while under way, making it equally easy to get back. An elegant solution is the zooming interface paradigm (ZIP), which in many situations solves the navigation problem and also provides a way around the problem of the limited screen real estate that any real display system must confront. Imagine, if you will, how readily mazes could be solved if only you could fly above them, see their layout, and go directly to your destination. A zooming interface paradigm offers that kind of fluidity and facility for many tasks you perform with computers. Although a ZIP is not optimal for all situations, I will concentrate on its positive aspects, with the aim of demonstrating that there are alternatives superior to desktop-oriented graphical user interfaces.

The ZIP described here is called ZoomWorld and is based on the idea that you have access to an infinite plane of information having infinite resolution. The plane is ZoomWorld. Everything you can access is displayed somewhere on ZoomWorld, whether it is on your computer, on a local network to which your computer is attached, or on a network of networks, such as the Internet.

To see more of ZoomWorld, you think of yourself as flying higher and higher above it. To look at a particular item, you dive down to it. ZoomWorld also has a content searching mechanism. The overall metaphor is one of flying, climbing to zoom out and diving to zoom in. You navigate both by flying above ZoomWorld and by doing content searches.

ZoomWorld is conceptually similar to the walls of a project planning room. After a while, the walls become covered with annotations, tacked-up sheets of paper, sticky notes, photos, or whatever else helps us to remember and to explain our ideas. When you come into the room, you may stand in the center and glance around, spot a place, walk over, and look at the material in detail. At other times, you walk directly to the place where the information you want to review or modify has been posted.

We can find things in such a planning room because we tend to remember landmarks and relative position, a fact sometimes called the psi effect and long known to psychological researchers. "The stuff about marketing is on the right wall, sort of lower down near the far corner," someone might tell you. On another occasion, you go right to a particular document because you remember that it is just to the left of the orange piece of paper that Aviva put up. As you work in the room, you sometimes step back to get your bearings. You don't have to step back far before you can no longer read the small print, but you can see headings and larger diagrams. From still farther away, you can make out only a few of the largest headings, colors of whole sheets, layouts of areas, and tell whether there are illustrations,

diagrams, or cartoons on them, even if you can't make out just what is being illustrated or diagrammed.

The ZIP readily permits labels to be attached to images and to collections of images yet does not impose any structure, hierarchical or otherwise, beyond association due to proximity. I suspect that most users will tend to organize collections of images into clusters and that some will create informal hierarchies. For example, a large heading Personal Photos might, when zoomed in on, reveal smaller headings on groups of pictures labeled Baby Pictures, Vacations, Pets, Hobbies, Friends, Relatives, and so forth. Zooming in to the writing under the heading Baby Pictures might reveal the children's names, Agatha, Gideon, and Hermione. A professional photographer is likely to have a very carefully constructed and organized collection. Note that you are not required to remember the names; you find them as you zoom and glide about and decide whether the image you want is likely to be in the named category. The same can be done with films and collections of sound, although you may have to activate a film or a sound to make sure that it is the one you want.

The totally disorganized have hope: If you are one such, you can just lay out the images any old way and zoom around until you find the one you want. You may remember its approximate position due to your spatial memory, which will speed the search.

If you want larger characters, say, in a document you are reading, it is clear that zooming in makes the characters larger, but you may not be able to fit entire lines into the width of the display, making reading from the screen difficult. The solution in this case is to use a command to enlarge the font, making the line length, in terms of number of characters, shorter.

Our inherent tendency to remember position and landmarks can be put to work in ZoomWorld. The method of getting around will not be via scroll bars, which are slow. (Just consider, in Fitts' law terms (Section 4-4), the size of the arrow boxes you must click on.) Nor will it be by clicking on zoom-in and zoom-out icons or menu items, which are also slow, but rather by emulating what a person does in a planning room: stepping back to view large areas, then walking up to just the one that is desired, and finally leaning forward to read the fine print or using a magnifying glass to see a detail in a photo.

Zooming in, so that you can see portions of the space in detail, is probably best implemented as a quasimode (see Section 3-2-3). In this quasimode, the graphical input device button causes a real-time zoom-in to wherever the cursor is located: The center of zooming follows the cursor. The secondary graphical input device button can be used for this function.

For compatibility with the application-free systems described in Sections 5-7 and 5-8, the zoom quasimodes can be positioned elsewhere, such as on a second graphical input device, on the keyboard, or as dedicated, labeled extra buttons on the graphical input device (see Appendix A).

Wherever the zooming controls are located, the point you are zooming in on is the cursor location, which can be adjusted during the zoom by the graphical input device, being used in its role as a positional pointing device. That is, during zooming, the system moves the ZoomWorld plane so that the cursor position is at or near the center of the display. If zooming is quick—at least a factor of 2 in linear dimensions per second—and has the appearance of being continuous, zooming to and from the cursor position will suffice as a graphical navigational tool.

In the planning room, you might put up larger signs over the main areas, signs you can read from anywhere in the room. Zooming operates similarly; the sizes of headings and text determine how far in you have to zoom before you can see details. This technique substitutes for, and improves on, hierarchical directories. A fast text search, such as LEAP, is an important adjunct, used when you need to find a textual detail quickly. A few distinctive geometric landmarks can also be used; see the rules for effective use of icons in Section 6-3 for what is meant by "few." A large red cross, for example, might signal an area containing emergency medical data—in both the planning room and in a ZIP.

Without having the phenomenon pointed out to them, users quickly become familiar with the textures of the kinds of work they do. Spreadsheets, tables, text, bitmapped images, drawings, and other products of computer use each have their own visual characteristics even when seen from afar. The products of various coworkers, groups, vendors, and other creators of material will often be immediately recognizable.

A zooming space gives you a great deal of flexibility in layout. If a document grows in length or width, it can simply decrease its scale uniformly to make the entire document fit within the same region it did before its content increased. Similar space-making tactics can be used when a table or a spreadsheet is enlarged. Because you can zoom in, the text can always be made large enough to read. The opposite occurs when a document or drawing gets smaller. Blank space is unlimited, if the system is implemented properly; a command allows a new document to be started anywhere—by copying a blank document, the ZIP analog of the GUI New command—and documents can be allowed to overlap adjacent documents or be restricted in their growth by them. Internal links and pointers to web sites (URLs) can bring you immediately to other documents at a place and size

determined at the creation of the link. Buttons can have as much information on them as you wish, including a complete manual, with examples, without affecting their simplicity when viewed from afar. Every feature can have an explanation built into it.

Zooming can also be nonlinear in time, starting more slowly and accelerating to full zoom speed, allowing for fine control of small changes without impeding large changes in scale. It can also slow down and briefly stop at certain predefined zoom ratios, making it easy to get characters to standard sizes.

Working effectively in a ZIP often permits, and sometimes requires, strategies that are unlike those used in desktop-based GUIs. For example, you can have a number of views of the same data kept on hand because real estate poses no limitations. As with the efficient temperature converter solution in Section 4-3-2, we let the machine do extra work or preparation that may never be used, in the interest of easing human workload. Comparisons can be effected by a split screen, which requires controls for splitting but that allows for independent zooming, and by a typical ZIP trick: moving copies of the two documents or two copies of the same document so that the parts to be compared are adjacent. The ability to move and to scale any object certainly applies to text objects, which is how you can make a document bigger and thus visible at a higher hierarchical (and graphical) level.

A footnote can be more than just a reference. You can zoom in to the entire referenced work. Zooming here is functioning like a link, except that to get back to the main discussion, you zoom out; you do not have to keep a trail of where you have been. To make it easy to find a set of documents, the documents themselves can be arranged in a distinctive pattern that is visible when zoomed out. A page with very large lettering can be seen while zoomed out and used as a title. Zooming out can serve in the role of a Back button in a browser.

Small-scale collaborative endeavors are easily modeled by having collaborators all zoom on the same document, with appropriate rules to prevent interference. A network can be represented as a space in which everyone's work is placed in his or her own region. In a ZIP collaborative space, you can choose to keep as much of your work visible or invisible to others as you wish. Invisible documents are also a way to implement varying levels of access permission.[3]

---

3. An interesting zooming user interface (ZUI), called PAD++ (it is now called *Jazz*), has been developed independently, originally at the University of New Mexico. See http://www.cs.umd.edu/hcil/pad++/. I am grateful to Dr. Donald Norman, then at Apple, for pointing out this work to me.



Figure 3.4. Fluke scopemeter: powerful and rugged, zillions of functions, a paucity of keys, but difficult to learn and sometimes slow to get to the function you need. On the other hand, its color-coded leads eliminate a traditional source of confusion about which lead does what.



Figure 3.5. The Tektronix oscilloscope has a myriad settings, but although initially intimidating, it is quick and easy to operate. All knobs that can uncalibrate the unit are in red.



Figure 3.7. The Sony 2010 multi-band radio achieves ease of use by providing thirty-two dedicated buttons for presets. When a preset is tapped, it simultaneously sets all parameters to the state they were when the user stored the station. Unlike many radio preset systems, which assign presets only with one particular band (for example, AM or FM), any button may access any station on the Sony 2010.

Storing stations is also easy: A dedicated and labeled SET button establishes a quasimode during which a press of a preset button stores the current station and all other pertinent operating parameters. It is nearly impossible to accidentally store a station when you intend to listen to one, and the label makes the SET function visible.

It is rare when a product combines both superior electronic technology and interface technology. Perhaps this explains why the Sony 2010 has been in continuous production for over a decade in an industry where most models fade in months.



Figure 3.9. Canon Cat's Use Front key and some of the command keys. The words USE FRONT are light blue, as are the legends on the fronts of the keys that are enabled by the Use Front key.

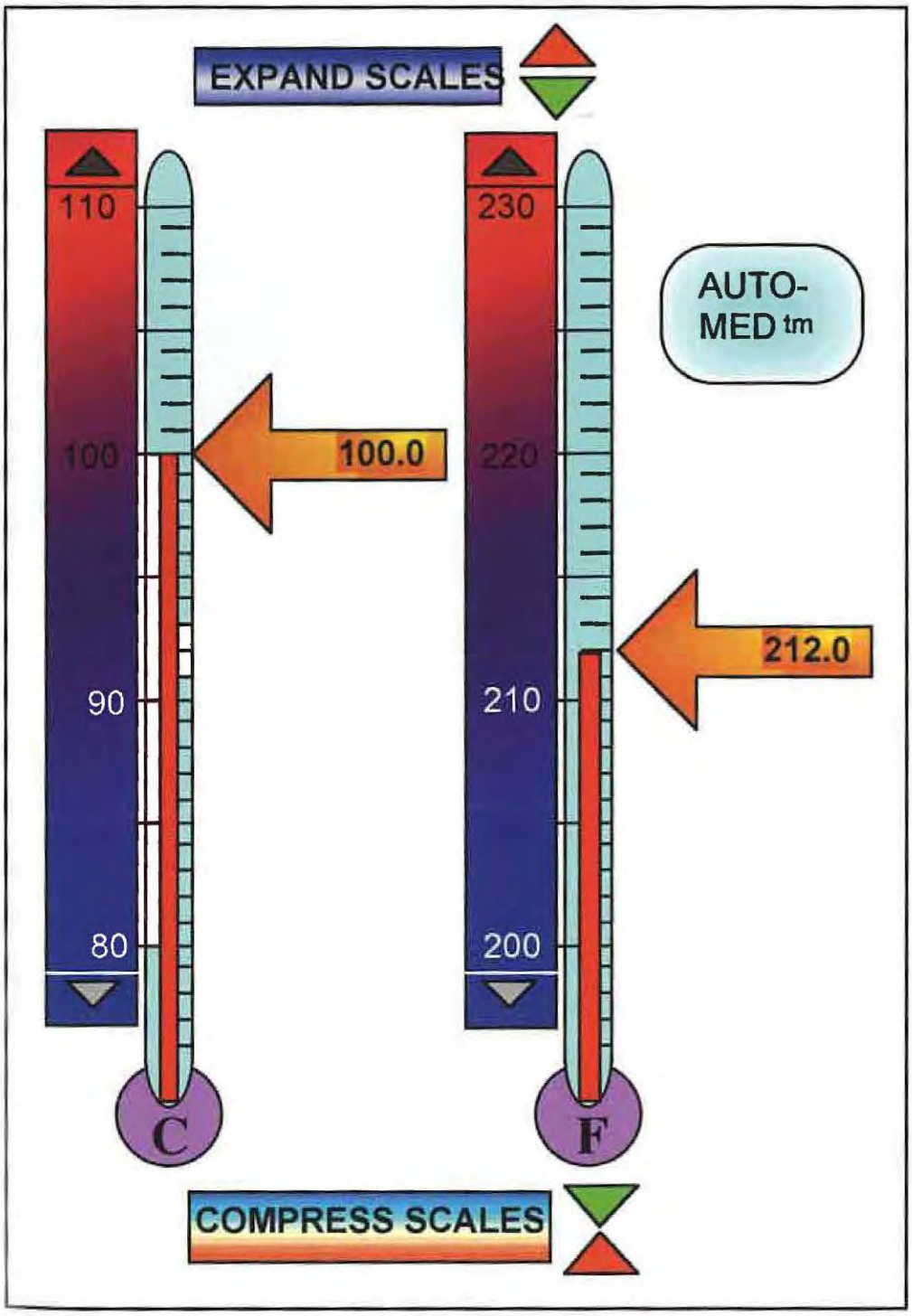


Figure 4.3. A GUI for Hal's interface.



Figure 4.7. The Macintosh menu, at the top edge of the screen, effectively increases its size compared to a menu that floats away from the edge.

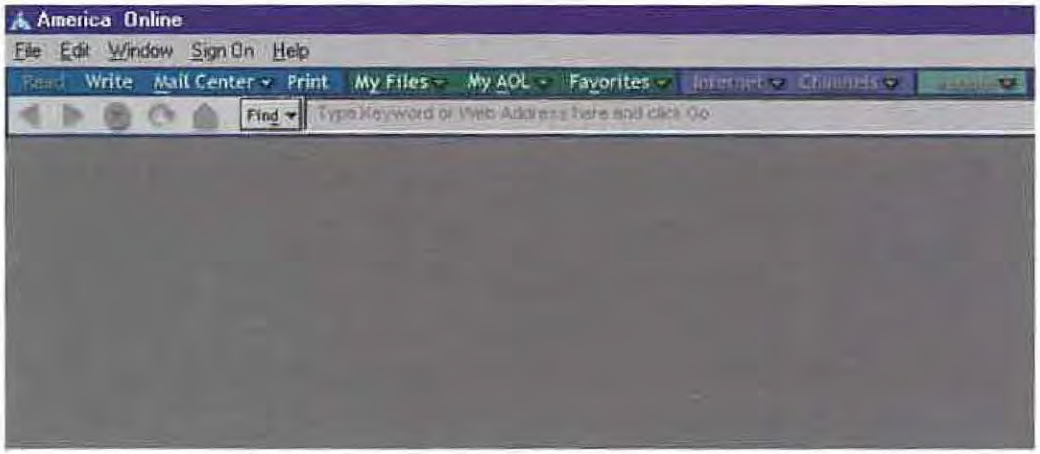


Figure 4.8. The Windows menu is below the top edge of the screen; you have to place the cursor more carefully to pull down a submenu.



Figure 6.1. A ZIP view of an ICU. The numbered rectangles are rooms, and the names are those of the patients in them (no real data shown).

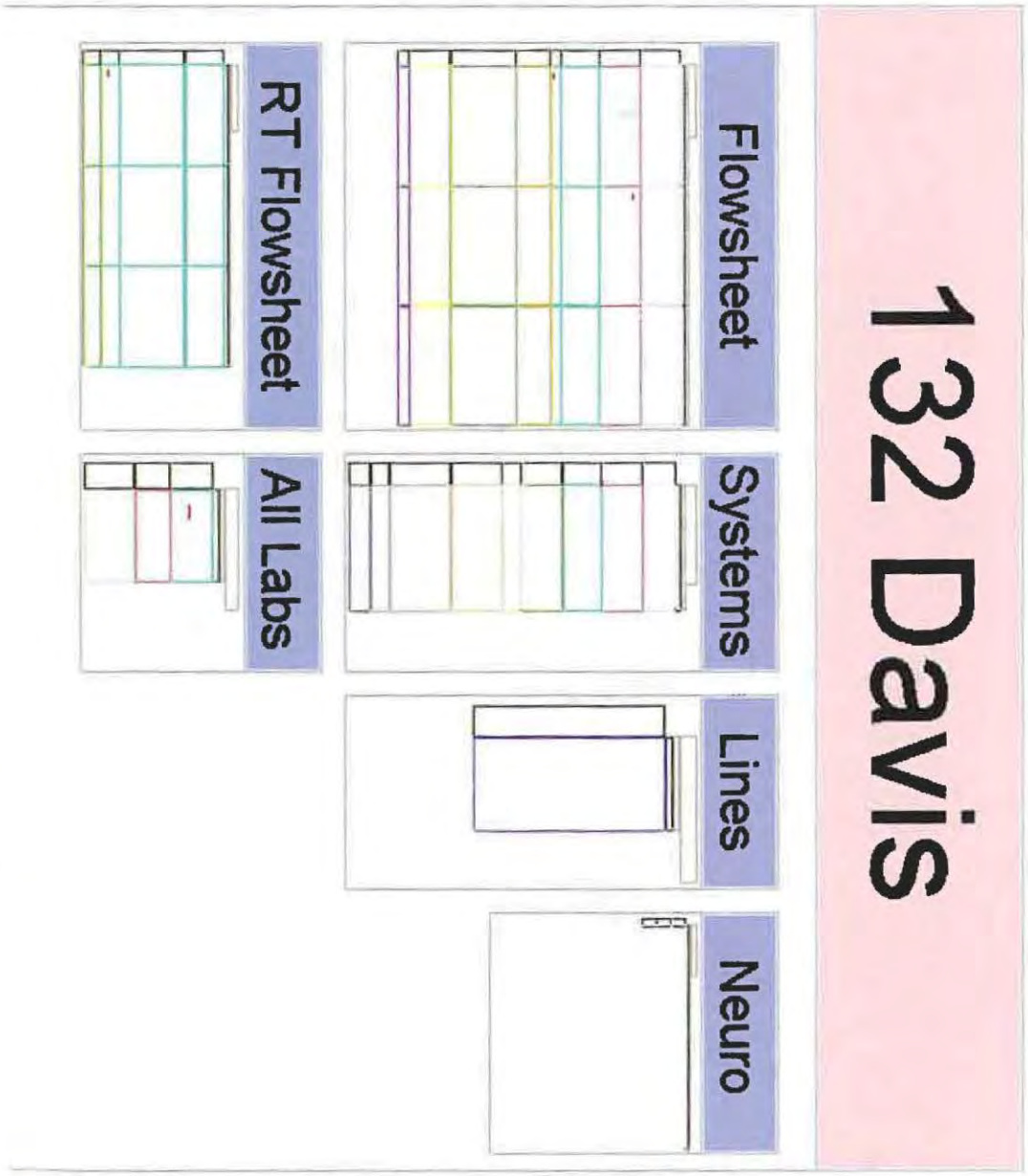


Figure 6.2. The charts for a particular patient can be seen when zoomed in this far.

