features.

Why

The design should guide the user's eyes immediately to the beginning of the most important information (or task) rather than have them wandering over the page in confusion. An unambiguous central entity anchors the user's attention. Just as the lead sentence in a news article establishes the subject matter and purpose of the article, so the entity in *Center Stage* establishes the purpose of the UI.

After that's done, the user will assess the items in the periphery in terms of how they relate to what's in the center. This is easier for the user than repeatedly scanning the page, trying to figure it out. What comes first? What's the second? How does this relate to that? And so on.

How

Establish a visual hierarchy with the primary content or document dominating everything else. (See the chapter introduction for a discussion of visual hierarchy.) When designing a *Center Stage*, consider these particular factors, though none of them are absolutely required:

Size
: The *Center Stage* content should be at least twice as wide as whatever's in its side margins, and twice as tall as its top and bottom margins. (The user can change its size in some UIs, but this is how it should be when the user first sees it.) Keep "the fold" in mind—when a small screen is used, where does the content cut off at the bottom? Make sure the *Center Stage* still takes up more of the "above-the-fold" space than anything else.

Headlines
: Big headlines are focal points and can draw the user's eye to the top of the *Center Stage*. That happens in print media, too, of course. See the chapter introduction and *Titled Sections* for more.

Context
: What is the primary task of the product? This will inform what the user will expect to see when they open the screen. Is it a graphic editor? A long text article? A map? A filesystem tree?

Notice that I didn't mention one traditional layout variable: position. It doesn't much matter where you put the *Center Stage*—top, left, right, bottom, center; it can be made to work. If it's big enough, it ends up more or less in the center anyway. Note that well-established genres have conventions about what goes into which margins, such as toolbars on top of graphic editors, or navigation bars on the left side of web or mobile screens.

Examples

The Google Docs text editor (Figure 4-27) devotes almost all of its horizontal space to the document being edited; so does Google's spreadsheet editor. Even the tools at the top of the page don't take up a huge amount of space. The result is a clean and balanced look.

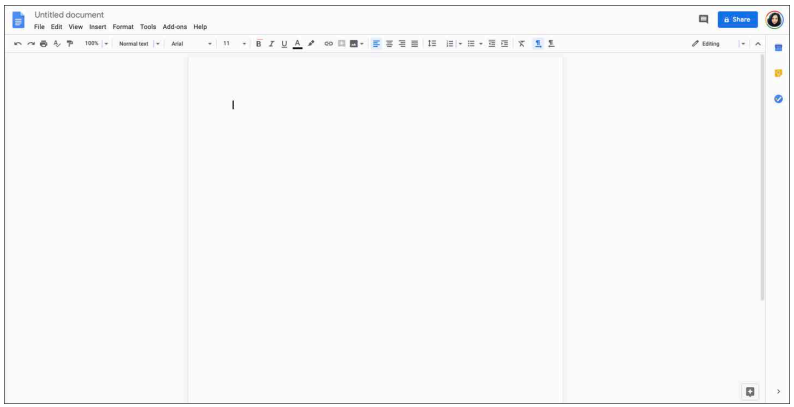


Figure 4-27. *Google Docs*

Grid of Equals

What

Arrange content items, such as search results, into a grid or matrix. Each item should follow a common template, and each item's visual weight should be similar. Link to item pages as necessary.

Use when

The page contains many content items that have similar style and importance, such as news articles, blog posts, products, or subject areas. You want to present the viewer with rich opportunities to preview and select these items.

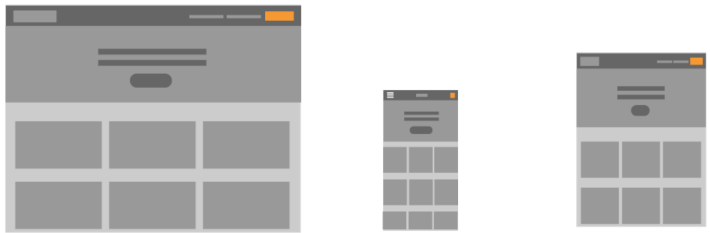
Why

A grid that gives each item equal space announces that they have equal importance. The common template for items within the grid informs the user that the items are similar to one another. Together, these techniques establish a powerful visual hierarchy that should match the semantics of your content.

How

Figure out how to lay out each item in the grid. Do they have thumbnail images or graphics? Headlines, subheads, summary text? Experiment with ways to fit all the right information into a relatively small space—tall, wide, or square—and apply that template to the items you need to display. Arrange content items in a grid or matrix. Each item should follow a common template, and each item's visual weight should be similar.

Now arrange the items in a grid. You could use a single row or a matrix that's two, three, or more items wide. Consider page width as you do this design work—what will your design look like in a narrow window? Will most of your users have large browser windows or use mobile or other devices (Figure 4-30)?



You might choose to highlight grid items statically (to emphasize one item over others) or dynamically, as a user hovers over those grid items. Use color and other stylistic changes, but don't change the positions, sizes, or other structural elements of the grid items.

Examples

In the Hulu example (Figure 4-31), the size and relative importance of each item in the grid is the same and has a consistent interaction behavior.

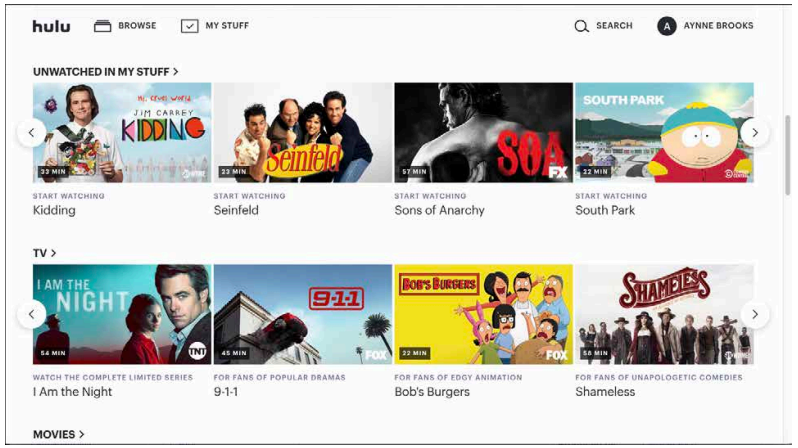
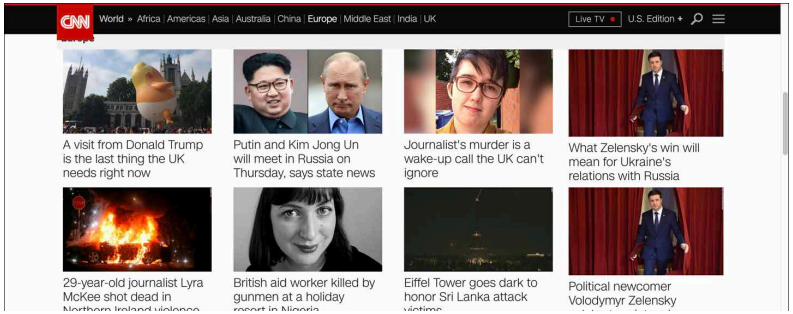


Figure 4-31. *Hulu grid*

In CNN's layout (Figure 4-32) and Apple TV's layout (Figure 4-33), the consistent visual treatment marks these items as peers of one another. An advantage of using grids to display lists of items is that a user of this interface will need to interact with only one grid item to understand how all of the grid items behave.



Two-Panel Selector or Split View

What

Also known as a *Split View*, this consists of two side-by-side panels on the interface. In the first one, show a list of items that the user can select at will; in the second one, show the content of the selected item, as demonstrated in Figure 7-1, which shows the Spotify website.

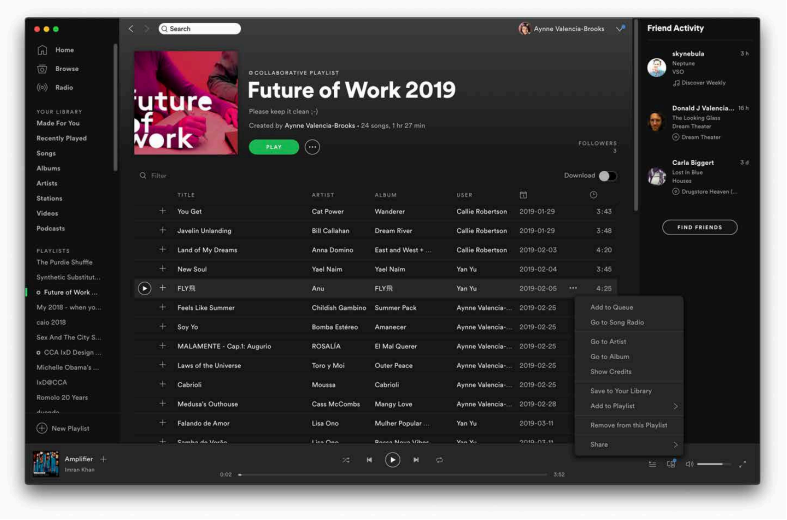


Figure 7-1. *Spotify*

Use when

You have a list of items to show. Each item has interesting content associated with it, such as the text of an email message, a long article, a full-sized image, contained items (if the list is a set of categories or folders), or details about a file's size or date.