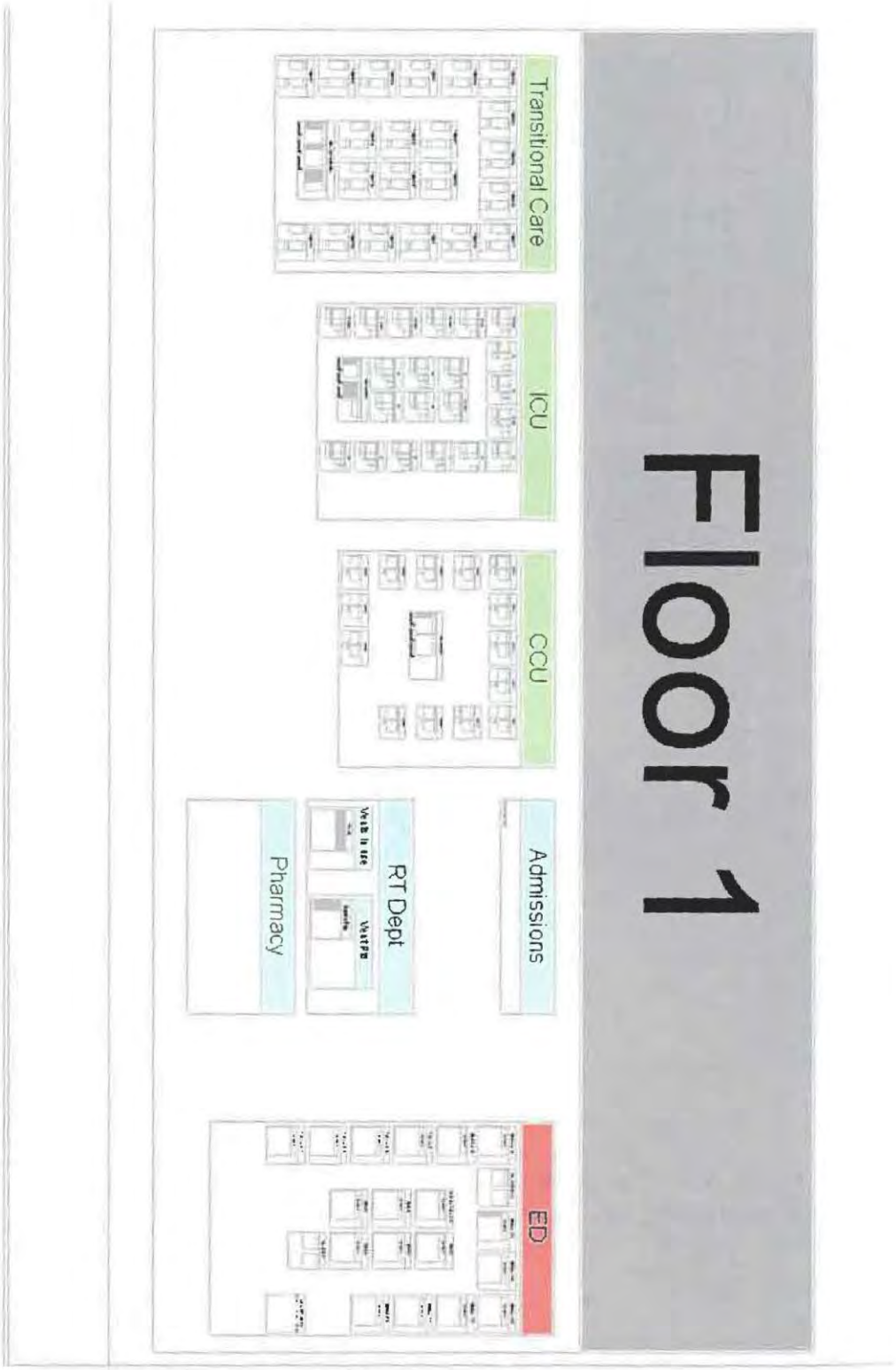


Figure 6.4. Zoomed out beyond the view shown in Figure 6.1, the entire first floor is visible.

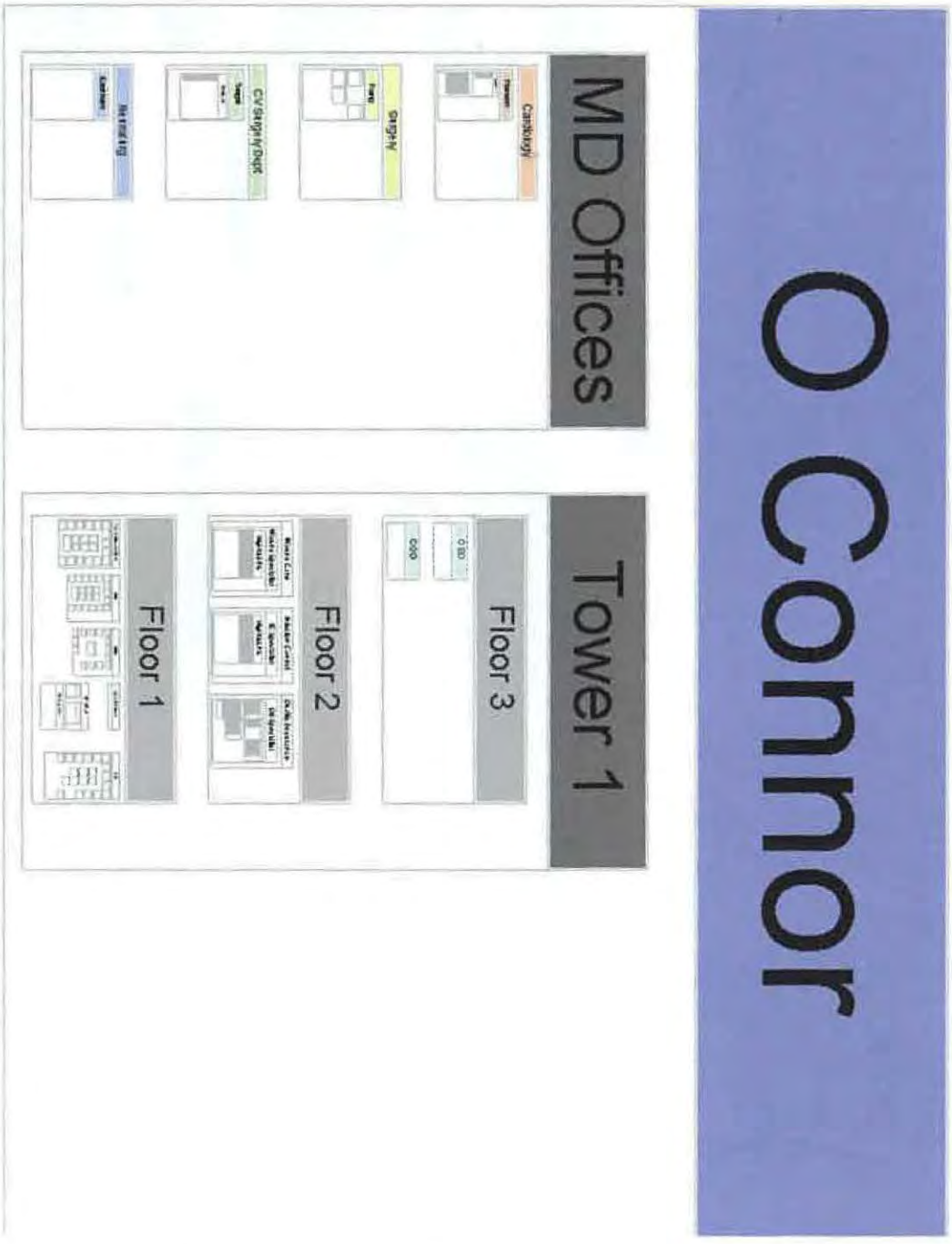


Figure 6.5. The entire O'Connor hospital can be seen.

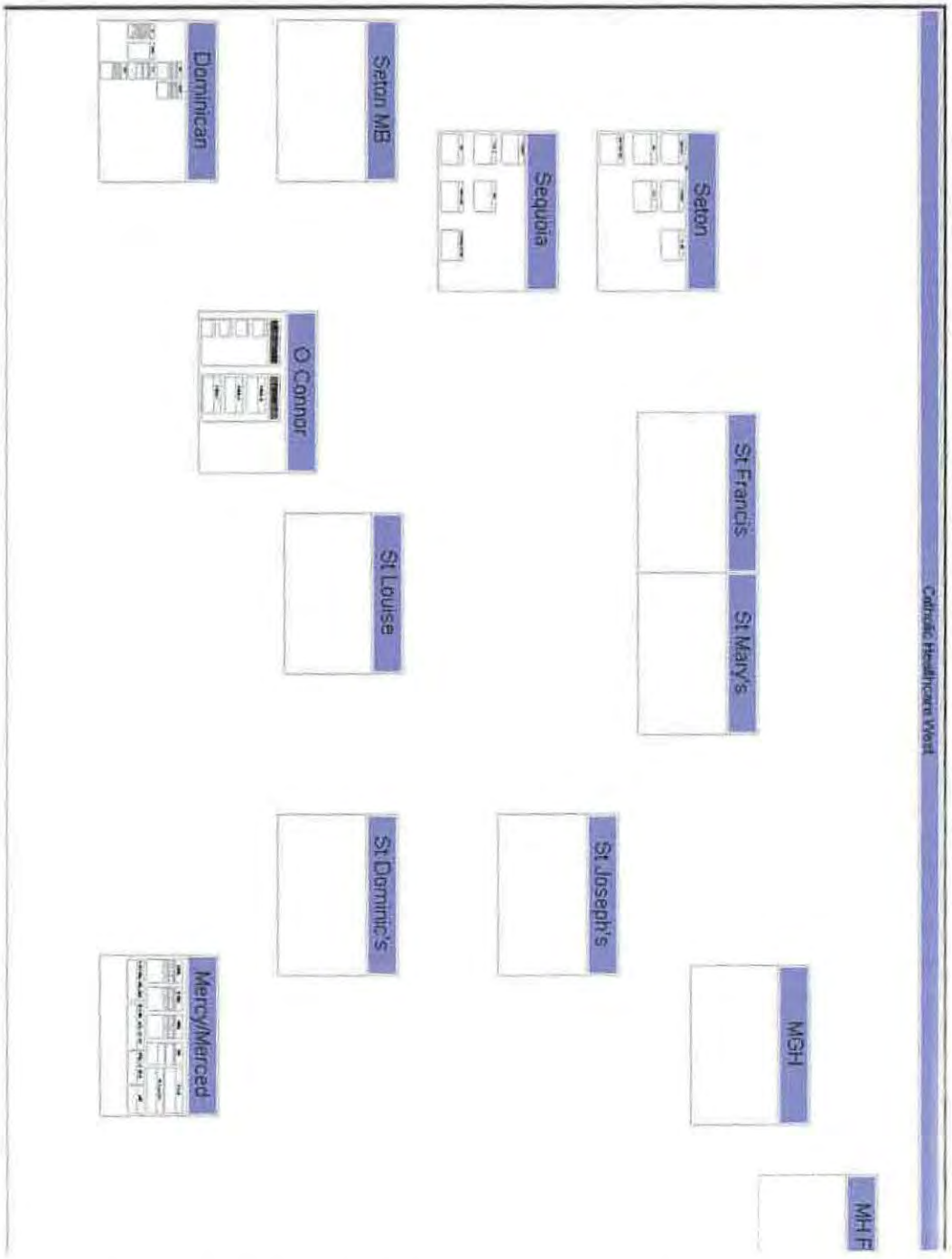


Figure 6.6. The entire enterprise can be accessed from this level.



Figure 6.7. A ZoomWorld portal to the World Wide Web.

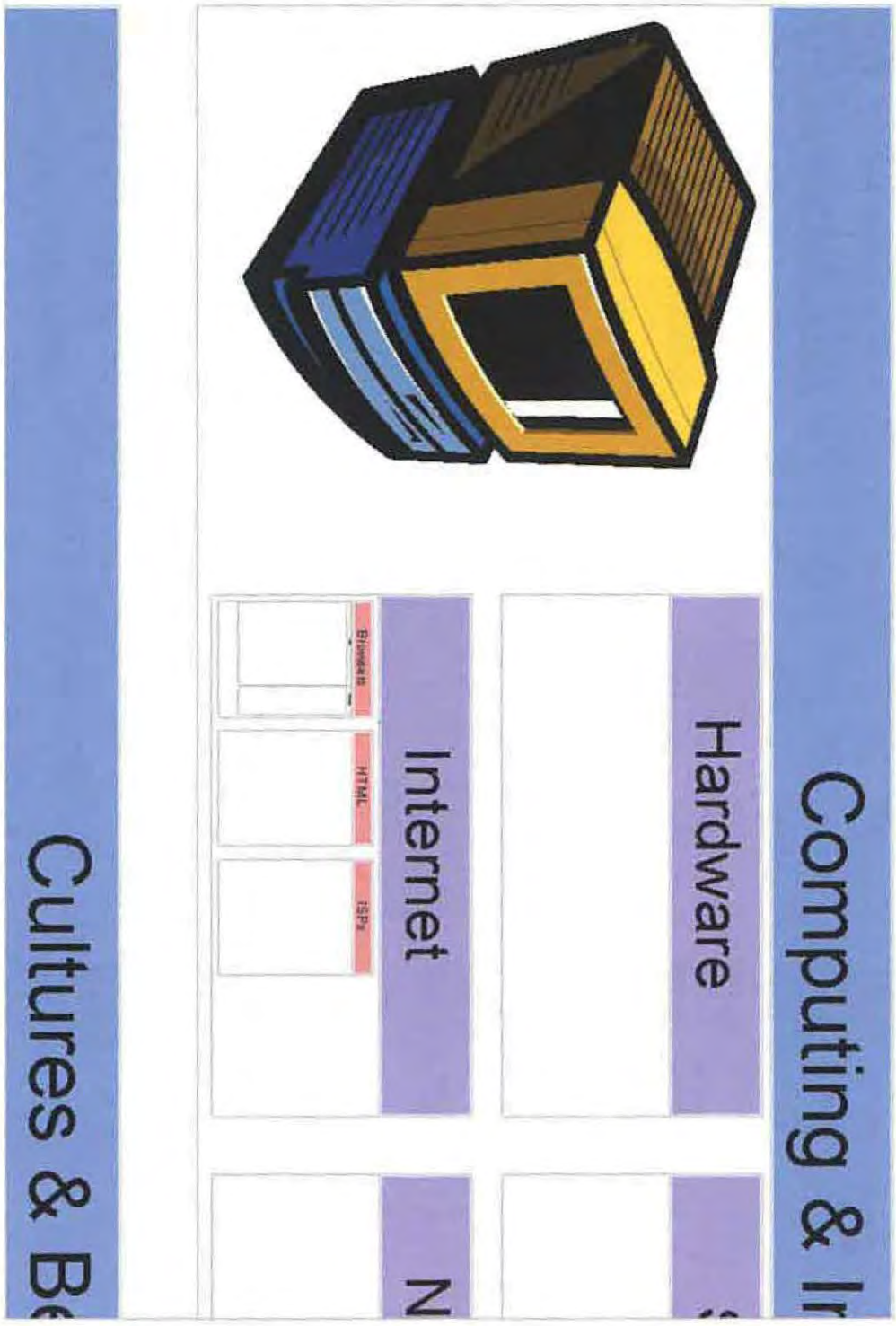


Figure 6.8. Zoomed in to one area of the World Wide Web.



Figure 6.9. Zoomed in on a list of browsers. Of course, ZoomWorld is a fine browser in its own right.

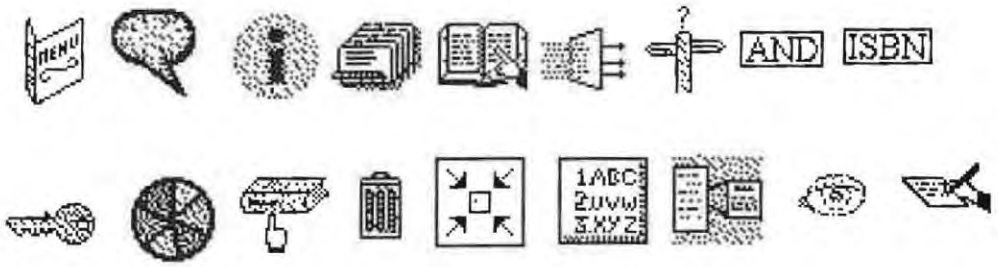


Figure 6.11. Bibliographic icons (from: http://www.scran.ac.uk/iconstd/). Many are inscrutable, as you can ascertain by looking at each and deciding what bibliographic category or operation it is supposed to represent.

A typical application for ZoomWorld arose in a design created for Apricus, which was trying to find a way to computerize a large (about a square meter) medical chart of the kind used in an intensive care unit (ICU). All of the methods tried proved slower than accessing the chart manually and also required excessive training; multiple displays could not be used to show the entire chart at once. Not only could ZoomWorld accommodate the chart, but also the same interface could serve as an enterprisewide database without significant extra development effort, allowing the company to extend its sights beyond automating the ICU database. The better interface allowed an expansion of the business model and the range of applications without requiring a more complex interface or any additional interface implementation, although a larger database was, of course, necessary. Figures 6.1–6.8 show versions of ZoomWorld that were developed at Apricus.[4] In testing, it was found that nurses could use the system after less than one minute of training.

In Figure 6.1, the ZIP shows an ICU unit, surrounded by other units. You can zoom in to see the kinds of data available for any of the rooms, some of which are unoccupied. In Figure 6.2, you have zoomed in to room 132, and the major charts can be seen.

You can continue to zoom in to a particular chart. When the text becomes large enough to be readable, it also becomes editable. In Figure 6.3, a horizontal time scale and a vertical set of legends have automatically appeared. These "float" over the background image so that as you move around in the chart, the scales stay in place relative to the screen. They automatically disappear if you zoom out to or beyond the point where the text is unreadable or if you zoom inside an entry in the chart.

For example, you might zoom in to a particular value and find useful data, such as normal ranges for the measurement or even a lengthy quote from a medical text. Note that such detail does not get in the way or take up any apparent display space, yet it is there, where it is needed.

The zooming can take place in the other direction. Climbing high above Figure 6.1, our first view of the hospital, you can see that the ICU unit is on Floor 1, and you can tell what other units are on the same floor. In addition, the physical layout of the floor is apparent (Figure 6.4).

---

4. My thanks to Apricus for permission to describe its version of ZoomWorld and for the use of some of the screens as illustrations. Many details and the implementation were provided by Drs. David Moshal and Emanuel Noik and their crew. The customization to the hospital setting is due primarily to Betti Newburn, R.N.

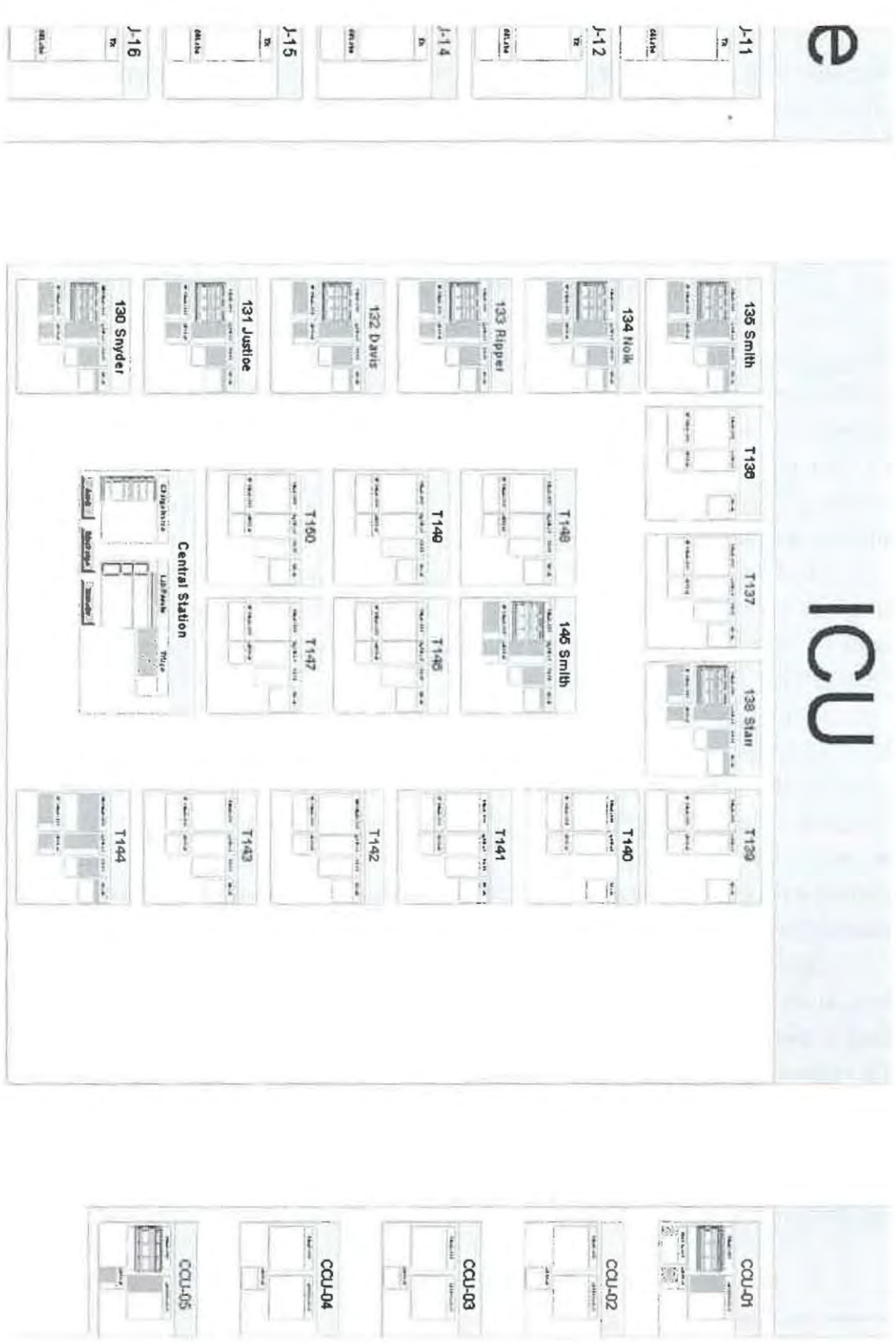


Figure 6.1. A ZIP view of an ICU. The numbered rectangles are rooms, and the names are those of the patients in them (no real data shown). (See color insert.)