You want the user to see the overall structure of the list and keep that list in view all the time, but you also want them to be able to browse through the items easily and quickly. People won't need to see the details or content of more than one item at a time.

Physically, the display you're working with is large enough to show two separate panels at once. Very small smartphone displays cannot cope with this pattern, but many larger mobile devices like tablets can.

Why

This is a learned convention, but it's an extremely common and powerful one. People quickly learn that they're supposed to select an item in one panel to see its contents in the other. They might learn it from their email clients or from websites; whatever the case, they apply the concept to other applications that look similar.

When both panels are visible side by side, users can quickly shift their attention back and forth, looking at the overall structure of the list ("How many more unread email messages do I have?"), and now at an object's details ("What does this email say?"). This tight integration has several advantages over other physical structures, such as two separate windows or *One-Window Drilldown*:

- It reduces physical effort. The user's eyes don't need to travel a long distance between the panels, and they can change the selection with a single mouse click or key press rather than first navigating between windows or screens (which can take an extra mouse click).
- It reduces visual cognitive load. When a window pops to the top, or when a screen's contents are completely changed (as happens with *One-Window Drilldown*), the user suddenly must pay more attention to what their now looking at; when the window stays mostly stable, as in a *Two-Panel Selector*, the user can focus on the smaller area that did change. There is no major "context switch" on the screen.
- It reduces the user's memory burden. Think about the email example again: when the user is looking at just the text of an email message, there's nothing on-screen to remind them of where that message is in the context of their inbox. If they want to know, they must remember, or navigate back to the list. But if the list is already on-screen, the user merely has to look, not remember. The list thus serves as a "You are here" signpost.
- It's faster than loading a new screen for each item, as can happen with *One-Window Drilldown*.

How

Place the selectable list on the top or left panel, and the details panel below it or to its right, as shown in Figure 7-2. This takes advantage of the visual flow that most users who read left-to-right languages will expect (so try reversing it for right-to-left language readers).

When the user selects an item, immediately show its contents or details in the second panel. Selection should be done with a single click. But while you're at it, give the user a way to change their selection from the keyboard, particularly with the arrow keys—this helps reduce both the physical effort and the time required for browsing, and contributes to keyboard-only usability.

... continua ...

Button Groups

What

A group of related actions as a small cluster of buttons, aligned and with similar graphic treatments. Multiple groups are possible if there are more than three or four actions.

Use when

There are many actions to show on the interface. You want to make sure they are all visible all the time, but you need to visually organize them so that they're not chaotic or difficult to sort out. Some of these actions are similar to one another—they have similar or complementary effects, for instance, or they operate with similar semantics—and they can thus be assembled into groups of two to five.

You can use *Button Groups* for app-wide operations (such as Open or Preferences), item-specific actions (Save, Edit, Delete), or any other scope. Actions with different scope ought not to be grouped together, however.

Why

Grouping buttons helps make an interface self-describing. Well-defined clusters of buttons are easy to pick out of a complex layout, and because they're so visible, they instantly communicate the availability of those actions. They announce, "These are the actions that are available to you in this context."

Gestalt principles (Chapter 4) apply here. Proximity hints at relatedness; if the buttons are all together, they probably do related things. So does visual similarity; if you make all the buttons the same dimensions, for instance, they look like they belong together. Conversely, button groups that are separated in space—or that are different in shape—imply unrelated groups of actions.

Proper sizing and alignment help the *Button Groups* form a larger composite visual shape (this is the principle of closure).

How

Make a group out of related buttons so that they form a natural or logical set. An additional option is to label them with short but unambiguous verbs or verb phrases. Use vocabulary that makes sense to the users. Do not mix buttons that affect different things or have different scope; separate them into different groups.

All buttons in the group should have the same graphic treatment: borders, color, height and/or width, icon style, dynamic effects, and so on. You can line them up in a single column, or arrange them in a single row if they aren't too wide.

(However, treat them differently if one action is a "primary" action, such as a Submit button on a web form. A primary action is an action that you want most users to take or that most users will expect to take. Give that button a stronger graphic treatment to make it stand out among the others.)

If all the buttons in a group act on the same object or objects, put the *Button Groups* to the left or right of those objects. You could put them below the objects instead, but users often have a "blind spot" at the bottom of complex UI elements such as multi-column lists and trees—the user might not see the buttons at all. To make them more visible, keep the rest of the interface clean and uncluttered. If you have a specific design that works better with the buttons at the bottom, test its usability and find out.

If there are enough buttons and if they have icons, you could also put them on a toolbar or ribbon-like strip at the top of the page.

By using *Button Groups*, you're trying to avoid a crowded mess of buttons and links, or perhaps a long and plodding list of actions with no apparent differentiation at all. With this pattern, you create a visual hierarchy of actions: the user can see at a glance what's related and what's important.

Examples

Standard tools for graphic editors are often grouped by function. Figure 8-2 shows some common tools in groupings in Google Docs (separated by vertical lines, or "pipes") that actually aid recognition. There are no fewer than 27 buttons on this interface. There's a lot to understand and keep track of. But thanks to careful visual and semantic organization, the interface is never overwhelming.

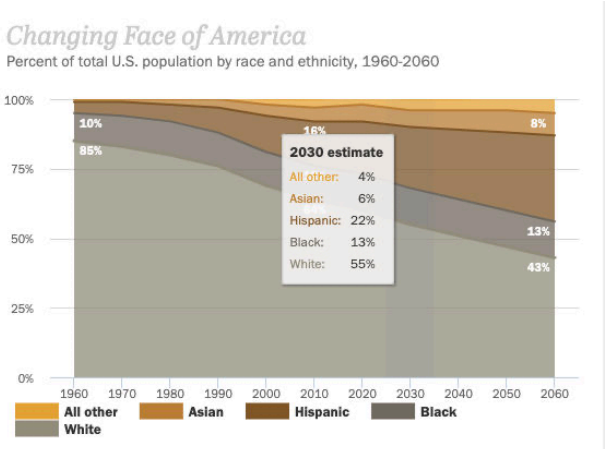
Figure 8-2. *Button Groups in Google Docs*

Datatips

What

Data values appear when your finger or mouse rolls over a point of interest on an interactive data table or when you tap or click the icon.

The Pew Research chart (Figure 9-11) example shows *Datatips* in action.



Use when

You're showing an overview of a data set, in almost any form. More data is "hidden behind" specific points on that graphic, such as the names of streets on a map or the values of bars in a bar chart. The user is able to "point at" places of interest with a mouse cursor or a touch screen.

Why

Looking at specific data values is a common task in data-rich graphics. Users will want the overview, but they might also look for particular facts that aren't present in the overview. *Datatips* let you present small, targeted chunks of context-dependent data, and they put that data directly where the user's attention is focused: the mouse pointer or fingertip. If the overview is reasonably well organized, users will find it easy to look up what they want, and you won't need to put it all on the graphic. *Datatips* can substitute for labels.

Also, some people might just be curious. What else is here? What can I find out? *Datatips* offer an easy, rewarding form of interactivity. They're quick (no screen loading), they're lightweight, and they offer intriguing little glimpses into an otherwise invisible data set.

How

Use a tool tip–like window to show the data associated with that point. It doesn't need to be technically a "tool tip"—all that matters is that it appears where the pointer is, it's layered atop the graphic, and it's temporary. Users will get the idea pretty quickly.

Inside that window, format the data appropriately. Denser is usually better, because a tool-tip window is expected to be small; don't let the window get so large that it obscures too much of the graphic while it's visible. And place it well. If there's a way to programmatically position it so that it covers as little content as possible, try that.

You might even want to format the *Datatips* differently depending on the situation. An interactive map might let the user toggle between seeing place names and seeing latitude/longitude coordinates, for example. If you have a few data sets plotted as separate lines on one graph, the *Datatips* might be labeled differently for each line, or have different kinds of data in them.

Many *Datatips* offer links that the user can click. This lets the user "drill down" into parts of the data that might not be visible at all on the main information graphic. The *Datatips* is beautifully self-describing—it offers not only information, but also a link and instructions for drilling down.