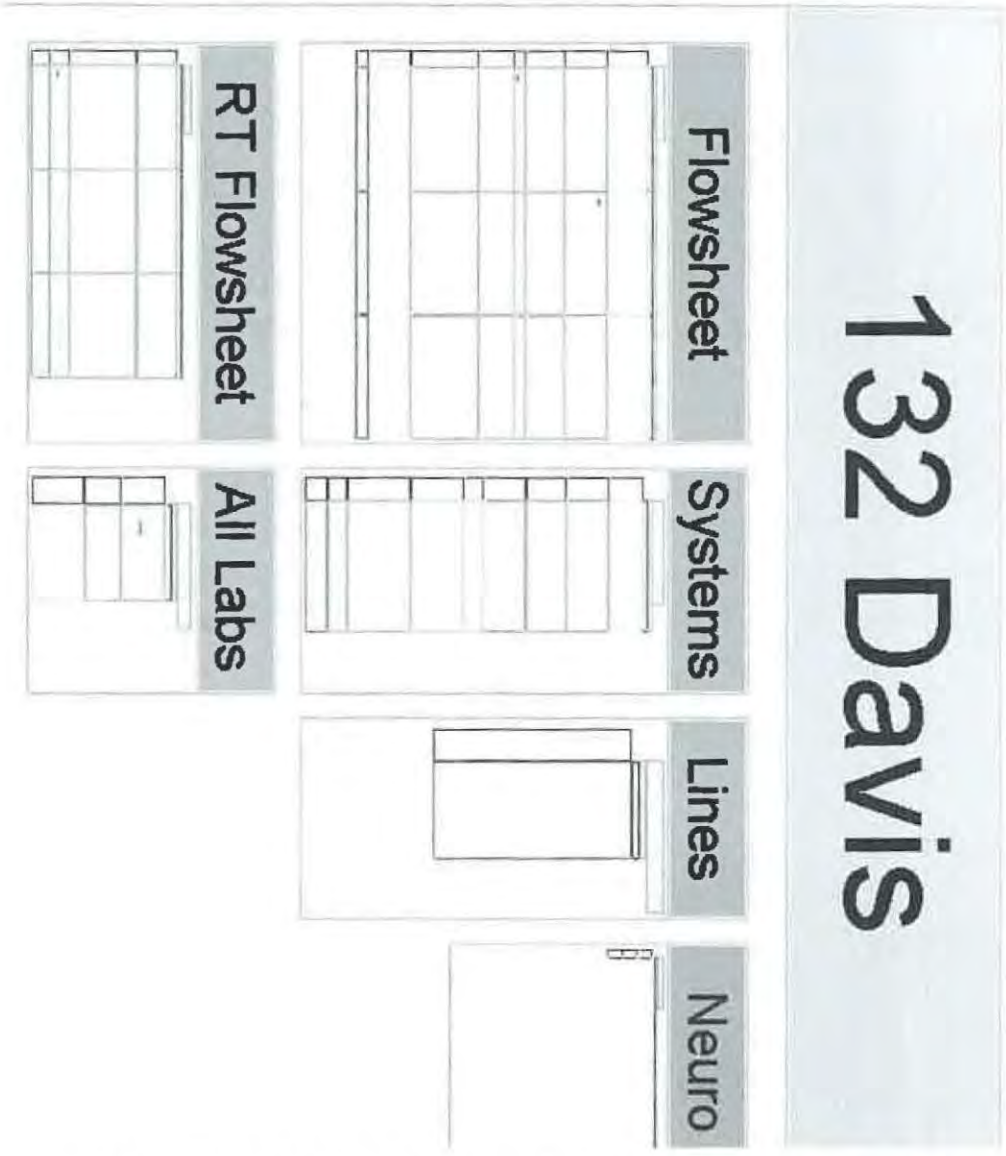


Figure 6.2. The charts for a particular patient can be seen when zoomed in this far. (See color insert.)

Fly still higher above the first-floor view, and the entire hospital becomes visible, with a one-story set of doctor's offices and a hospital tower of three floors (Figure 6.5).

Gradually climbing still higher, the entire chain of hospitals can be viewed (Figure 6.6), roughly in their geographic alignment. It is a matter of

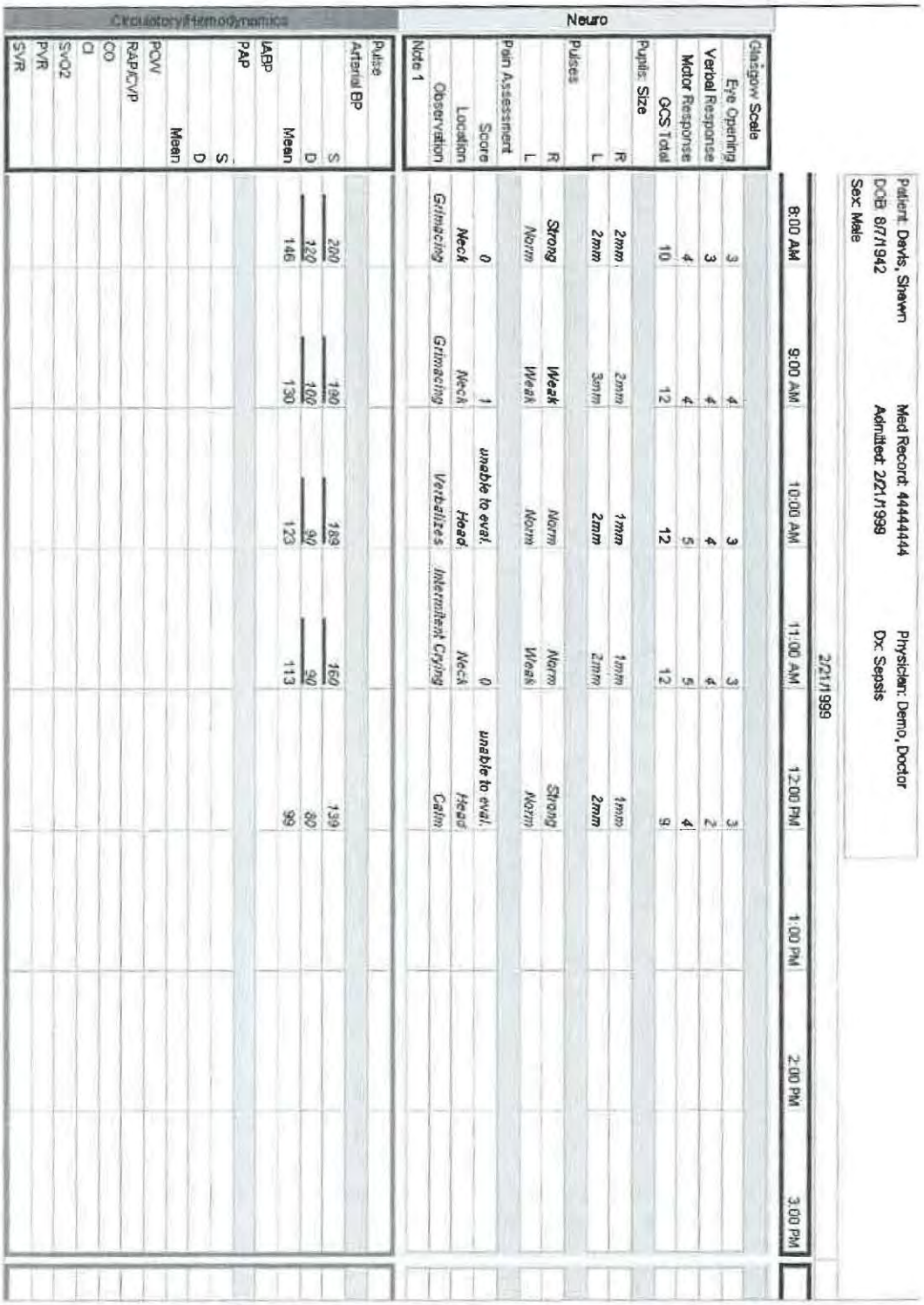


Figure 6.3. The chart's entries are now visible and can be operated on. Floating headings have appeared so that the columns and rows can be identified.

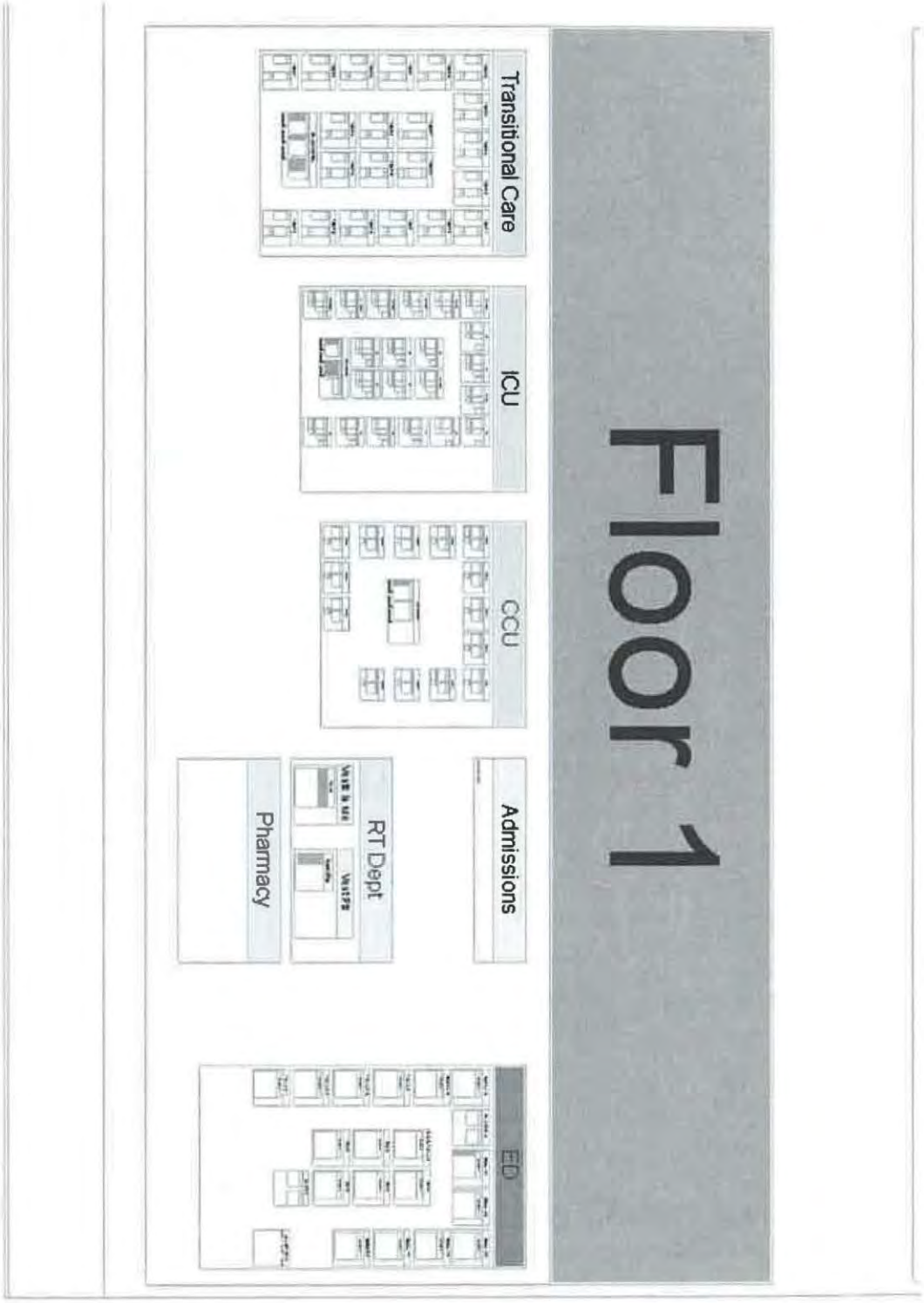


Figure 6.4. Zoomed out beyond the view shown in Figure 6.1, the entire first floor is visible. (See color insert.)

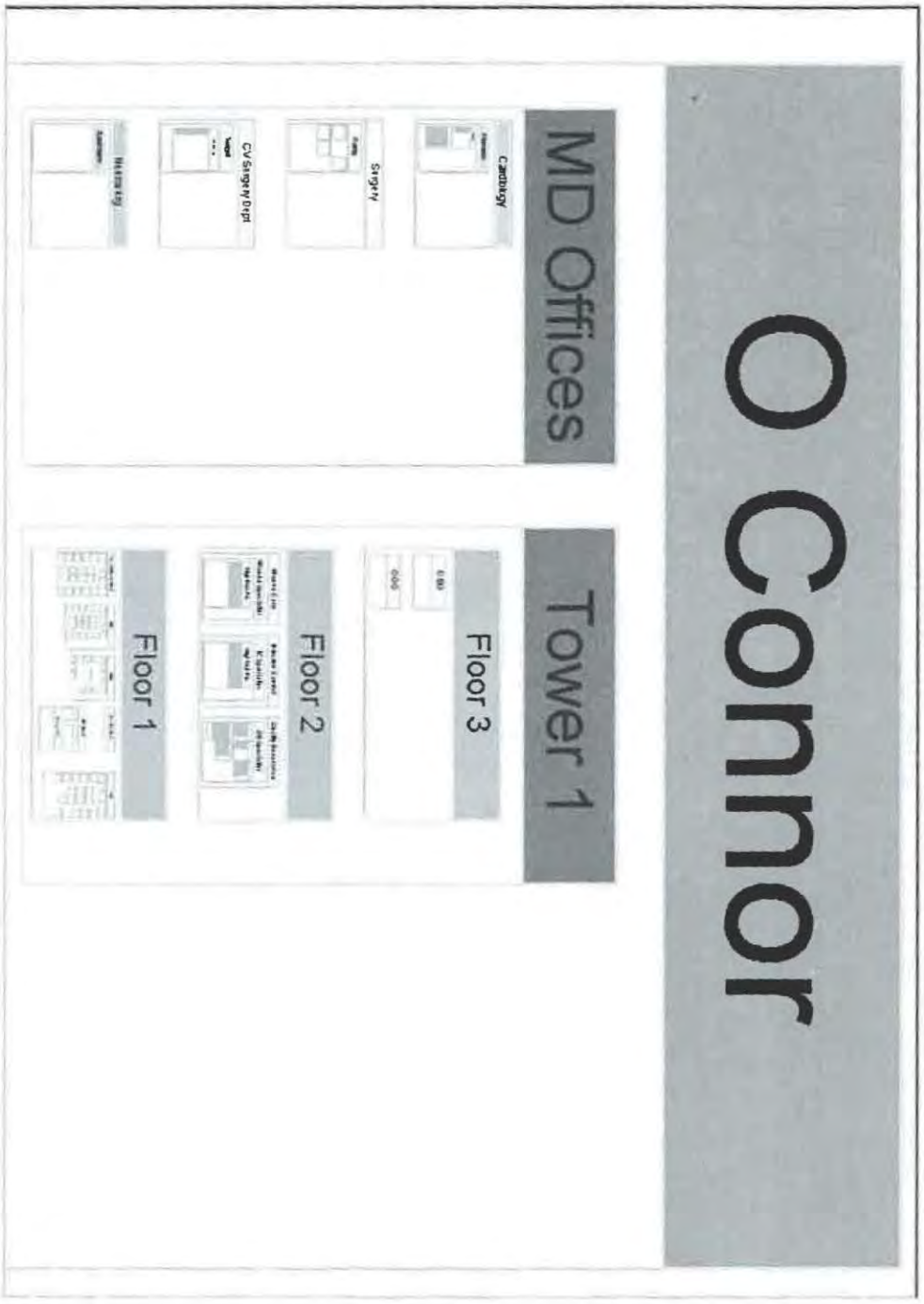


Figure 6.5. The entire O'Connor hospital can be seen. (See color insert.)

seconds to get to any patient's chart in any hospital, and the means to do it are evident with only a few seconds of instruction. You could operate this interface from what you've read here, and you would do it correctly.

It is not only patient charts that can be found from the ZoomWorld image of the hospital system. Assuming that you had logged on at the appro-

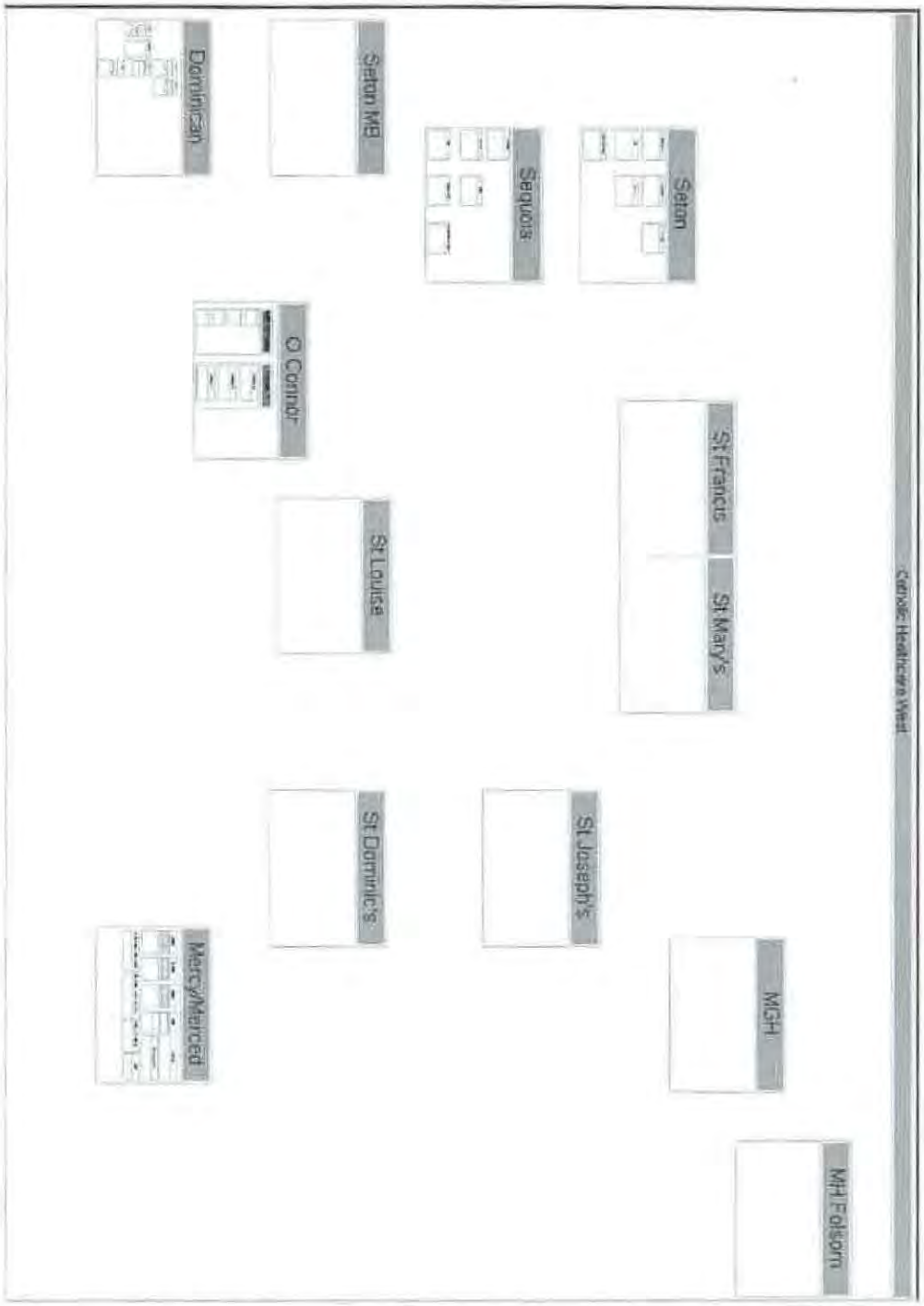


Figure 6.6. The entire enterprise can be accessed from this level. (See color insert.)

priate level of security, you could also zoom in on the comptroller's office, check on budgets, inventory records of medicines and supplies, and check human resources records or any other facet of the enterprise.