Analisi dei trade-offs

An alternative way of dynamically showing hidden data is to reserve some panel on or next to the graphic as a static data window. As the user rolls over various points on the graphic, data associated with those points appears in the data window. It's the same idea, but using a reserved space rather than temporary *Datatips*. The user must shift their attention from the pointer to that panel, but you never have a problem with the rest of the graphic being hidden. Furthermore, if that data window can retain its data, the user can view it while interacting with something else.

In contemporary interactive infographics, *Datatips* often work in conjunction with a *Data Spotlight* mechanism . The spotlight shows a slice through the data—for example, a line or set of scattered points— whereas the *Datatips* shows the specific data point that's under the mouse pointer.

Examples

The CrimeMapping example (Figure 9-12) shows an icon to indicate what kind of crime has occurred and plots this point on a map. A user can zoom in and out of the map and filter the nature of the crime by selecting "What" from the panel on the left.

Input Prompt

What

Prefill a text field with an example input or instructional text that helps the user with what to do or type. This is also called *placeholder text*.

Use when

The UI presents a text field, drop-down list, or combo box for input. Normally you would use a good default value, but you can't in this case—perhaps there is no reasonable default. The examples from the Blueprintjs UI Toolkit (Figure 10-19) show how prompts can help explain the function of the input field.

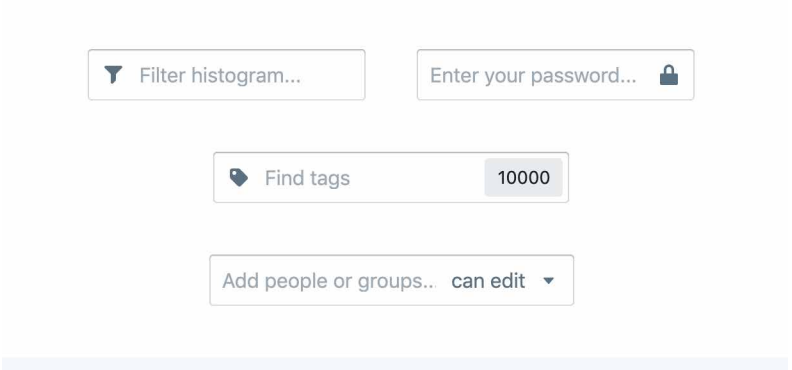


Figure 10-19. *Blueprintjs UI Toolkit; four example inputs with prompts*

Why

It helps make the UI self-explanatory. Like *Input Hints*, an *Input Prompt* is a handy way of supplying help information for controls whose purpose or format might not be immediately clear.

With *Input Hints*, someone quickly scanning the UI can easily ignore the hint (or miss it entirely). Sometimes this is your desired outcome. But an *Input Prompt* sits right where the user will type, so it can't be ignored. The advantage here is that the user doesn't have to guess whether they have to deal with this control or not—the control itself tells them to. (Remember that users don't fill out forms for fun—they'll do as little as needed to finish up and get out of there.) A question or an imperative "Fill me in!" is likely to be noticed.

Not the same as floating labels. Contemporary form design frequently uses "float labels" (see Brad Frost's article on Float Labels). This uses HTML label elements inside the form fields, very similar to input prompts, for the sake of elegance and simplicity. However, an input prompt disappears when the focus is on a form input that has prompt text. Floating labels move and change size but do not disappear on focus. Look again at Figure 10-15 (the Gmail registration screen); there are no labels outside the form inputs. The label "Phone number (optional)" and the next form input label "Recovery email address (optional)" are actually floating labels inside the input fields. In this case, input prompts are redundant. The phone number field has been selected, and instead of the text vanishing completely (formerly a huge drawback of input prompts), the label has moved up to the top edge of the input field. The user can still refer to it, and not need to remember it. On the other hand, only using floating labels can be problematic when you need to have both label text and input prompt text to explain different information.

https://bradfrost.com/blog/post/float-label-pattern/

How

Choose an appropriate prompt string:

- For a drop-down list, use Select, Choose, or Pick
- For a text field, use Type or Enter
- A short verb phrase

End it with a noun describing what the input is, such as "Choose a state," "Type your message here," or "Enter the patient's name." Put this phrase into the control where the value would normally be. (The prompt itself shouldn't be a selectable value in a drop down; if the user selects it, it's not clear what the software should do with it.)

Because the point of the exercise was to tell the users what they were required to do before proceeding, don't let the operation proceed until they've done it! As long as the prompt is still sitting untouched in the control, disable the button (or other device) that lets the user finish this part of the operation. That way, you won't need to throw an error message at the user.

For text fields, put the prompt back into the field as soon as the user erases the typed response.

Use *Good Defaults and Smart Prefills* instead of an *Input Prompt* when you can make a very accurate guess about what value the user will put in. The user's email address might already have been typed somewhere else, for instance, and the originating country can often be detected by websites.

UI Patterns for Web Design

Getting input

Forms
WYSIWYG
Password Strength Meter
Input Feedback
Calendar Picker
Structured Format
Input Prompt
Fill in the Blanks
Settings
Autosave
Expandable Input
Keyboard Shortcuts
Forgiving Format
Captcha
Drag and drop
Morphing Controls
Rule Builder
Preview
Inplace Editor
Good Defaults
Undo

Explaining the process
Wizard
Steps Left
Completeness meter
Inline Help Box

Community driven
Vote To Promote
Wiki
Pay To Promote
Rate Content
Flagging & Reporting

Navigation

Tabs
Navigation Tabs
Module Tabs

Jumping in hierarchy
Shortcut Dropdown
Notifications
Breadcrumbs
Modal
Fat Footer
Home Link

Menus
Vertical Dropdown Menu
Horizontal Dropdown Menu
Accordion Menu

Content
Carousel
Cards
Adaptable View
Event Calendar
Progressive Disclosure
Pagination
Article List
Tag Cloud
Continuous Scrolling
Archive
Categorization
Tagging
Thumbnail
Favorites

Gestures
Pull to refresh

Dealing with data

Tables
Table Filter
Sort By Column
Alternating Row Colors

Formatting data
Dashboard
Copy Box
Frequently Asked Questions (FAQ)

Images
Slideshow
Gallery
Image Zoom

Search
Search Filters
Autocomplete

Onboarding

Guidance
Blank Slate
Walkthrough
Coachmarks
Playthrough
Guided Tour
Inline Hints

Registration
Account Registration
Lazy Registration
Paywall

Social

Reputation
Collectible Achievements
Leaderboard
Testimonials

Social interactions
Follow
Friend list Mini
Activity Stream
Auto-sharing Mini
Chat
Friend
Reaction
Invite friends