It is clear that ZoomWorld can serve as a database search interface for an enterprise or for a set of enterprises, governmental agencies, schools, and

scientific data sets, to name a few examples. Three-dimensional objects are represented by a two-dimensional projection or other transformation on displays, although a three-dimensional ZoomWorld can be created.

Although moves within ZoomWorld are usually accomplished by zooming out and in to the new location, a short move that preserves readability and context is sometimes needed. Therefore, interface designer Emanuel Noik's implementation of this system included a convention that delayed the beginning of zooming. If the graphical input device was moved during the brief delay—a few hundred milliseconds—instead of moving the zooming point, the graphical input device dragged the image. In testing, it worked well, although in general, the use of such delays can cause problems (see Section 6-4-5); in this case, an occasional zoom when a drag was intended and vice versa was observed. Faster and more certain is to drag the image while both GID buttons are held. Either mechanism is superior to conventional scroll bars, which, aside from the difficulties already discussed, may require the user to decide whether to move horizontally or vertically first and then to decide whether to use the arrows, to move the scroll box, or to click above or below the scroll box. Such decisions cause Hick's law delays (see Section 4-4). Scrolling itself is inherently slow. The user is limited by the system's speed of scrolling, which cannot be made too fast because the user will not be able to see what is flitting by the window; zooming can be fast because the visual images don't change but only scale, and you do not lose your sense of where you are. Note also that when you zoom in, that area is active to the extent that you can operate within an area. You do not have to click or otherwise signal the system that you want to use whatever facility you have zoomed in on. The conventional concepts of "opening" and "closing" documents or applications do not apply, either in ZoomWorld or in the complementary methods discussed in this book.

If we zoom out from Figure 6.6, the image of the hospital enterprise gets smaller and is surrounded by empty space, like an island. As this happens, other islands come into view, and when we zoom in one of them, we find that it is a portal to the World Wide Web (Figure 6.7). From that portal, we can zoom in to a specific area of the World Wide Web (Figure 6.8).

It is clear how the World Wide Web is surfed in ZoomWorld (see Figure 6.9). Another island is your local system, its mass storage, and any other information that can be reached without going over a network. When a volume, such as a DVD, is mounted, it appears adjacent to your local system island.

The consequence of this discussion should now be obvious: *The zooming interface paradigm can replace the browser, the desktop metaphor, and the tra-*



Figure 6.7. A ZoomWorld portal to the World Wide Web. (See color insert.)

ditional operating system. Applications per se disappear. Combined with the methods described earlier in this book, ZoomWorld can simplify the use of computer systems in general. With care and understanding of cognetic principles, it can be built so as to completely conform to the requirements of human cognitive abilities and would be easier to implement, learn, and use than present software methods.

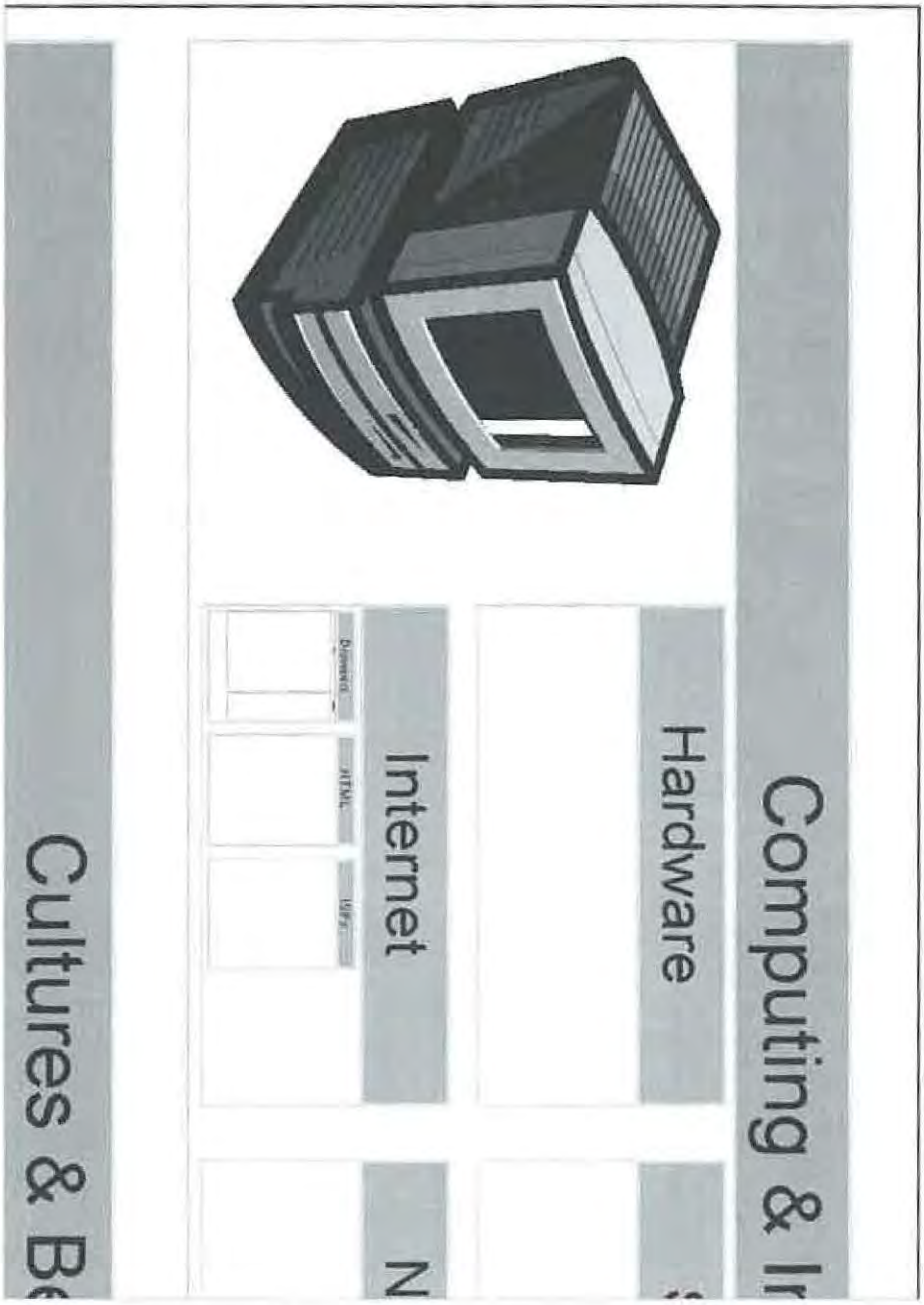


Figure 6.8. Zoomed in to one area of the World Wide Web. (See color insert.)

ZIPs may help alleviate another seldom-recognized interface desigı problem. We have acquired a fashionable verbal terseness in our interfaces trying to find the ideal one word that will make the menu item meaningful and have been forced into being cryptic in order to cram more labels on ou displays. Sometimes, a few words or a sentence would better serve the pur pose; in communicating, one should always use as few words as possible, bı



Figure 6.9. Zoomed in on a list of browsers. Of course, ZoomWorld is a fine browser in its own right. (See color insert.)

computer interfaces sometimes go beyond possible to the point of incomprehensibility. With a ZIP, screen real estate becomes more available, and we can more easily afford to maximize clarity rather than to minimize the use of pixels. Of course, even without a ZIP, *we should always prefer clarity over brevity in our labels.*

The current crop of interface-building tools cannot be used to build ZoomWorld. It must be programmed in a language such as Perl, C, C++, or Smalltalk; a reasonable mockup can be done with Macromedia Flash. If the

programmers have not studied the rationale for the change in paradigm, when ZoomWorld or LEAP is implemented, the programmers and designers working on the project tend to slip in details that work the old ways. Many times I've come in to review a client's progress only to discover that someone had built in a toggle or another modal feature or had used dialog boxes. This is one of the difficulties in maintaining the quality of these interfaces, namely, our old design habits. *The only solution to keeping an interface design on track is careful study of and attention to the fundamentals of interface design so as to take cognitive issues into account. That often means retraining the whole team,* from management to designers to testers, so that they understand the parts of cognetic technology pertinent to their roles. The training needs to include a study of design heuristics not discussed in this book but well known to the human factors community.

On the hardware side, a mouse for working with a ZIP might have three buttons: two marked Select and Activate on top and one marked Grab on the side, and which might zoom in when you twist it clockwise and zoom out when you twist it counterclockwise, as with standard screw thread direction, on the side.

6-3 Icons

Icons, those familiar little pictures used to identify buttons and other objects, are a shibboleth of modern interface design. Apple Computer, which is well known for its leadership in this field, advised us that "icons can contribute greatly to the clarity and attractiveness of an application. The use of icons also makes it much easier to translate programs into other languages. Wherever an explanation or label is needed, consider using an icon instead of text" (Apple Computer 1985, p. I-32). Later versions of the manual were not so dogmatic about using icons, but the damage had already been done.

Icons contribute to visual attractiveness of an interface and, under the appropriate circumstances, can contribute to clarity; however, the failings of icons have become clearer with time. For example, both the Mac and Windows 95 operating systems now provide aids to explain icons: When you point at the icon, a small text box appears that tells you what the icon stands for. The obvious reaction, which I have observed repeatedly when users first see this facility, is Why not just use the words in the first place? Why not indeed? *Instead of icons explaining, we have found that icons often require explanation.* If you wanted to obscure or to encode an idea to keep it from prying eyes, substituting icons for the words might not be a bad start. The problem with icons can be considered an issue of diminished visibility: The interface presents an icon, but the meaning of the icon is not visible, or it may give

the wrong message to someone for whom the graphic is unfamiliar or has a different interpretation. For example, an icon that shows the palm of an upraised hand indicates "halt" in the United States but signifies "here's excrement in your face" in Greece (Horton 1994, p. 245).

You do not have to be an interface expert to be bothered by icons. A car reviewer's complaint is typical, "You had to get out the manual to understand the [radio's] controls—there were no words on the buttons (you know, like 'volume'), only symbols" (Hotchkiss 1997, p. 14A).

Sometimes, the use of icons is rationalized on the grounds that they are language-independent, which is a useful property for software to have in an increasingly international marketplace, although, as noted, icons are not culturally independent. But even if an icon does not require translation, the linguistic aids and help screens used to explain it must be translated, nullifying any supposed advantage. With either icons or words, careful translation by an acculturated expert native speaker and writer of the language used by the target audience is required, as well as review by editors who have grown up and lived in the culture. Although words can be understood in at least one language, an icon is an equal-opportunity puzzle, potentially lacking comprehensibility across the entire spectrum of Earth's tongues.

In this example, the meaning of an icon was understood by only a minor subculture, to the detriment of nearly everybody else: On the Apple IIe, each connector on the back was identified by a distinctive icon. One especially enigmatic icon consisted of a horizontal line under which was a dashed three-segment line of the same total length (Figure 6.10). Of the thousand or so people I have asked, primarily at conferences, fewer than a dozen have known unequivocally its meaning and fewer still its derivation.

The icon represents an oscilloscope trace: An oscilloscope (Figure 3.5) is a device used by electronics technicians to render electrical parameters, such as voltage, visible. The dashed line in Figure 6.10 is the zero volt position, and the solid line is what you would see on the display of the oscilloscope if you put the DC voltage from the power supply across its inputs.

In the preface to *The Icon Book,* author William Horton says, "I've used systems with graphical user interfaces for a decade and would prefer to



Figure 6.10. A particularly obscure icon.

click on an understandable image than to enter technojargon commands—even if I could remember how to spell them" (Horton 1994). Of the two poor choices he presents, icons may be preferable, especially for the newcomer or the occasional user. However, he omits another choice: clicking on a button identified by a well-chosen word or two. To be fair to Mr. Horton, he states that in certain places, both words and icons are needed, and he correctly emphasizes the importance of testing, whether words or icons or both are used in an interface. Bob Horn (Jacobson 1999) has developed a style of combining the attributes of words and icons into combined symbols, which reinforces the dictum that *text is often the best visual cue*. We are expert at visually differentiating one word from another, and words can convey quite specific content. Ergonomic factors, such as case, font size, color, and other attributes of the text, are also important.

Mayhew (1992) cites a number of research studies on the use of icons. Unfortunately, most of the studies did not compare labels to icons. But from these and other studies, we can conclude that icons are most effective when there are at most a dozen of them and when at most a dozen are seen at one time. In addition, it is essential that they

- Are visually distinct
- Do a good job of representing the appropriate concept
- Are presented at a reasonably large size, typically larger than a text label would be

In every study that considered the question, icons were demonstrated to be more difficult to understand than were labels, especially at first viewing, which contradicts one of the most frequently cited reasons for using icons, namely, comprehensibility for beginners. GUIs often present us with windows full of identical icons, each with a label. The icons are small and numerous, and there are dozens of different icons. The limited conditions under which icons are effective do not obtain in present computer systems.

Although it is true that tiny icons can take less screen space than labels, you have to ask: At what cost? The smaller a button, the longer it takes to operate it, and the more difficult it is to find; also, it is difficult to make a small icon distinctive. Another small point: Icons take more time to create than do words.

A major problem with icons is that it is often difficult to figure what to call them, whether for communicating with others, when writing about them, or even if you want to think about them in a verbal way. How do you sort or classify them; where do they fit in an index? For example, one icon on Macintosh keyboards looks somewhat like a diagram of a freeway clover-

leaf interchange. In print, it has been called a cloverleaf, the propeller, or the pretzel. In this case, the key does have a name: In Apple manuals, it is called the Command key. This leads to the natural question as to why, on a keyboard that has explicitly labeled Shift, Option, and Return keys, did Apple not put the word *Command* on this key? What advantage does this icon confer on the user? How can you tell, by looking at it, that it is the Command key? The key with the little window on it on current Windows-specific keyboard falls into the same category of marketing triumphant over usability.

If we are to be consistent, pressing a key with a funny shape on it should simply type that funny shape into our document; keys with words perform other functions. (Arrow keys are probably a useful exception.) I suspect that the increase in sales due to such shenanigans is nil, whereas the difficulties that they cause—for example, requiring a new character in the fonts used for writing the manuals of every product that will require mentioning that key—are unending.

Trying to describe icons with words can lead to extreme verbal calisthenics. Consider the elaborate vocabulary that describes the heraldic symbols of English nobility. A shield might be emblazoned "Per pale or and vert a lion argent rampant."* Will we require such a language to describe icons when you phone up someone to get help in making a computer work properly?

There are, of course, many places where words fail. A color palette is such an example. One program on my computer offers the palette in words, presented in (mostly!) alphabetic order. Here is a portion of the list:

Cyan	Blue Violet
Blue	Brick Red
Magenda [sic]	Burnt Orange
Apricot	Burnt Sienna
Aquamarine	Cadet Blue
Bittersweet	Carnation Pink
Blue Green	Corn Flower
Blue Gray	Forest Green

* I believe the translation is: a shield divided into left and right halves, with the left half gold and the right half green. In the center of the green half is a silver image of a lion in heraldic style, set at about a 45 degree angle to the horizontal, with its head to the right and above.

Gold	Indian Red
Goldenrod	Lemon Yellow
Gray	Maize
Green Blue	Maroon
Green Yellow	

What color is "Bittersweet"? What if you are botanically challenged and don't know a dandelion from a corn flower? The index for a collection of clip art is often effective when it has thumbnail representations of each image, but the list of categories of images, or any higher-level organization, should be text. An image of a number of a flowers does not convey the same information as the word "flowers." The image might represent "summer," "a listing of florists," or "hayfever sufferers chat room."

On the desktop of my Mac, I have exactly one often-used icon outlined in red; changing the border color of an icon is a system option. The color does make that particular icon easier to find among the dozen or so icons that clutter my desktop. I do not have the option, which I would prefer, of having one red word. Horton (1994) points out that "color coding can fail if you use too many colors or if there are too many icons in each color." He specifies that if the icons are close together, you should use no more than seven colors; if they are scattered about, you should use no more than four colors. He is right, and his suggested limits apply to the use of color with words, as well. But too many GUI designers seem not to be reading Horton.

Apple's early guidelines state that "To use the screen to its best advantage, Macintosh applications use graphics copiously, even in places where other applications use text. As much as possible, all commands, features, and parameters of an application, and all the user's data, appear as graphic objects on the screen" (Apple Computer, Inc. 1985, p. I-31). However, the best advantage for the user is to achieve clarity and ease of use. The tendency to overuse graphics has been an impediment to good interface design.

In creating a telecommunications interface, its designers had to indicate the following states: dialing, ringing, busy, no dial tone, excessive noise on line, attempting to connect, and connected. After some months of trying icons and testing them, they found it impossible to design a set of icons that would invariably convey the information to users. When consulted about this problem, I asked: How do you draw the sound of a busy signal? I suggested that they use the list of words they had provided instead of icons; they

did, and testing confirmed the utility of the solution. That they did not see this for themselves is probably a result of the effective "brainwashing" of interface guidelines such as those quoted.

Using icons sometimes prevents designers from seeing the possibility of direct manipulation. To discard a document in most GUIs, you drag its icon to the wastebasket or trash can. If the document is visible at the time, why not drag the document itself? In a case like this, the icon is an unnecessary proxy and a level of abstraction you must learn and understand. We can do even better and avoid having a trash icon altogether: Why not simply select the document to be erased and use the Delete key? That would make deleting a document monotonous with other forms of deletion.

Icons seduce designers into looking for a representation, model, or analog for the task and then implementing an operation on the representation instead of the thing itself. On occasions such as the one just mentioned, this is a design error. Sometimes, icons seduce designers into creating yet more icons. A bibliographic database application designer asked me to comment on the quality of his team's choice of icons. I responded that most people who use bibliographies can read and that therefore words might be preferred. Figure 6.11 shows a few of the dozens of icons they had created. See whether you can decipher them (footnoted answers on next page).[5]

The clearest ones are probably Menu, AND, and, especially if you are professionally interested in books, ISBN. Having read the intended interpretation of each only two days before writing this paragraph, I find that I still cannot remember what they all mean, especially the one that looks to me like a beach ball.