Miscellaneous

Shopping
Product page
Pricing table
Coupon
Shopping Cart

Increasing frequency
Tip A Friend

UI Patterns
User Interface Design Pattern Library
by Anders Toxboe

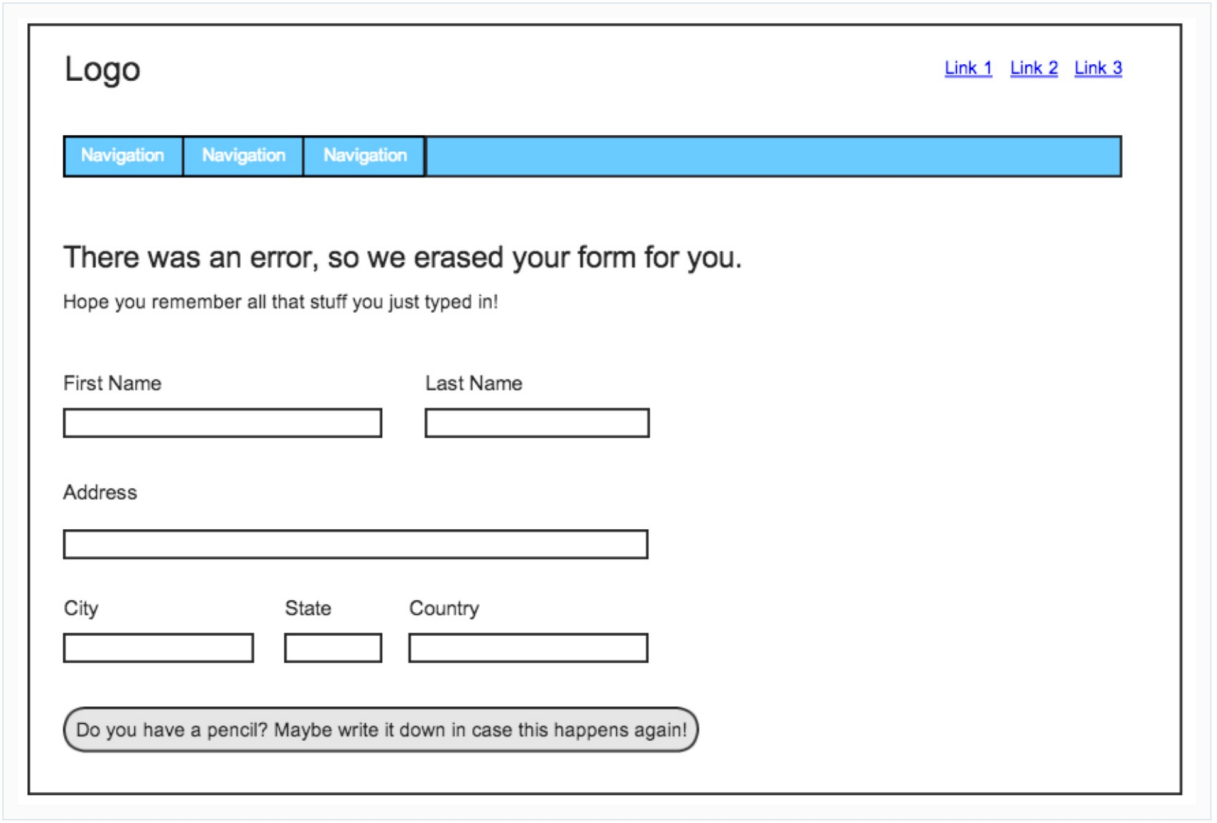
http://ui-patterns.com/patterns/

Pattern, anti-pattern e dark pattern

- I design pattern **evolvono** nel tempo man mano che gli utenti usano i sistemi basati su di essi e l'esperienza aumenta
- Le preferenze degli utenti possono cambiare e i designer possono inventare nuove modalità di interazione e presentazione
- Alcuni design pattern **possono diventare "deprecati"** nel senso che diventano obsoleti e non più considerati esempi di buona progettazione (es., il pattern Carousel – https://www.nngroup.com/articles/designing-effective-carousels/)
- Alcune idee di progetto vengono definite (o possono diventare nel tempo) **anti-pattern**, ovvero soluzioni di design da non imitare (es. "clicca qui" come pattern di navigazione, non spiega dove porterà il link)

Esempio di anti-pattern

Erasing information on error



Dark pattern

I **dark (o deceptive) pattern** non sono necessariamente soluzioni di design scadente ma sono usati per imbrogliare l'utente (per fargli fare o non fare qualcosa) - **https://www.deceptive.design/**

CONFIRMSHAMING

Confirmshaming is the act of guilting the user into opting in to something. The option to decline is worded in such a way as to shame the user into compliance. The most common use is to get a user to sign up for a mailing list, and it is often found in exit intent modals and other popups. In the screenshot below,, Amazon uses confirmshaming to guilt users and discourage them from opting out.

Un cenno alle applicazioni mobili

- Molto probabile che una applicazione venga utilizzata **primariamente (ed esclusivamente)** tramite dispositivo mobile
- Dispositivi che consentono una vera **manipolazione diretta** tramite touch: apprendimento facile e intuitivo
- **Non è più plausibile** procedere sviluppando una versione compatta/semplificata del sito web
- Ora le aziende usano uno dei seguenti approcci:
  - **Mobile-first**
  - **Responsive-design**

Progettazione in ambito “mobile”

- Analisi requisiti tramite
  - Scenari e personae
  - Interviste, questionari, focus group
  - *Shadowing*
  - *Diario d’uso*

Anche per valutazione

- Indagano cinque diversi “spazi”:
  - Spazio di informazione
  - Spazio del sè
  - Spazio delle relazioni
  - Spazio di intrattenimento
  - Spazio commerciale (m-commerce)

Sfide e opportunità

- **Schermi piccoli**
- Schermi **di dimensioni variabili**
- **Touch screen**
  - Serve spazio adeguato per selezione
  - Certi comportamenti non si possono realizzare
  - Varie tipologie di gesti (alcuni sconosciuti agli utenti)
- **Inserimento dei testi**
  - Importanza di autocompletamento e form pre-compilati

Sfide e opportunità (2)

- **Ambiente d'uso**
  - Le persone usano i dispositivi mobili in spiaggia, nel buio di un teatro, in macchina, nei negozi, sul bus, passeggiando
  - Design for "fat fingers" e per gli errori
- **Consapevolezza della posizione**
  - I dispositivi "sanno" dove li stiamo usando, ciò rende possible fornire informazioni in base alla specifica locazione
- **Influenze sociali e attenzione limitata**
  - Gli utenti prestano poca attenzione al contenuto di un'app (a meno che non stiano giocando): progetta per "l'utente distratto" e rendi tutto semplice e auto-esplicativo
  - Persone spesso coinvolte in conversazione, che mostrano ad altri quello che appare sul loro schermo, che devono spegnere la suoneria velocemente: progetta tenendo conto di queste situazioni sociali