To summarize: Surprisingly, icons violate the principle of visibility: It is their *meanings* that are not visible. Use icons only in the few situations

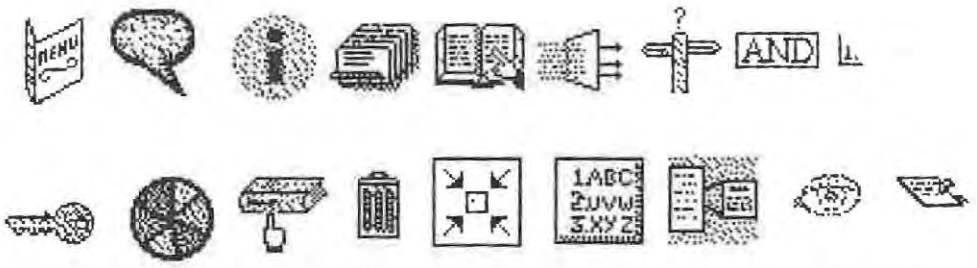


Figure 6.11. Bibliographic icons (from: http://www.scran.ac.uk/iconstd/). Many are inscrutable, as you can ascertain by looking at each and deciding what bibliographic category or operation it is supposed to represent.

where research has shown them to be advantageous. Otherwise, words are better.

6-4 Techniques and Help Facilities in Humane Interfaces

Be a driver, not a mechanic.

—Automobile advertisement

A desktop-based interface has a very low efficiency because you do not accomplish your tasks when you are in the desktop. A humane design that has neither desktop nor applications should leave the user involved with content at all times.

As we have seen, we can eliminate files and file names, leaving only one universe, or content space. You do not need to open or close documents; you zoom to them and just start working. You do not open applications; you duplicate a blank document (or whatever). You do not launch a game; you zoom in on it (a multiuser game may even be in progress). Separation of text into user-defined content areas uses the separators from the character set, user-chosen words or codes, or by making positional distinctions.

As the testing that led to SwyftWare and the Canon Cat demonstrated, neophytes find a system without a desktop or named files very easy to use. However, if experienced users of present systems are not given a few minutes of explanation, they can find these interface concepts so alien to their experience that they seem puzzling. An extreme example of this was when IBM reviewed the Canon Cat for possible interest in the interface. To avoid having to execute a nondisclosure agreement, IBM chose two ex-IBM employees who were experienced personal computer users to review the interface. Their self-chosen modus operandi was to see whether they could quickly learn the interface without manuals, online help screens, or tutoring.

At the end of the time, they reported failure. They had experimented with a range of command techniques, from the slashes of the IBM punch

---

5. *Answers:* display the main menu, change language, display information about this facility, browse, thesaurus, reduce the number of returned results, display information about the selected object's location (for example, the URL of a web page or a shelf location in a library), retrieve items in search for which both Boolean expressions are true, international standard book number, keyword, subject, title, clear current information, reduce view of current information, display a brief list of current search results, show all fields, show the results of the current search, author.

card–based Job Control Language—an ugly style of delimiter that, amazingly enough, persists to this day in web addresses—to IBM DOS commands, in their attempt to get the word processor started. They were, of course, typing and erasing text as they made these attempts. But they assumed that they were doing this at a "system level."

An interface, as far as is possible, should be self-teaching. This does not mean that an interface is, or can be, intuitive but that a user can readily find comprehensible explanations or instructions when she needs them.

To facilitate learning, a display of instructional text should be presented the first time the product is activated. A tutorial and the complete reference manual should be part of the interface, accessible at any time. Help displays are simply part of the content. No special mechanisms or techniques are required to use them. A LEAP to any keyword—and perhaps a few taps of a LEAP Again key—will suffice to find the desired information. Alternatively, you can zoom to a help area and then into the particular portion you need. Help systems in GUIs are an addition to the rest of the interface and have their own rules and implementation, which are an additional burden on the user and on the system's implementers. To where do you turn if you need help learning to use the help system in a conventional GUI: the help system's help system? As with lists of commands, which were removed from the realm of programmer-only creations and became a specie of ordinary text (in Section 5-4), the help feature of a humane interface can be operated without any special methods.

You're in Locked Text and Try to Type

You can move the cursor into locked text, say, by LEAPing there or just positioning the cursor directly. Now, if you try to type, what should the system do?

The more or less traditional computer method is to give you a beep and possibly flash the screen or the menu bar to indicate that you are doing something wrong and perhaps offer a dialog box with a message saying that you are doing an illegal operation. This is not a humane interface: It gives you an error message that you have to dismiss, and any characters you typed are lost, violating our parallel to Asimov's first law, that all user content is sacred.

It is also unacceptable for the system to move your input point to somewhere else in the document or universe. You have determined what you want to look at, and the machine shall not change it. A

number of options do not violate the cognitive principles of humane-interface designing. For example, the screen could split, one part showing the cursor in the locked text, with the material you are typing in the other part, say, just after the locked text. Another solution might be to provide a transparent overlay with the message that you are trying to modify locked text, that same overlay accepting the text you do type, and selecting it so that you can readily move it to wherever you wish. As soon as you move the text to where you wish it to be, the overlay disappears.

Transmitting e-mail is a separate application in most systems. In the system being discussed here, sending e-mail consists of selecting the text, selecting the address, and executing a Send command. Because nearly every operation begins by making a selection and ends by executing a command, the standard sequence of operations becomes habitual, and the operation of sending e-mail does not feel like three distinct steps but only one.

Your e-mail directory is simply your address list or any list containing names and e-mail addresses. It is just ordinary text, not a special mechanism. Say that you want to send an e-mail to your Uncle Herman. You type the note and select it. You then LEAP to *Herman Jackson* or however you have him listed and then select his e-mail address. Lastly, you type or select from a menu the Send command and execute it as previously described, with the Command key. Instead of a single name, of course, you could have selected a list of names for an e-mail broadcast.

Another approach is to have an e-mail command use the first line of the selection as an address, the second line as the subject heading, and the remainder as the content of the message. A command would be used for making a selection an attachment to an e-mail. No doubt various vendors would supply different methods for sending e-mail. A wise user would buy only one.

Here is a seemingly peculiar method of receiving e-mail that, in practice, works much better than it sounds. When an e-mail arrives, two document characters, which sandwich the growing contents of the e-mail, are inserted immediately before your current cursor location. You can, of course, continue to work as the e-mail streams in. No special window or "You have mail" message is needed. You can ignore the e-mail or read it in progress. At any time, during transmission or afterward, you can select the e-mail and then move it to wherever you wish. An experienced user is likely to have a spot labeled "E-Mail Repository," "E-Mail Goes Here," or some such

so that it can be LEAPed to and the e-mail (deposited there automatically as well as being presented at the focus as it arrived) can be read at leisure. The value is that the e-mail, with any attachments, comes in automatically and visibly and becomes part of your text; you already know how to move text to where you want it. Attachments download in the background so that your work is not interrupted and that, if there is fear of software viruses, can be placed into a nonexecutable software quarantine. A command that gathers recently received e-mail and puts it into a particular area in your universe is a feature that software vendors might supply.

Encryption is a matter of selecting some text and applying the encryption command. Encrypted text can be left in place, e-mailed, or otherwise used. Decryption is similar.

Any abilities now provided by applications also fit into the same mental model. Consider spreadsheets: If a selection has the syntax of an algebraic expression and if a Calc command is executed, the expression is stored, and the results of the expression are displayed in its place. Another command reverses the process and shows the stored expression, which, if necessary, can be edited and recalculated. If expressions are permitted to have variables, if values can be given names, if a syntax that allows position-relative references is provided, if there is some notation for ranges, and if the expressions are put into a table, a fully functional spreadsheet has been created. But it is more than just a spreadsheet, because references to the spreadsheet values can be incorporated anywhere in text, and any value in the text can be incorporated, by name, in the spreadsheet. That is, the results from a spreadsheet in a report can be used in the introduction to the report, and values in the text can be used in the spreadsheet—without having to set up any special mechanisms. There are no sharp demarcations among text, mail, and spreadsheets or, for that matter, any other content.

6-4-1 Cut and Paste

Another problem with conventional interfaces is the cut-and-paste paradigm. Most users of cut and paste have experienced the loss of some work when they inadvertently performed a second cut before pasting the first. When text is deleted, it should not disappear into limbo, especially not into an invisible cut buffer. One solution is to have the cut text appear as the last item in a document that collects deletions. The deletion document is just an ordinary document and can be treated as ordinary text. (It may be useful to have a special command that deletes in a more permanent way.) The essential point here is that nothing mysterious happens to deleted text, and

no special command or knowledge is needed to find the document containing deleted text, beyond having to know that the plaintext words, such as "This document contains deleted text," are in it. The user could type in any phrase to use as a target for this document. Of course, any phrase in the recently deleted text can also serve to find it, just as if it hadn't been deleted.

Any humane design for deletion

- *Has deletion operating no differently from other commands*
- *Puts nothing at risk when text is deleted or moved*
- *Creates no special buffer or other "system-level" or hidden place to where text is moved*
- *Treats single-character deletions no differently from multiple-character deletions*
- *Can be undone and redone*

6-4-2 Messages to the User

Always do right. This will gratify some people, and astonish the rest.

—*Mark Twain*

Whenever you find yourself specifying an error message, please stop; then redesign the interface so that the condition that generated the error message does not arise. In other words, an error message signals an error, to be sure, but the error is usually in the design of the system or its interface, not on the part of the user. On occasion, the work my associates and I have done in trying to eliminate an error message has resulted in our realizing that fundamental design decisions were incorrect and that design revisions were in order. In this regard, and in this regard only, error messages are a great boon. For example, in designing a package to do arithmetic, there seemed, at first, to be no way to avoid giving an error message when the user tried to divide by 0, but a better solution was the creation of a value called *undefined*. The IEEE (Institute for Electrical and Electronic Engineers) 754 standard for arithmetic uses NaN, or "not a number," for this. Arithmetic on *undefined* is well defined (*undefined* + 3 = *undefined*, for example), and results involving it are more useful and diagnostic than simply stopping the calculation. Using *undefined* also solves the problem of what to do when the Calculate command is applied to an object that does not have the syntax of an arithmetic expression. The information that something has gone wrong appears as the user's locus of attention, namely, as the result desired. More informative is to

replace the simple *undefined* by divide-by-0 and other messages as appropriate. Arithmetically, they all behave as does *undefined*.

Another example of where the desire to eliminate an error message affected a hardware decision arose in the design of the Macintosh; it is also an example of a "bet the farm" kind of decision that often arises when you are creating a new product. At the time, we were choosing the storage medium for the Macintosh. Hard drives were too expensive to be considered. The 5¼-inch floppy was in nearly universal use, and a number of technologies were competing to replace it. The Macintosh team decided on the 3½ inch floppy, which turned out to be the right decision, as the rest of the personal computing world went the same way. Had IBM, for example, chosen otherwise, users might have found it difficult to buy diskettes for the Mac.

But the choice of the Sony drive was, for us, sealed by a humane interface consideration: Most of the brands of drives we examined allowed you to eject a disk by pressing a button on the disk drive. Unfortunately, nothing prevented you from doing this before your current work had been saved on the disk. In that case, we would need a warning message to let users know that they had made an error: "You have removed the diskette that this document came from. To save your work back onto the diskette, please replace the diskette in the drive." Of course, if the user had already moved away from the computer, taking the disk, the next user would be stuck. Then I learned of a drive that did not have an Eject button but that ejected the disk only on a command from the computer. Now, when you wanted to eject the disk, you signaled the computer, and the computer had time to check that your command was appropriate before ejecting the disk. If your command would lose some data or would otherwise cause a problem, the system could correct the situation before ejecting the disk. We chose the self-ejecting disk drives, which, during the reign of the floppy, was one of the many factors that made the Mac easier to use than its competition.

If a message must be given and no user response is required, the message can usually be eliminated. If, for some reason, the message absolutely must appear, it can be presented as an overlaying transparency, as discussed in Section 5-2-3. With a transparent layer for messages, the underlying screen can be seen and used as if the transparency were not there. In other words, no special action need be taken to dismiss the message. You just work through the message, and the message disappears when you perform any operation on the underlying layer; moving the cursor is not an operation in this sense. Unlike a standard dialog box, no extra user actions are required to dismiss a message, and a message never totally obscures any information on the display, a problem common in today's interfaces.

The claim is sometimes made that responding to a message is necessary to meet a legal standard. This presumption could, I believe, be challenged in court by pointing out that a user closes such message boxes habitually and that closing the message box does not imply that the user has read, or even consciously seen, the message (see Section 2-3-2).

Messages: A Case Study

One of the more remarkable designs that came from eliminating a set of error messages was the set of methods developed by Dr. James Winter for storing and retrieving information from mass storage on Canon Cat. Users of GUIs are familiar with the many messages associated with storing and retrieving information; for example, you are warned if you try to close a file that has not been saved. Originally, I had proposed that we have a pair of commands, each activated by a dedicated button, for saving and retrieving the user's universe; there was, as I have discussed, no file structure.[6]

Dr. Winter showed that only one command was needed and that this interface would be more secure. My first reaction was that it could not possibly work, a common response to good, radical ideas. After all, what could be simpler than my two commands? Dr. Winter's method did all that he had claimed, and it was implemented in the commercial product, where it proved successful and popular. In particular, many errors made in saving and loading information, which can so often devastate a user's data, could not occur.

To follow his idea requires a bit of background. The Cat had already dispensed with the concept of files in the usual sense, so that all work was stored in a user's workspace. The removable storage—in those days, a floppy disk—contained the same amount of information as memory, so the entire workspace was kept whole whether in memory or on disk. One then had the mental model of swapping entire workspaces, a concept users found very comfortable. Once a workspace was swapped in, the computer kept track of whether it had been modified, and every disk was given a probably unique serial number derived from a checksum on its contents.

---

6. Ideally, there would be no commands for saving your work whatsoever. Your material, including all intermediate states of its development, should be automatically preserved, unless you deliberately permanently deleted any or all of it. Our hardware resources at the time did not allow such an implementation.

Winter's idea was this: Only one command, which he called DISK, was needed. It would consider the state of affairs and automatically do the right thing. To demonstrate that this would work, he created a simple chart that showed the action the system would take when the DISK command was activated under various conditions. Here is the central portion of that chart.

State of memory:	Unchanged	Changed	Empty
State of disk:			
Same	no action	save	no action
Different	load	warn	load
Blank	save (dup)	save	no action

If you have placed a workspace from disk into memory and not changed it, the memory state was "unchanged." If that same disk were in the machine and you pressed DISK, there is no need to do anything; it was suggested that this might trigger a verification of the memory contents against the disk. However, if the disk had been removed and replaced with another disk ("different"), because there was a copy of the workspace—on the disk from which it had been taken—so that the system knows that no information would be lost, it erased memory and loaded the workspace from the disk. In the third case, there might be a blank, unused diskette, in which case the user probably wants to make a copy of the workspace, and that's what the system does; diskettes did not have to be formatted, as this was done "on the fly," as necessary. If the system has guessed wrong in any of these cases, no harm has been done.

The rest of the chart is self-explanatory. The term "warn" means that a message was made available that told the user that because the workspace had been changed and the disk in the drive had also been changed, the system could not do anything without losing information. It recommended that if the user wanted to retain the changes, he or she return to the original disk or save the workspace to a blank disk. After that was done, the DISK command would load the new workspace. There was a way to force a load. The system also automatically performed a DISK command if the system were left unused with the workspace in the changed state for a few minutes, adding further safety to the system.

Winter had originally proposed an even easier method, based on making use of a disk drive that could detect when a disk was inserted and eject it under program control. Unfortunately, these

drives were made by Sony, and we could not convince Canon, a competitor, to use them: an example of making users suffer in the name of corporate pride. Given the Sony hardware, we would have needed no disk-related commands at all, just an Eject button on the drive that would have been read by the system and the disk ejected or not, depending on the chart. At Apple, which did not make disk drives, we were able to specify the Sony drive; however, I did not have the benefit of Dr. Winter's insight at the time I was writing the specifications for the Macintosh project.

In classroom testing, the DISK command was often praised by educators. Aside from requiring almost no class time to learn, and no time was wasted formatting diskettes, it prevented one of the most frequent excuses for lost work: If a first student left without saving her or his work and a second student put a disk in and tapped the DISK command, the warning would appear. The second student would then either rush to find the first student or would save the material on a blank disk for the first student, because the second student's disk would not load until that had been done!

Winter's DISK command was easy to use. The instructions reduced to: Whenever you want to do something with the disk, tap this button. It also made every attempt to preserve the users' work. The one-disk-equals-one-workspace concept also vastly simplified the user's mental model; the CAT became a "window" that apparently looked at the information on the diskette.

It has been suggested that the DISK command created a modal situation. The objection, however, occurs only if the DISK command is thought of as Load and Save commands hidden behind the same key. But if you don't know about those commands, as on conventional systems, you may just think of the DISK button as the "do-what-needs-to-be-done-with-the disk" command. In the event, no mode errors were observed in testing.

If your computer still uses floppy disks, you are probably aware that you must either "format" a floppy before you can use it or purchase "preformatted" floppy disks. On the Canon Cat, disk formatting was done as data was stored, so that it seemed to take no time and the user did not have to be aware that the process existed. Because the user gets no utility from being able to control the formatting process, there is no reason for the user to know that it exists. This is another

example of the general principle: *If a control must always (or never) be operated, don't provide it.*

6-4-3 Simplified Sign-Ons

Users are doing more work than necessary when signing on to most systems. You first state who you are—your "handle," "online name," or "system name"—and then you provide a password. The name presumably tells the system who you are, and the password prevents unauthorized persons from using your account.

In fact, you are telling the system who you are twice. All that is logically required is that you type in a password. There is no loss of system security: The probability of guessing someone's name and password depends on how the password was chosen, its length, and the like. Finding the user's online name is usually trivial; in fact, it is commonly made public so that she can be communicated with. A badly chosen password, such as your dog's name, is the most common reason for poor security.

The technical argument that typing two separate strings of characters gives more security is false. If the online name is j characters and the password is k characters, the user, to sign on, must type $j + k$ characters, of which only k characters are unknown to a potential interloper. If the password was chosen randomly—this is the best you can do—from a character set with q characters, the probability of breaking into the account on a single guess is $1 / q^k$.

Requiring a password of even one additional character and eliminating the requirement that a user supply a name increases the difficulty of guessing the password by a factor of q and saves the user from having to type $j - 1$ characters and from having to wait for, recognize, and respond to two prompts, or fill in two fields, instead of one. We get greater security, use less screen real estate, *and* gain greater ease of use by increasing the minimum password length by one character and eliminating the name field. We lose nothing by just dropping the name. Less bothersome security techniques, such as voiceprinting or using fingerprints or another unmodifiable physical characteristic of a user, would be better yet for some applications, although you could not tell a trusted associate how to sign on to your account unless there was also an alternative way to pass the security check.

The question arises: How can you ensure that everybody has a unique password in a password-only system? What if two or more users choose the same password? One option is to have the system assign them. Badly implemented, the passwords are unmemorable, such as *2534-788834-003PR7* or *ty6*>ffˋd%d.*

There are many ways of creating memorable passwords, and in such quantity that you can give the user a choice of five or six. For example, you can have the computer choose two adjectives and a noun at random from a large dictionary, and present a list such as

- savory manlike oracle
- exclusive malformed seal
- old free papaya
- blooming small labyrinth
- rotten turnip sob story

from which the user can choose her favorite. In English, there are at least two trillion such combinations. With a million tries a day, it would take an average of over 25 years to guess such a password. That's reasonably secure.

When the idea of improving the interface to a web site or a computer system by simplifying the sign-on process to require only a password is suggested, it is usually rejected on one of two grounds. Either the programmers say that that's just not the way it's done, or they say that they have no control over the sign-on procedure. But someone, of course, does have that control.

6-4-4 Time Delays and Keyboard Tricks

We are likely to have conventional alphanumeric keyboards attached to our computers for a while. In spite of many attempts at reform, such as the Dvorak arrangement of keys, the inertia of the mass of millions of people trained to touch-type on the QWERTY keyboard has proved impossible to overcome. All we can do as interface designers is to peck at the edges and make small improvements that do not require tedious retraining. Here are some small improvements that we might get away with.

Initiating automatic repeat on most keyboards requires that you hold a key for 500 msec, after which the repetition begins. This is an example of a fixed delay; however, there are good reasons to avoid using fixed timed delays in interface design. Any choice of delay is likely to be both too long and too short, depending on the user and the circumstance. In this case 500 msec is too short when you linger while holding down a key, perhaps because you are thinking about what you want to type next. In this case, you may awake from your reverie to discover a few lines of sssssssssssssssss on your page. (My cat is adept at eliciting this behavior from my computer.) A slow typist or a person who suffers from any of a number of neurological or physiological problems may also find the 500 msec autorepeat delay too short.