Alcuni principi generali

1. Capire ciò di cui gli utenti hanno veramente bisogno
2. Togliere i contenuti ridondanti o poco utili
3. Se puoi, sfrutta l'hardware del dispositivo
4. Linearizza il contenuto
5. Ottimizza le sequenze di interazione più frequenti:
   - Elimina la digitazione
   - Limita il numero e la mole dei caricamenti
   - Riduci scrolling, soprattutto orizzontale
   - Riduci il numero di "tap"

Tecniche di design

- Solitamente fornite **linee guida** e system development kit (**SDK**) per garantire coerenza rispetto ad applicazioni della stessa piattaforma
- Uso di **emulatori** su PC
- Ma spesso emergono ugualmente problemi di usabilità e non solo…
- L'emulazione non consente di provare tutto
- Le prove sul dispositivo difficilmente si riescono a svolgere nel giusto contesto

L’ausilio dei design pattern

- Proposte diverse collezioni (no linguaggi) di **design pattern** per mobile computing
- Esempi:
  - Capitolo 6 - "Mobile Interfaces” di J. Tidwell et al. (2020)
  - T. Niel, *Mobile Design Pattern Gallery*, 2nd Edition, O’Reilly, 2014

I Design Pattern di Tidwell (3rd ed.)

- Vertical Stack
- Filmstrip
- Touch tools
- Bottom navigation
- Collections and Cards
- Infinite list
- Generous borders
- Loading or Progress indicators
- Richly connected apps

Esempi: Touch Tools e Richly Connected Apps

Touch Tools

What

Show certain tools in response to a touch or key press. Functions appear in small, dynamic overlays atop the content.

Netflix (Figure 6-12) is a digital product that allows viewers to watch video content. The intent is that the user will be mostly focused on the video content but might want to pause, turn captioning on and off, rewind or fast forward the video from time to time. The user can invoke the functionality by tapping the screen. The options go away again after about five seconds of nonuse.

Figure 6-12. *Touch Tools on the Netflix mobile application*

Use when

You are designing an immersive or full-screen experience, such as videos, photos, games, maps, or books. To manage that experience, the user will sometimes need controls—navigation tools, media player tools, information about the content, and so forth. The tools require significant space but are needed only sometimes.

… continua …

Richly Connected Apps

What

Use the features that come for "free" with your mobile device. Some examples of these are direct links to other apps, such as the camera, the phone dialer, map, or browser; and prefilling credit card passwords and address with data from the user's current context.

Use when

The mobile app shows data that is "connectable" in obvious ways, such as phone numbers and hyperlinks.

More subtly, your app can offer ways to capture images (via the device camera), sound, or video. It might even be aware of social networking conventions, such as Facebook or Twitter usernames. In all cases, your app might direct the user to another app to perform these device-based functions.

Why

A user can see only one mobile app at a time, even when multiple apps are being used at once, and it's annoying to switch between them by hand.

Mobile devices often have enough context and available functionality to offer intelligent paths between apps.

As of this writing, mobile devices have no good way to arbitrarily shuffle small amounts of information from one application to another. On the desktop, you can easily type, or use copy and paste, or even use the filesystem. You don't have those options on a mobile platform. So, you need to support moving that data automatically.

How

In your app, keep track of data that might be closely associated with other apps or services. When the user taps or selects that data, or uses special affordances that you provide, open another app and handle the data there.

Here are some examples. Consider all the ways that data in your app can connect directly to other mobile functions:

- Phone numbers connect to the dialer
- Addresses connect to the map, or to the contacts app
- Dates connect to the calendar

… continua …

Per il vostro progetto

- La progettazione o ri-progettazione del vostro sistema dovrebbe **fare riferimento ai design pattern** (scegliendo la/le collezione/i che si preferisce/preferiscono)
- I design pattern possono ispirare soluzioni a cui non si era pensato
- Se il progetto originario implementava (anche inconsapevolmente) dei design pattern sarebbe utile **evidenziare come la loro implementazione è cambiata o se si è preferito effettuare altre scelte di design**