Nevertheless, 500 msec is also too long. For one thing, delays are just that: delays. The user has to wait for an effect to take place. An example that

users find particularly annoying occurs in the Macintosh interface: To change a file name after you open a volume or a folder, you click on the name and then wait for about half a second for a special border or color to appear, indicating that it is now editable. The reason for doing this was to allow you to select a file name with a simple click, without risking accidentally editing it. Once the file name changes to the preeditable state, you must click on the name again to put the system in the editing state. Evidence that the delay annoys users comes both from interviews and from the number of times that magazines mention tricks for getting around the delay. Users do not like to be forced to wait.

John Bumgarner, while working at Information Appliance, came up with an elegant solution to the autorepeat problem. He began with the observation that in most phonetic languages, the same character is almost never typed three times in a row. He also observed that autorepeat is rarely used unless more than five instances of the repeated character are required; otherwise, the user simply types the character the desired number of times. His autorepeat software started to autorepeat if a key was held down for more than 100 msec after the third consecutive instance of a key being typed. In other words, to get a line of equal signs, you would type

= = =↓

You would then wait until the desired length of equal signs was produced before releasing the Equal-Sign key.

Pressing the same key repeatedly is faster than typing different letters, and a GOMS analysis shows that the expected time for starting autorepeat drops from the conventional method's 700 msec to 400 msec. Bumgarner's autorepeat proved easy to learn and, in our testing, was never activated accidentally. (It won't autorepeat even if your cat sits on the keyboard.) One negative that it shares with the standard autorepeat method: It is an invisible function, labeled nowhere on the computer.

Well-designed computers and information appliances have **chord keyboards**, so that the software can recognize multiple keys held down simultaneously. Older and more primitive computers had keyboards that could recognize only a few special keys, such as Shift, that could be recognized as being pressed at the same time as were other keys. Chord keyboards allow us to solve a number of otherwise difficult interface problems. For example, how to do an overstrike: A logical method of creating two symbols that appear in the same character location is needed. For example, if you wanted to create a dollar sign by overstriking an *s* with a vertical bar (|), you should be able to use a temporal "overstrike" on the keyboard to represent the physical overstrike:

This would not conflict with the overlapping keystrokes that occur in normal high-speed typing, whereby the key pressed first is released only after one or more other keys have been struck. The word *the* is often typed not as

t↓ *t*↑ *h*↓ *h*↑ *e*↓ *e*↑

but, to give one of many possible sequences, as

t↓ *h*↓ *e*↓ *t*↑ *h*↑ *e*↑

Modern keyboards and their enabling software can accommodate these overlapped keystrokes, a phenomenon called **rollover**. Most keyboards have ***n*-key rollover**, which means that the system will not get confused if as many as *n* keys are held down at once while typing continues. Given human anatomy, there is little need for *n* to be greater than 10, although there is no technical reason for *n* to be limited at all if the computer has a chord keyboard.

Given the convention of creating overstrikes by holding one key while tapping another, accents and diacritical marks could be treated as overstrikes and handled in a uniform way. For example, *é*, as in the name Dupré, is produced on the Macintosh with the unguessable key sequence

Option↓ *e*↓↑↑ *e*↓ *e*↑

Note that this is an example of a modal, verb-noun method, a violation of Apple's own guidelines. It also operates inconsistently. If you type the following sequence, you get a quote followed by a *t*, not a *t* with an acute accent, as you might expect:*

Option↓ *t*↓↑↑ *t*↓ *t*↑

Typing accents and diacritical marks becomes simpler and more logical if overstriking is accomplished with a quasimode:

e↓ '↓'↑ *e*↑

You hold down *e* and, while holding it down, tap the accent. You could also type *é* by overstriking in the reverse order:

'↓ *e*↓ *e*↑'↑

It makes no difference, logically, in which order you perform the operations.

Overstriking is also useful in creating mathematical and other special symbols and for some computer languages, such as APL. You might argue that rather than accommodate overstriking, we should just add whatever characters

* If you guessed Option↓e↓↑↑ t↓ t↑ to get an accented t, you'd still be wrong.

we need to the character set; after all, we have fully bitmapped displays. True, but not all of us want to take time or have the skills to design and to install a new character and add it to every font in which we want the new character to appear. Also, it seems absurd that on a modern computer we cannot accomplish what we used to be able to do with the lowly mechanical typewriter!

Overstriking need not be limited to two characters; any number can be overlaid, such as

Shift↓ *s*↓ *Shift*↑ |↓ /↓ /↑ |↑ *s*↑

That sequence will produce a dollar sign with a slash through it. Considerations of esthetics and readability, rather than hardware or software concerns, should form the only limits to overstriking.

To give immediate feedback to the user as each key is typed, assuming that *n*-key rollover and this book's overstriking technique are both operating, the interface may have to temporarily display a pair of overstruck characters as adjacent characters. The reason is that it cannot tell the difference between overstriking and rollover until the keys are released, at which time overstruck characters would coalesce automatically. I will mention here an essential keyboard reform that I touched on earlier—the elimination of the Caps-Lock key. It introduces a mode.

6-5 Letter from a User

When I was working on a project for a large company, an experienced user of that company's software wrote a letter that illustrates a few of the points made in this book. The quoted statements are from the letter.

- "The software was represented to me as a more mature product." Interviewing the programmers revealed that schedule was put ahead of quality in the priorities. What had been offered to the customers was the dream of the original project leaders. What was delivered was a schedule-driven "minimal usable subset." Many desirable details had to be omitted because the tools, which had been chosen prior to the completion of the interface design, could not implement the desired interactions.
- "The user is required to press Enter or click the mouse button many more times than is necessary to enter information." When entering material into a field, it is *not* necessary to have the user press Return or Enter or do anything else at all. *When the user moves to the next field or screen or uses a menu or a button, the system should just accept the input as it stands.*

Using Tab instead of arrow keys to move between fields caused problems. Two of the fields on one screen had free-form text input. In those fields, a user could tab to indent paragraphs or make indented lists. Therefore, Tab would not get the user to the next field. It was painful to watch a user repeatedly tap the Tab key to try to get to the next field.[7]

These examples represent two common interface problems. The first is the use of Return to delimit a field, a custom that dates back to the limitations of the teletype machine as used in time-sharing and minicomputer applications many decades ago. The second is the overloading of Return and Tab, using them to mean one thing in free-form text fields and another in shorter fields.

- "When a search option is selected, the cursor should appear in the appropriate text box so the user can start entering information without having to click the box with the mouse or press the Tab button." This is a specific case of a general principle: *If the user can do only one thing next, have the computer do it.*

- "Useless dialog boxes are probably the major cause for wasted user input and frustration." The dialog boxes he showed were the kind that inform the user that something has happened and require a mouse click or a press of the Enter key to get rid of them. There is no choice; you must click the box to go on. This is another specific case of the general principle just mentioned: If the user can do only one thing next, do it for him. As the writer said elsewhere in his letter, "It is important that every time a user must interact with a dialog box that there is a productive outcome to the interaction." This can be generalized to: *Every time a user must interact with a computer, there must be a productive outcome to the interaction. Moving forward in the work flow in and of itself is not a productive outcome.*

The writer went on to complain that another dialog box merely "tells the user that the item is already listed" when the name or number of an existing item was entered on the screen that allowed you to enter a new item. You had to dismiss the dialog box before you could go on. He suggested instead that three buttons should appear, allowing you to leave the item as is, to delete the listed item, or to bring you to a screen where you could edit it. His design is better than what was provided, but we can do better still. Part of this problem is the idea that entering a new item is different from editing

7. A Next Field key would seem to be a useful addition to computer keyboards.

or deleting it. Here is a simpler method: The user summons the form and puts in the item descriptor; if it is new, the item is entered, and the user continues as she had expected to continue. If it is an item already on the list, that item's data is brought up so that the user can instantly see that it already exists. She can then edit it directly. Deletion, of course, is just one of the ways you can edit.

- The writer pointed out that a screen soon fills up with identical icons differentiated only by the names that appear underneath each of them. He suggested that a greater variety of icons be used because the "environment is a *visual* one." He's right in that the icons in a screen full of identical icons just wastes space. He suggested that four different icons be used. But with only four, there will still be lots of identical icons lying around. The solution is to realize that the icons are unnecessary. In making graphics-based interfaces, we must remember that *text is also a visual cue;* it is a very powerful one that is full of specific content and one that we all know very well (see Section 6-3).

- "If you have a purchase order open and you want to do an item inquiry, you receive the following dialog box:" The box says: "This application may not be used when 'Create / Update P.O.' is running." The correct user reaction is "Why not?" Here, the designers were not aware of the real work flow of the user's task.

The general principle is that almost any overly structured work flow approach to system design risks impeding a user whose task or inspiration demands a different approach. In this case, the interface moves from aiding the user to being a dictator. *A computer should be a servant, not a peer or a boss.*

- "There is a tendency in the computer industry to conform whether or not it has a productive outcome. . . . Conforming and having a standard design are very important . . . because it takes the user less time to get up to speed . . . , but if by conforming or standardizing, you are creating uselessness, then you have failed in your design." This is exactly one of the points made a few years back by Grudin's well-known article "The Case Against User Interface Consistency" (Grudin 1989). Obviously, Grudin's analysis has not permeated into the industry. *Abandon a standard when it is demonstrably harmful to productivity or user satisfaction.*

- "When the software was designed, it used the standard Windows menuing style." His example showed: "All the menus have uselessness built into them. One menu has the Exit command hidden below the File menu." He was trying to point out that Exit was the

only item in that menu. "Exit should sit on the main menu line and the choice File should be removed." As he says, "one item does not make a list." *It is wasteful to have to open a menu when there is no further choice.*

• The writer often made specific suggestions to improve a detail when there was a deeper design flaw that needed correction. For example, in making up a purchase order, the user is first given a screen, called Purchase Order Direct Entry (Add), for a particular product. The user must then choose a quantity. The default quantity is 0, and as the writer points out, "The default value should be 1. I really doubt that anyone will order a quantity of 0." He's right. But the whole screen is a mistake. The user should be presented with a list of items that he can scroll through or access by means of a search. Then he would just change the quantity in the list. In this case, 0 would be an appropriate default. In some applications, the list should retain a particular user-chosen quantity—for example, the last quantity ordered—as the starting point. Depending on the demands of the task, it might be useful to have a Set All Quantities to 0 button. Clearly labeled and undoable, of course.

There is also a Purchase Order Direct Entry (Remove) screen in the system. This screen is unnecessary: Setting the quantity to 0 in a Purchase Order Direct Entry should remove the item from the purchase order.

The screens have another wasteful convention. There are buttons, marked *Save* and *Exit,* at the bottom. Most users are not sure what they do. Does *Save* both save and exit, in which case it should be labeled *Save and Exit,* or do you have to click them both in sequence to get out of the screen with the entry saved? If you exit without saving, do you get a warning, such as "Do you want to save your changes before exiting?" or does it just dump your work? In any case, having either button is unnecessary. Once you move the cursor outside the box and start to do anything else, the system should allow you to do it, saving the contents of the old dialog box.

• When a customer takes the time to carefully analyze and make constructive suggestions about your product, pay attention! It is not an attack or an embarrassment. This person is not an enemy attacking you; he or she is showing loyalty to and interest in your product.

SEVEN

Interface Issues Outside the User Interface

It is a profoundly erroneous truism, repeated by copybooks and by eminent people when they are making speeches, that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. Civilization advances by extending the number of important operations which we can perform without thinking about them.

—Alfred North Whitehead

This chapter presents a potpourri of other technology-related areas where some out-of-the-box thinking (or nonthinking) may be of value to designers. Section 7-1 addresses the fact that programming language environments contain some of the worst human interfaces in the industry. We look at two aspects that could be improved. One is that the initial hurdle in terms of system and development environment has become so large that the beginning programmer is not encouraged to experiment and to learn by doing. The second is that although the benefits of documentation are well known, not much of it is done. A small change to programming languages might make the process easier.

Section 7-2 looks at the profusion of cables growing like snakes on Medusa's head from our computers. We never seem to have the right adapter, extension, or proper kind of cable for what we need. This problem would be far more tractable if we didn't have special male and female ends to cables but if instead every cable for a given function could plug into any like cable or connector on the computer. It is possible to do this.

Section 7-3 discusses a question of ethics: Building interfaces puts the designer into intimate and extended contact with the mind and body of the user. This entails certain responsibilities. Much has been written about how to make sure that there are appropriate curricula and training for practitioners of the interfacial arts, to protect users of their work, but not much has been said of what safeguards and societal protections are needed to allow the competent designer to do good work.

7-1 More Humane Programming Language Environments

7-1-1 System and Development Environment

Programming environments have had even less benefit from research into cognetics than user interfaces have had. There is no question that modern systems are becoming increasingly complex and that programming tools need to accommodate this increasing complexity. Simple things have been made unnecessarily difficult, and we have failed to provide sufficient and sufficiently well-designed software tools needed to ease the difficulties of working in today's computer environment.

I will start with a simple example. To write a program that adds two numbers on the long-dead Apple II, you turn it on (boot time is undetectable!) and press *Control-B,* which gets you into BASIC. If you then type `PRINT 3+4` and press Return, you get 7 instantly and painlessly. From launching BASIC to result: five seconds. As the industry is well aware, ease of use requires copious resources of memory and speed. Therefore, we know that the Apple II can operate with such dispatch and ease of use because it is a powerhouse of brute hardware: 2MHz 8-bit processor; 48K bytes of RAM (all that you could stuff in!), and a 400Kb disk. A 1999 400MHz 32-bit processor with 192MB of RAM (and there's room for more!) and gigabytes of disk storage takes more than three minutes. Considering bus width and processor speed, the new machine is about 1,500 times faster than the old. Considering the time it takes to get started on writing a program, the new machine is about 36 times slower.

I asked two professional programmers to write a Visual Basic (VB) program that adds 3 + 4 and prints the result. The first began by complaining that the machine was hobbled by having only 8MB of memory and was powered by an obsolescent 75MHz 32-bit processor. Not counting the two-minute boot time, BASIC was up in 54 seconds. Then an Insert Module had to be opened, a Get Options box was opened and manipulated, a button and a form were created, and the programmer then had to type the middle line of the following:

```
Private sub Command1_Click ()
MsgBox 3 + 4
End Sub
```

Before this program could be used, it had to be launched and then a button clicked to make it run. Only two or three errors were made during this process, which took 3:40 (again, not counting boot time).

Another programmer, with a 75MHz 32-bit Pentium processor and 40MB of RAM, launched Visual Basic and did the same task in 28 seconds (over five times slower than the Apple II). The program, created by a slightly different process, was

```
Private sub Form-Load ()
MsgBox Str (3 + 4)
End Sub
```

I asked the second programmer why he didn't use what the first programmer had:

```
MsgBox 3 + 4
```

He said he wasn't sure that it would work. In other words, he wasn't sure how VB worked in this case. There's nothing odd about this; like other modern computer languages, VB is huge and inconsistent in design. The rationale for its size is that it makes large projects easier, but that is no reason to make little things more difficult. To the extent that big things are made out of lots of little ones, making small tasks easier also makes the overall task easier. It is only bad language- and system design that explains why one experienced VB programmer made errors and another was not sure of the proper syntax for an elementary program. I got similar times and results with three Smalltalk programmers, thus demonstrating that it is not just VB that has these problems. Obviously, each of these languages also offers many advantages, but if they and, especially, their environments had been better designed from a human factors viewpoint, the advantages could have been had at less human cost.

Something wonderfully straightforward has been lost: in particular, the immediate feedback that humans need in order to quickly iterate their way to an effective program. I am not so naive as to think that we can keep the former simplicity in toto and achieve the level of program complexity that is now demanded, but I am certain that we can do a lot better than we have.

7-1-2 Importance of Documentation in Program Creation

Many sources explain that it is important for programmers to copiously document the code they write. Two reasons are usually given: to aid the reader of the program in understanding it (Knuth 1992, p. 99) and to make it easier to adapt the program to new circumstances (Weinberg 1971, p. 164). A typical program has an occasional line or two of comments and somewhat more frequent single-line comments attached to lines of code. Many programs are almost totally devoid of comments.

As Knuth also noted, writing the comments *before* or as one creates the code seems to make writing code easier, improves the design of the algorithms, lowers the number of errors and iterations required to complete a project, and also gives the advantages usually cited for providing comments. There seems to be a good cognitive reason why Knuth's impressions are correct.

When we develop algorithms and write code as experienced programmers, the process is accomplished, in part, by the cognitive unconscious. As has been noted, this mental facility can sustain contradictions. I suspect that the cause of some programming errors is that your cognitive unconscious is sustaining a contradiction between what you want to do and what the code is telling the computer to do.

However, to write out your intentions clearly in natural language forces you to make your reasoning process conscious, and it is in the cognitive conscious that contradictions most readily become apparent. Even if this hypothesis is incorrect, writing comments makes you think through the problem an extra time, in a different medium, and from another point of view.

Unfortunately, programming environments have been designed to make comments difficult to add. For example, comments in many computer languages are limited to one line; where multiple-line comments are allowed, there is often no word wrap or other features that the simplest word processors have to aid the typing process. (Two exceptions are UCSD Pascal and Oberon.) If you want to edit a paragraph-length comment in Visual

Basic, a language of the 1990s, you have to word-wrap it by hand. Programmers are, according to legend, not especially proficient in spelling, so you would think that a spell-checker would be part of every programming environment, but such a facility is almost never present in a programming environment. In the current iteration of Mathematica, a generally superb program for working with mathematics, comments have been stripped out of programs and put into separate windows, a move in exactly the wrong direction. A few systems, such as Knuth's WEB (1992), were created specifically to aid the documentation process. Another approach, less sophisticated but effective, was taken by the author (Lammers 1986, p. 226).

To keep a program working, any changes to a program must be preceded by changes to the explanation. There should be no separation between programming and any other activity you do with your computer, just as we have seen that there is no need to separate other forms of computer use into applications. Experience has shown that most computer users are reluctant to even try programming. Perhaps if it were integrated into the environment so that snippets of programming would be useful without having to learn a whole programming language or a programming environment, some of the benefits of programming could be more widely realized. It would also be advantageous to be able to take capabilities from various languages and blend them, not into a potpourri, such as PL/I, but in a way analogous to what has been proposed in this book for merging applications. It might be rewarding to be able to meld LISP's structures for list processing, APL (or its progeny, J) with arrays, SNOBOL's powerful string handling, Smalltalk's inheritance and objects, and so Forth.

Cognetics can improve computer interfaces, but the blend of state-of-the-art language design and what we know of human factors has yet to be exploited to any major degree; language designers and interface experts too seldom work together. This is true even though one of the earliest books in the human-computer interface field examined this very question: Weinberg's *Psychology of Computer Programming*, published in 1971, was very far ahead of its time. We haven't caught up yet.

7-2 Modes and Cables

Software runs on hardware. If you can't figure out how to connect the hardware, the software is just so many lost bits. Among the simplest piece of hardware is the cable, a few wires or fiber-optic strands bundled together with a connector at each end. Cables should be able to be plugged and unplugged without having to worry about whether the computer was on or

not. (Cables that are not "hot-swappable" are modal!) You should not have to configure devices, as is required for SCSI connections. With the USB and FireWire standards, these desiderata are finally being addressed. But there are still interface issues that have not been addressed, even with the new standards. For example, it is frustrating to have a cable with the right kind of connector but of the wrong sex. Because there are male and female ends to cables and because some pieces of equipment have connectors that mate with the male ends of cables and others that mate with the female ends of cables, you end up owning a surprising number of variations of each kind of cable. Many computer owners find themselves also purchasing sex-changing adapters because adapters are smaller and less expensive than cables. For example, say you own only male-to-female cables but you need to connect two devices that both have female connectors. You might choose to purchase a male-to-male cable, or you might choose instead to buy a male-to-female adapter and attach it to the female end of the cable you already own. Attaching the adapter results in a cable that is, effectively, male to male.

This dilemma is avoidable, but the methods usually proposed do not work. One solution I have heard is that all connectors on equipment could be standardized to be, say, female, and therefore all connectors on cables would be male. Even so, you would still need female-to-female adapters to join two short cables into one longer one, and a manufacturer would be wise to supply male-to-female cables to act as extensions for existing cables. Following this logic, you can construct situations that require every possible combination of male and female ends on cables and adapters, even with the convention specifying that all connectors on equipment will be female.

A conventional connector pair is typically built as a male connector with pins and a mating female connector with sockets. This approach results in an inventory of eight kinds of parts that can be used as connectors on equipment or cables:

- Male connector for equipment
- Female connector for equipment
- Male connector for cables
- Female connector for cables
- Male-to-female adapters
- Female-to-male adapters
- Male-to-male adapters
- Female-to-female adapters

With hermaphroditic connectors, any two cables connect together, and any cable connects to any connector on a piece of equipment. The inventory is reduced to two items:

- Connector for equipment
- Connector for cables

Electronic requirements force us to have different cables for different kinds of signals, but within each class of cables, nothing in the nature of electronics or manufacturing keeps us from designing hermaphroditic connectors, which are neither male nor female. Any two hermaphroditic connectors of a given class will mate. Surprisingly, any kind of electronic signal connection or electrical power connection can be made via hermaphroditic connectors. This includes multiple-pin connectors, power connectors, and coaxial cables.[1]

If you have two hermaphroditic cables of a given class, you can use the cables either as two separate cables or joined together into one extended cable. In some cases, the hermaphroditic connector would be no more expensive or complex than a standard, sexed connector. This will not always be true; in many cases, a hermaphroditic connector will be somewhat more complex and more expensive to manufacture, but the added expense is counterbalanced by such factors as

- Increased user satisfaction
- Simpler manuals
- Fewer manufacturing setups are required
- Distributors and retailers needing to stock fewer different items

Figure 7.1 shows the schematic of a four-conductor linear hermaphroditic connector. For n conductors in a linear arrangement, you need a minimum of $2n - 1$ contacts. You can trace the conductors in Figure 7.2 to see how it works. Figure 7.3 shows the conductors for a hermaphroditic coaxial connector. The idea can be extended to multiple coaxial conductors. To make better human-computer interfaces possible, we are willing to pay for more complex computers. The same reasoning should be applied to the maddening tangle of wires and cables in which computer and electronics

1. Some cables used in setting up widely spaced components, such as a photographer's remote strobe lights, have hermaphroditic connectors. The now obsolete GR coaxial power radio frequency connectors are another example.

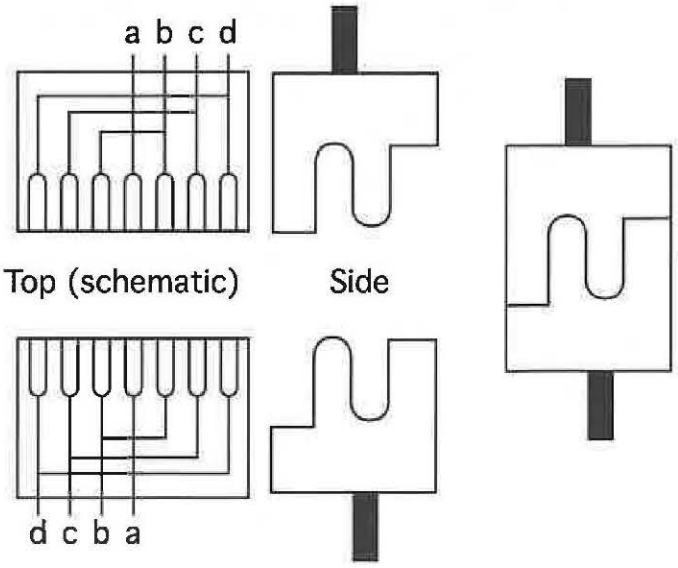


Figure 7.1. A hermaphroditic four-conductor connector. The four conductors are labeled a, b, c, and d.

users are perpetually enmeshed. It may seem that some linear scheme with fewer than $2n - 1$ contacts is required, but keep in mind that connecting cables have also to work as extension cables.

7-3 Ethics and Management of Interface Design

The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable.

—George Bernard Shaw

It is difficult to create a good interface when management does not think that interface design is important. In the short-term view, careful interface design may appear to add to the expense of creating a product and to lengthen the development time. In my experience, the short-term view is wrong even in the short term; improving the user interface often simplifies the design. Careful design and detailed specifications do not hinder but rather speed implementation. A superior interface is also an exceptional long-term investment that returns

- Higher productivity for the customer
- Increased customer satisfaction

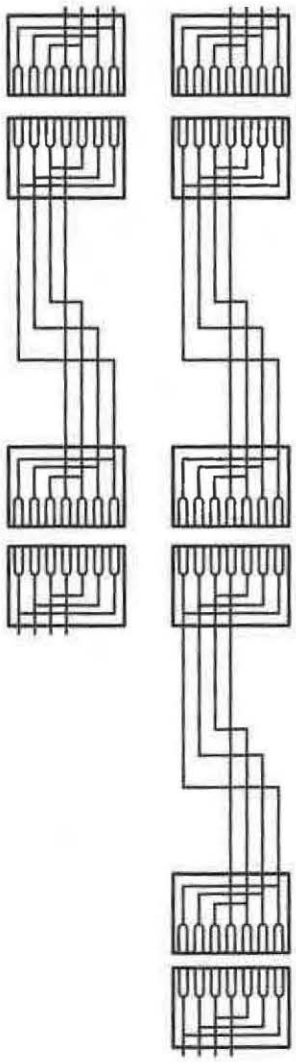


Figure 7.2. Hermaphroditic connector-equipped cables can be used to connect to units (at top and bottom of each column) or as an extension cable (right column).

- A greater perceived value
- A lowered cost of customer support
- Faster and simpler implementation
- A competitive marketing advantage
- Brand loyalty
- Simpler manuals and online help
- Safer products

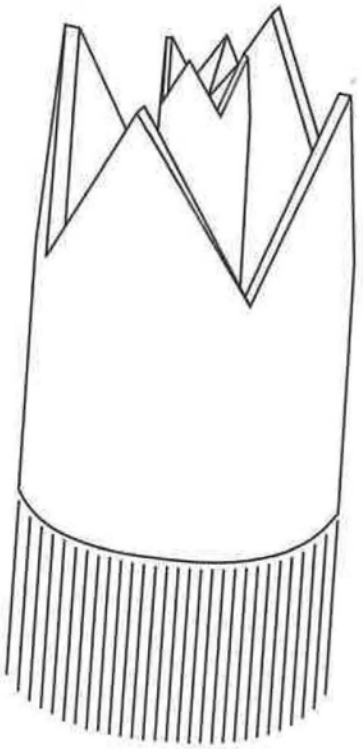


Figure 7.3. Conductors for a hermaphroditic coaxial connector. An outer, insulating sheath with four prongs, similar to the ones shown but with enlarged tips and matching widenings on the bottom (not shown), serve to lock the units together.

Interface designers are rarely in a position to control where in the development cycle interface design will occur and what weight will be given to its pronouncements. Where it has been given primacy, as in the Macintosh project, the results can be spectacular.

Aside from the field being relatively new, so that few practitioners have moved into higher management, another problem is that interface designers have little professional or collective clout. There is some work toward partially solving this problem by proposing educational standards and testing, but a practitioner's having a certificate is no guarantee of competence. The concern addressed here is with the other face of the problem: Assuming that a designer is competent, he or she is often required to design bad interfaces. I note with a trace of envy that doctors have legal safeguards if they wish to do right. For example, doctors can sue for wrongful termination if they are fired for refusing to follow a course of action that poses a threat to their patients. Structural engineers can seek the protection of the law if fired for being asked to violate the canons of their profession.

Interface designers work in an area in which incorrect decisions can contribute to physical injury and psychological debility. For example, excessive key- and mouse clicks in an interface can hasten the onset of or exacerbate existing repetitive stress injuries (RSI). Poor interface design can cause psychological distress. What is needed is a basis for establishing legal safeguards to protect conscientious practitioners. Another necessity is to establish some standards of practice—not interface design standards—for the profession. Measures, such as those discussed in this book and those that will be developed in the future, may allow numerical, objective benchmarks to be established. For example, a structural engineer must show that she has designed a bridge that meets codified standards requiring that it withstand a load of, say, twice the maximum expected, and a car must have less than 0.2 percent carbon monoxide emissions in its exhaust in order to be licensed. Similarly, we may be able to specify that an interface to a word processor is not acceptable if, say, it has an overall information-theoretic efficiency of less than 0.7 or an overall character efficiency of less than 0.8, with no individual feature rating less than 0.5.

Criteria might also be developed such that the average times—and number of keystrokes and graphical input device movements and clicks—to do tasks, averaged over a frequency-of-use weighted set of tasks, in a new word processor cannot exceed those of any prior or contemporary commercial product that does substantially the same task. Products that meet the criteria would receive some form of certification. The criteria would self-adjust as interface technology improves. At present, new products are often more difficult to use than older ones, but there is no way to know this until you've tried the product. Because these criteria affect productivity, and thus the bottom line, enterprise management should take great interest in them. Not only practitioners and management but also consumers would gain protection from the publication of objective standards of interface quality.

Steve Wildstrom, a writer for *Business Week,* pointed out that "computer manufacturers, and especially software publishers, often seem to believe that the requirements of the Uniform Commercial Code only apply to other people" (personal communication, October 1998). Many of the present software licenses that are unilaterally foisted on consumers do not promise even that the software will do the task for which it is advertised. Many of these documents specifically deny merchantability, the concept that the sale of a product automatically carries with it the presumption and promise that the product can do the task for which it was clearly intended. Some states in the United States have passed legislation that nullifies denials of merchantability for computer products; all sovereign powers should do so.

An interface-quality rating system, administered by an independent organization, might be of value to buyers of products that have a significant interface component. User interface design itself must not be regulated or restricted, and we must avoid guidelines that are based on specific interface mechanisms so that innovation will not be penalized, but relative performance ratings that compare the productivity of comparable products will spur designers to move in the right directions.

We must walk a fine line between making a product so novel that users experienced with conventional interfaces will feel lost, and making it so much like standard GUIs that we are failing to aid users to the best of our ability. At one extreme, we must avoid novelty for its own sake, although at the other extreme, we must not lose a valuable marketing opportunity by slavishly making our product too much like other products.

One of the oldest canards in the interface business is the one that says "Maximizing functionality and maintaining simplicity work against each other in the interface" (Microsoft 1995, p. 8). What is true is that adding ad hoc features works against simplicity. But that's just bad design. It is often, but not always, possible to increase functionality without increasing difficulty at a greater rate. Often, added functionality can be had without any added interface complexity; note the difference between interface complexity and task complexity. If the added functionality unifies what had previously been disparate features, the interface can get simpler.

"One way to support simplicity is to reduce the presentation of information to the minimum required to communicate adequately" (Microsoft 1995, p. 8). That is certainly the case, except that I'd replace "adequately" with "well." They err, however, by saying, "For example, avoid wordy descriptions for command names or messages" (Microsoft 1995, p. 8). But what is the minimum required to communicate well? Most of today's interfaces emphasize brevity at the expense of clarity. Why should we have to decipher a cryptic item marked "List" in a word processor's pull-down menu when it could say "Make Index or Table of Contents"? (Remember that a pull-down menu takes up no room from your document, because it goes away as soon as you move the cursor away from it or choose an option.) Do not confuse a clean, open look to the screen with a simple-to-use interface.

Navigation versus White Space

We seem to have a real fear of displaying data in our interfaces. We know that people quickly can find one among a few items much more quickly than they can find one among dozens: There is less to

look through. But it does not follow, as some seem to think, that it is therefore better to have fewer items on each screen. If you have hundreds of items and split them up among dozens of screens, you lose more in navigation time than you gain in searching for the individual item, even if the one you seek is swimming in a sea of similar-looking items. Looking for something in long, gray lists is not always a terrible thing in an interface.

If people weren't good at finding tiny things in long lists, the *Wall Street Journal* would have gone out of business years ago. Would you rather have the stocks listed 15 to a page, each page decorated like a modern GUI screen, even given some scheme for helping you to find the right page, such as:

STOCKS	PAGE
AA–AD	1
AD–AS	2
AT–AZ	3
BA–BK	4
and so forth	

Such a scheme would be considered childish, wasteful, and a nuisance. Yet we sometimes spend more pixels on a screen in making neat, gray-shadowed borders than we do in presenting information. When a person is motivated—by interest or by salary—to refer to data, he finds long lists no problem at all. Visual designer Edward Tufte's (1983, p. 105) first three principles for displaying information are:

- Above all else, show the data.
- Maximize the data-ink ratio.
- Erase nondata ink.

All we need do is substitute *pixels* for *ink* for his advice to apply to display-based devices. A serious, professional user wants screens packed with useful stuff. Screens should be well labeled, with methods to make finding things easier and dense with the information that represents the real value of each screen. (After all, we do have a computer sitting there, and we should be making the best use of it we can.)

There are a multitude of studies on the design of displays. Many early but still useful studies are discussed in Tullis (1984). Some of the results are still valid, such as search time through a list of items is

approximately 30 milliseconds per item (p. 126). Tullis's major results apply to 24-by-80 alphanumeric displays and gave a quantitative evaluation.[2] If the results were extended to today's bitmapped displays and could give a measure of target-finding time, that could be used to optimize not only single, isolated, nonwindowed displays (Tullis's restrictions) but also, in conjunction with estimates of navigation time, could lead to a more global optimization. There is almost certainly a tradeoff between screen complexity and navigational complexity, as Tullis himself noted (p. 132). The tradeoff depends on the speed and ease of navigation and how the data is structured. Where a search facility, such as LEAP, is used for a significant percentage of within-screen searches instead of visual scanning, still other measures would have to be devised. This is a lively area for further research.

In any case, following the popular white-space-makes-for-readability philosophy of screen design to its logical conclusion would teach us that there should only be one item of data on any screen. Thus, the user will always be able to detect it visually with an irreducible minimum of effort.

Having improved many products by decreasing the number of screens and increasing the information on each of the remaining screens—improving the logic of the design to achieve the reduction—I have come to believe that, in almost all commercial software, we have erred on the side of too little data per screen.

The best way to differentiate your product's interface is to make it work. A well-written and cogent argument for the importance of interface issues, directed in large part to management, is *The Invisible Computer* (Norman 1998).

---

2. Tullis's measures did not look at content quality but only layout. It is possible to construct counterexamples, that is, displays that rate highly on his measures but which are difficult to use.

EIGHT

Conclusion

One began to sympathize with the dog whose muzzle is removed at the end of a prolonged Muzzling Order, and who does not quite know what to do with himself.

—*C. G. Grey (in Jane's* All the World's Aircraft *1919)*

If you start with the goal of making interfaces as simple as you can, taking account of our limitations and exploiting our abilities, there are two things you have to do. One is to understand what we can and cannot do, to study the maps of human thought presented by the science of cognitive psychology and see where they lead to the engineering discipline of cognetics. This book has followed one major highway from the map, one that leads from research into the division of abilities between our cognitive conscious and our cognitive unconscious, to an understanding that we have but one locus of attention and a recognition of the central nature of habit formation in how we react to various interface methodologies. We also learn that individual differences are small when dealing with habituation, in contrast to the large differences between individuals in other regards.

From the science, we learn that modes, long recognized as undesirable, are at the heart of some of the most vexing irritants in current

computer interface paradigms. Any complete cure requires that an interface be modeless and is enhanced by as much monotony as we can design in. The uniformity among individuals is exploited. Productivity is improved by our being able to spend less time doing a task, and this observation leads to the interface version of the classical time-and-motion study, the quantitative GOMS analysis. GOMS leads to a feel for what details make interfaces slow and fast to use and leads to the question of how fast and efficient an interface can be and to the need for quantitative measures of efficiency.

Having gone down the road of understanding human capabilities, we then look at our present computer equipment and how we use it, finding a small set of elementary actions and methods that apply to a broad and varied landscape of applications. This uniformity can also be exploited.

We are led by our studies in an unfamiliar direction that turns out to be a shorter path to usability than the standard methods provide. Fast search methods combine with the elimination of unnecessary mechanisms, such as file names and URLs, hierarchical file structures, and applications, to allow us to take speedy shortcuts. The ZIP allows us to fly over the terrain, to see more, and to get there more quickly. We have also taken a few side paths, for example, to see how cables can be made easier to use.

This book has many lacunae; for example, I am sure I have missed prior work that I should have known about, and I may have failed to correctly attribute to their originators ideas that were not my own. Parts of this book lay out areas where, had I the resources, I would have performed experiments to test my conclusions and guesses. Please consider these areas as invitations for research.

Thank you for having read this book and for joining me on a quest for more humane interfaces. Where our journey has crossed untrodden ground, I may have taken a wrong turn here and there. Nonetheless, I am convinced that my compass is a good one and that my overall direction is correct.

Starting from what we know of human cognition, we have been led to fundamental changes in the design of human-machine interfaces. Nothing less will do.

APPENDIX A

The One-Button Mouse: History and Future

Men loven of propre kynde newefangelnesse.

—Chaucer, "The Squire's Tale"

I have been variously castigated and applauded for creating the one-button mouse and some of the basic methods for using it. Questions from readers of drafts of this book indicated that they thought that the way the Macintosh worked was the way that earlier mouse-driven systems from Xerox PARC worked and were curious as to the differences. This appendix describes the mouse-based systems that I saw at Xerox Palo Alto Research Center (PARC) when I was a visitor there. The even earlier use of the mouse at Douglas Englebart's group at the Stanford Research Institute (SRI) was embedded in a system that was decades ahead of its time in many ways and contained valuable ideas that have not yet been widely applied. Englebart's software, however, was often modal and was sometimes inefficient at the keystroke level.

Few users of modern personal computers remember what you had to go through to make a selection with the PARC system, as exemplified by its most popular text editor, BRAVO. Tapping each of the three mouse buttons on the PARC mice will be denoted by *L, M,* and *R* (for left, middle, and right). BRAVO did not use mouse button quasimodes.

To select a character: point to the character, *L.*

To select a word: point to word, *M.*

To select an arbitrary string of characters: point to first character, *L;* point to last character, *R.*

To select a string of words: point to first word, *M;* point to last word, *R.*

A typical mistake was using *L* and then *M,* which merely restarted the selection at what you thought was going to be the end of the desired selection. This was very frustrating, especially when the selection was large. Notice, however, that you never had to "back out" of a selection; you could always just start a new one without penalty. This was a true advance over most contemporaneous systems, which required that you use ESC or some such to cancel a selection-in-progress before you could start another.

There was an earlier editor developed, but not used much, at PARC (and of which I was unaware at the time) that used click and drag to select text. Using the notation presented in Section 3-1: To select some text, point to the top of the text, $L\downarrow$, point to a lower point in the text, $L\uparrow$. But this idea was not extended to other kinds of selection.

At Apple, I demonstrated how we could use one button for selections more broadly: To select any contiguous area on the screen, whether text or not, point to one corner of the region, $L\downarrow$, and point to the other end, $L\uparrow$. This technique came to be called "click and drag."

Other BRAVO methods were also more complex than the Mac's.

To delete text: Select text and type *d.* The problem here was that you had to be aware of the existence of a selection before you knew what typing the letter *d* would do. In the Macintosh, I chose to use the Delete key to delete text both when you are typing and when you are editing.

To insert text: Point to an insertion point, *L;* type *i;* type the new text, tap the ESC key. The ESC key took you out of insert mode. The Macintosh's design did not have an insert mode, and its method has become universal: To insert text, point and click at the desired insertion point, *L;* type new text. Instead of having an explicit delimiter to start the typing, as at PARC, you just started typing. You did not have to press ESC or anything else at the end; merely starting a new task would end the old. In BRAVO, you replaced text with this method: Select old text, type *r,* type new text, tap the ESC key. To replace text on the Macintosh: select old text, tap Delete, type new text. This design differs from the current Macintosh and Windows model, which seems simpler: To replace text, select old text, type new text.

As users have discovered, however, this often causes accidental loss of text. In the original design, which requires one keystroke more than the present method, typing is never destructive, and the text editor is, and feels, more secure; this is a lot to gain from one additional keystroke.

Unlike BRAVO and some other editors designed at PARC, on the Macintosh and, to a large extent, on subsequent systems, typing a plain letter never acts as a command. In retrospect, the improvements seem obvious, but they did not seem so at the time. My design for an interface based on a one-button mouse was refined and extended in discussions with many coworkers, most notably Brian Howard and Bill Atkinson, and, of course, many adjustments were made based on observations made during user testing and during subsequent development. Some users do have difficulty both holding down a button and simultaneously moving a graphical input device, but this is partly dependent on the physical design of the graphical input device and whether it is a mouse or another input device. In the Macintosh design, this problem was alleviated by having only one large button on the mouse, with low holding force and good tactile feedback. (Some of the recent touchpads with poorly designed buttons, especially those on portables, make dragging difficult and error-prone.) Additionally, the improvement and error reduction afforded by the elimination of modes often overweigh the pointing errors due to dragging, even with less-than-optimal graphical input devices.

What I did not see at the time is that multiple buttons on a mouse can work well *if the buttons are labeled*. If the Macintosh mouse had had multiple buttons, if the buttons had been permanently labeled, and if they had been used only for their designated functions, a multiple-button mouse might have been a better choice. A better mouse might have two buttons, marked Select and Activate, on top and on the side, a button activated by a squeezing action of the thumb. This last button would be marked Grab. Some mice at present have a wheel on top that is used primarily for scrolling. Better still would be a small trackball in that location. The mouse would control the position of the cursor; the trackball could be used, for example, to manipulate objects or to make selections from menus that float with the cursor.

APPENDIX B

SwyftCard Interface Theory of Operation

Some of the principles discussed in this book were first published in the SwyftCard manual, released in 1984. SwyftCard, which plugged into the then highly successful Apple II, was simple by today's standards. Appendix B of its manual contained an unusual feature: Along with the usual theory of operation of the hardware, it also contained a theory of operation of the software and what is probably the first appearance of a user interface theory of operation in any commercial product. In a way, that appendix was the beginning of this book. The quoted material is from the second edition (Alzofon and Raskin 1985).

> The paradigms used in SwyftCard were invented to cure a host of problems shared by almost all current systems—most of them small enough in their own right, but which taken together make learning and using conventional software far more time-consuming than necessary, and which make using computers a frustrating and annoying process.
>
> We have always wondered why, for example, you have to format disks—isn't the computer smart enough to see if a disk isn't formatted

and do it if necessary? We find cursor control keys far too slow, and when you consider the number of auxiliary commands they require (move to next/previous word, sentence, paragraph, page, move to beginning/end of line, document, file . . .) we find them too complex. The GID is only a small improvement because most of them take your hands away from the keyboard, and uses up much screen space for menus, scroll bars, and the rest of the associated GID apparatus. We are annoyed when we are put through menus instead of being able to do what we want right now, and we are puzzled by the huge number of commands in most systems. We hate disk systems that allow you to lose work through trivial human error. We are amazed that many word processors cannot keep up with human typing speed.

SwyftCard shows that with proper design all these questions and bothers—and many others that have plagued us for years—can be answered and fixed. And the system works on an inexpensive computer with only one disk drive, with minimal memory requirements. Our product does what most people need done—without an operating system, expensive price tag, or bells and whistles.

The major design principles include numerous innovations, as well as applications of what we have learned from the work of others.

1. The cursor LEAP concept, whose average time to target is about three times faster than that of the most advanced method in common use up until now: the mouse.

2. The cursor itself, whose two parts show you exactly where what you type will appear and where delete will operate. The cursor also collapses upon being moved so that you do not have to aim "one off" if you want to delete.

3. A very small set of fundamental operations that allows you to accomplish a wide range of tasks with ease.

4. The elimination of an operating system, thereby allowing all operations to be performed directly and immediately from the editor without having to go into different modes.

5. The elimination of modes in general, which makes habit formation easy because you do not have to think about what state the system is in to figure out what a command will do. This property is called modelessness.

6. Not providing many ways to do a task—again so that you do not have to think about alternate strategies when you are about

to do something. We call this principle monotony. Like modelessness, monotony aids habit formation.

7. The emphasis on habit formation is itself a fundamental principle of the design, and one often overlooked by other designers. We consider it important that after a brief period of learning, a user should not have to think about the system while using it.

8. The DISK command, which simplifies the usual complexities of a DOS (Disk Operating System) into one simple command. It also provides protection against most common mistakes that would cause a loss of data on other systems. The technique of making one disk correspond to one Text is what makes this command possible.

9. The emphasis on making speed of operation proportional to frequency of use (often-done tasks must be very fast, seldom-done tasks can be slow).

10. What you see is what you get—the way it looks on-screen is the way it prints on paper. (This principle was violated for underlining due to a limitation in Apple display hardware.)

11. Noun-verb design of commands. First you specify what you are going to work on (that gives you time to make sure you are right and to make corrections), then you give the order as to what to do. Some systems work the other way around, or even worse, mix the two styles.

12. It is very hard to louse yourself up or clobber something you are working on. It's not impossible, but it's hard enough to do that it is not likely to happen by chance or a momentary lapse of attention.

13. The inclusion of programming and communications within a general purpose environment, where the output is placed in the editor/retrieval environment.

14. The allowance of months of testing and reworking time in the schedule, so that purchasers of the system are not being used as test subjects.

This is only a barest sketch—the system specs run to some 50 pages—but we hope it gives you a feel for what led us to design SwyftCard the way we did.

REFERENCES

At times, I may have misinterpreted or misunderstood the intent of the works I cite. I apologize in advance for any such errors.

Accot, Johnny, and Shumin Zhai. "Beyond Fitts' Law: Models for Trajectory-Based HCI Tasks" (www.dgp.toronto.edu/~accot/Common/Articles/CHI97/chi.html, 1997).

Alzofon, David, David Caulkins, Jef Raskin, and James Winter. *Canon Cat How-To Guide* (Tokyo: Canon, 1987).

Alzofon, David, and Jef Raskin. *SwyftCard,* 2d ed. (Menlo Park, CA: Information Appliance, 1985).

Anderson, J. R. *Rules of the Mind* (Hillsdale, NJ: Lawrence Erlbaum Associates, 1993).

Apple Computer. *Inside Macintosh,* Vol. 1 (Cupertino, CA: Apple Computer, 1985).

———. *Human Interface Guidelines: The Apple Desktop Interface* (Reading, MA: Addison-Wesley, 1987).

Ashlar. *Vellum 3D Manual* (Sunnyvale, CA: Ashlar, 1995).

Asimov, Isaac. *I Robot* (New York: Bantam Books, 1977).

Baars, Bernard J. *A Cognitive Theory of Consciousness* (Cambridge, U.K: Cambridge University Press, 1988).

Business Week. "Special Report on Information Appliances" (22 Nov. 1993), p. 110.

Buxton, William. "Chunking and Phrasing and the Design of Human-Computer Dialogs," *Information Processing '86: Proceedings of the IFIP 10th World Computer Congress* (Amsterdam: North-Holland, 1986).

Card, Stuart K., Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction* (Hillsdale, NJ: Lawrence Erlbaum Associates, 1983).

Cohen, Jonathan D., and Jonathan W. Schooler, eds. *Scientific Approaches to Consciousness* (Hillsdale, NJ: Lawrence Erlbaum Associates, 1997).

Collins, Richard. *Flying* 121:10, p. 67 (October 1994).

Cooper, Alan. *About Face* (Foster City, CA: IDG Books Worldwide, 1995).

Dennett, Daniel C. *Consciousness Explained* (Boston: Little, Brown, 1991).

Dijksterhuis, E. J. *The Mechanization of the World Picture* (London: Oxford University Press, 1961).

Drori, Offer. "The User Interface in Text Retrieval Systems," *SigCHI Bulletin* 30:3 (1998).

Eriksson, H., and P. Magnus. *UML (Unified Modeling Language) Toolkit* (New York: John Wiley & Sons, 1998).

Garrison, Peter. *Flying* 121:12, p. 112 (December 1994).

———. "Drifting Off Centerline," *Flying* 122:1, p. 43 (January 1995).

Gray, Wayne D., Bonnie E. John, and Michael E. Atwood. "Project Ernestine: Validating a GOMS Analysis for Predicting and Explaining Real-World Task Performance," *Human-Computer Interaction,* 8:3, pp. 237–309 (1993).

Grudin, J. "The Case Against User Interface Consistency," *Communications of the ACM,* pp. 1164–1173 (1989).

Hewlett-Packard. *User Interface Design Rules for the New Wave Office System* (Cupertino, CA: Hewlett-Packard Personal Software Division, 1987).

Hotchkiss, B. "The Car Column," *Pacifica Tribune,* 12 Nov. 1997, p. 14A.

Horton, William. *The Icon Book* (New York: John Wiley, 1994).

IBM. *System Application Architecture, Common User Access, Panel Design and User Interaction* (Boca Raton, FL: IBM, 1988).

Jacobson, Robert, ed. *Information Design* (Cambridge, MA: MIT Press, 1999).

John, Bonnie E. "Why GOMS?" *Interactions:* pp. 80–89 (October 1995).

Johnson, J., and G. Englebeck. "Modes Survey Results," *SigCHI Bulletin* 20:4, pp. 38–50 (1989).

Kaplan, Justin, ed. *Bartlett's Familiar Quotations*, 16th ed. (Boston: Little, Brown, 1992).

Knuth, Donald E. *Literate Programming* (Stanford, CA: Center for the Study of Language and Information, 1992).

Lammers, Susan. *Programmers at Work* (Redmond, WA: Microsoft Press, 1986).

Landauer, Thomas K. *The Trouble with Computers* (Cambridge, MA: MIT Press, 1995).

Laurel, Brenda, ed. *The Art of Human-Computer Interface Design* (Reading, MA: Addison-Wesley, 1990).

Lewis, C., and D. A. Norman. "Designing for Error," in D. Norman and S. Draper, eds., *User Centered System Design* (Hillsdale, NJ: Lawrence Erlbaum Associates, 1986).

Linzmayer, Owen. *Apple Confidential* (San Francisco: No Starch Press, 1999).

Loftus, Elizabeth F. *Eyewitness Testimony* (Cambridge, MA: Harvard University Press, 1979).

———. *Memory* (Reading, MA: Addison-Wesley, 1980).

Mackenzie, I. S. "Movement Time Prediction in Human-Computer Interfaces," in R. M. Baecker, W. A. S. Buxton, J. Grudin, and S. Greenberg, eds., *Readings in Human-Computer Interaction,* 2d ed., pp. 483–493 (Los Altos, CA: Kaufmann, 1995).

Malone, Michael S. *Infinite Loop* (Chicago: Doubleday, 1999).

Mayhew, Deborah. *Principles and Guidelines in Software User Interface Design* (Englewood Cliffs, NJ.: Prentice-Hall, 1992).

Microsoft. *The Windows Interface Guidelines for Software Design* (Redmond, WA: Microsoft Press, 1995).

Miller, George A. "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *Psychological Review* 63, pp. 81–97 (1956).

Moore, J. S., and R. S. Boyer. "A Fast String Searching Algorithm," *Communications of the Association for Computing Machinery* 20:10, pp. 762–772 (1977).

Norman, Donald A. "Categorization of Action Slips," *Psychology Review* 88:1, pp. 1–15 (1981).

———. "Design Rules Based on Analyses of Human Error," *Communications of the ACM* 26:4, p. 255 (1983).

———. *The Psychology of Everyday Things* (New York: Basic Books, 1988).

———. *The Invisible Computer* (Cambridge, MA: MIT Press, 1998).

Penrose, Roger. *The Emperor's New Mind* (London: Oxford University Press, 1989).

Raskin, Jef. "Looking for a Humane Interface: Will Computers Ever Become Easy to Use?" *Communications of the ACM* 40:2, p. 98 (Feb. 1997).

———. "The Quick-Draw Graphics System." Ph.D. diss. (State College, PA: Pennsylvania State University, 1967).

———. "FLOW: A Teaching Language for Computer Programming," *Computers and the Humanities* 8:4 pp. (July 1974).

———. "Computers by the Millions," *SIGPC Newsletter* 5:2 (1982).

———. "Systemic Implications of an Improved Two-Part Cursor," *Proceedings of CHI 89: Human Factors in Computing Systems,* Austin: 30 April 1989, pp. 167–170 (New York: ACM Press, 1989).

———. "Down with GUIs," *Wired* pp. (December 1993).

———. "Intuitive Equals Familiar," *Communications of the ACM* 37:9, pp. (September 1994).

Raskin, Jef, and James Winter. U.S. Patent No. 5,019,806, Method and Apparatus for Control of an Electronic Display, 1991.

Reason, James. *Human Error* (Cambridge, U.K.: Cambridge University Press, 1990).

Shneiderman, Ben. *Designing the User Interface* (Reading, MA: Addison-Wesley, 1987, 1998).

Sellen, A., G. Kurtenbach, and W. Buxton. "The Prevention of Mode Errors Through Sensory Feedback," *Human Computer Interaction* 7:2, pp. 141–164 (1992).

Shannon, Claude E., and Warren Weaver. *The Mathematical Theory of Communication* (Urbana: University of Illinois Press, 1949, reprinted 1963).

Smith, S. E., and D. J. Duell. *Clinical Nursing Skills,* 3rd ed. (East Norwalk, CT: Appleton & Lange, 1992).

Stallman, Richard. *GNU Emacs Manual,* 9th ed. (Cambridge, MA: Free Software Foundation, 1993).

Tesler, Larry. "The Smalltalk Environment," *Byte* (August 1981).

Tesler, Larry, and Timothy Mott. *Report on the Xerox Palo Alto Research Center Gypsy Typescript System* (Palo Alto, CA: Xerox, 20 April 1975).

Thomas, Lewis. *The Lives of a Cell* (New York: Viking Press, 1974).

Tognazzini, Bruce. *Tog on Interface* (Reading, MA: Addison-Wesley, 1992).

Tolkien, J.R.R. (Douglas Anderson, ed.) *The Annotated Hobbit* (Boston: Houghton Mifflin, 1988).

Tufte, Edward. *The Visual Display of Quantitative Information* (Cheshire, CT: Graphics Press, 1983).

Tullis, Thomas S. "Predicting the Usability of Alphanumeric Displays," Ph.D diss., Rice University, 1984.

de Unamuno y Jugo, Miguel. *The Tragic Sense of Life,* Chapter 9 (1913).

Weinberg, Gerald M. *The Psychology of Computer Programming* (New York: Van Nostrand Reinhold, 1971).

INDEX

Note: An online index is available at www.awl.com/cseng/. Search on the keyword "Humane."

About Face, 27
Absorption, 26
  airplane crash and, 25
Accelerator pedal
  interchanged with brake, 20
Accents, 186
Access permission
  and invisible documents, 156
Accidents, 26–27
Actions
  applying to objects, 59
Action-object, 59
Activated content, 104
Activations, 106
Adaption
  adaptive menus, 57, 70
  adaptive palettes, 57
Adjectives, 38–39
Affordances
  and visibility, 62–65
Aircraft
  and buttons that change in the night, 51–52
Aircraft crashes
  absorption related to, 25, 26–27
  preventing, by eliminating a mode, 43–45
America Online
  electronic mail package, 41
AND
  in search patterns, 174
Anderson, J. R., 12
APL, 186, 195
Apple Computer, 168, 172, 182
Apple iBook, 31
Apple Interface Guidelines, 59, 172
Apple Macintosh menu
  and Fitts' Law, 94–96, *95*
Apple Newton, 7
Apple II, 192
  and SwyftCard, 211
Apple IIe, 169
Applications, 46
  cognitive differences among, 104
  as imposition, 148
  liberating commands from, 100
  similarity of, 101
  software, xi
  as visitors, 146–148. *See also* Interfaces; Unification
Apricus, 157
Architecture, xi, 12
Argument, 76
Armstrong, Neil, 124
Ashlar pull-out palette, *58*
Ashlar's Drafting Assistant, 40
Asimov, Isaac, 34
Atkinson, Bill, 209
Attention, 17*ff*
AutoCAD, 40
Autoformat (Microsoft Word), 48
Automaticity, 20–21, 23, 28
Autopilots, 66, 67
Autorepeat, 36, 185
Awareness, 14–16

Baars, Bernard J., 11, 12, 24, 27–28
Backspace command, 42, 43
Backward compatibility, 66
Bandwidth, 8, 87
BASIC, 192–193

Bay Area Rapid Transit (BART)
and affordance avoidance, 64–65
Beginner-expert dichotomy myth, 68–70
Berra, Yogi, 19
Between-character cursor, 136, 138
Bibliographic icons, *173*
Bifurcated interfaces, *91*, 92
Bitmaps, 144
Bits
information measured in, 85
Block cursor, 136
Bob (Microsoft), 58
Boolean, 127, 139, 173*n*
Boot-up times, 7
Boustrophedonic scripts
delete cursor for, 138
Boyer-Moore algorithm, 129
BRAVO, 207, 208, 209
Brevity
and clarity, 202
Browsers
"favorite locations" in, 152
zooming in on list of, *167*
Bumgarner, John, 185
Business Week, 201
Buttons, 190
are fewer better?, 52–54
that change in the night, 51–52

C, 167
C++, 167
Cables, 206
and modes, 195–198
CAD. *See* Computer-aided design
Calculation, 47, 72, 76–78, 89, 95, 178
Calculator
versus computer, 141–142
Canon, 182
Canon Cat, 29–30, *30*, 31, 50, 100, 121, 122, 127, 129, 174, 180
Calculate key built in, 142
disk formatting on, 182
ex-IBMers and testing, 174
"Local Leap" on, 138
and quasimodes, 56
Use Front key on, *56*, 57
Caps Lock key, 39, 55
Card, Stuart K., 71, 76, 97
Cartoons,
emphasizing users' problems, 154–155.
"Case Against User Interface Consistency, The" (Grudin), 189
Case studies
messages, 180
noun-verb fix to verb-noun problem, 60–62
Cathodes, television set, 7
Certification
for interface technology, 201
Cervantes, Miguel de, 1
Chair ergonomics, 10
Character-at-a-time interaction, 130
Character efficiency, 89
Character sequences
need for sameness of, 127–128
Chaucer, Geoffrey, 87
Check box, 38, 39
Chord keyboards, 185
Chunking, 37
Circular searching, 130, 132
City Slickers, 2
Clarity
and brevity, 202
of function, 70
and overuse of graphics, 172
Clark, Larry, 39
Click and drag technique, 208
Clicking, 34
Climate-control mode, 51–52
Clip art, 172
Clockwork, 12
Close button, 106
Coaxial cables, 197
Cockpit, 25, 51, 66
Cognetics, 10, 12, 195, 205
and ergonomics, 9–11
and interface design, 168
and locus of attention, 9–32
Cognitive conscious, 13, 14, 205
and cognitive unconscious, 11–16
limitations of, 24
properties of, *16*
Cognitive Theory of Consciousness, A (Baars), 11
Cognitive unconscious, 11, 13, 14, 205
habit-forming property of, 22
properties of, *16*
Collaborative space
ZIP, 156

Color
  display, 101
  and icons, 170
  palette, 171–172
Command key, 36, 171
Command line-driven systems, 50
Commands, 109–115
  improved method of invoking, 111
  liberating from applications, 100
  and transformers, 143–148
  and visibility, 112
Commands-plus-transformers computer environment, 145
Comments, 194, 195
Composite selection, 107
Compound document, 140
Computer
  boot-up times, 7
  calculator *versus*, 141–142
  crashes, 5, 6
  humane interface for, 102
  problems, xix
  and productive outcomes, 188
  as servant, 189
  user agitation and unexpected behavior by, 27
  and wasted time, 6
Computer-aided design, 40
Computer games, 63
Computer interfaces
  cognetics and improvement in, 195
  hardware comprising, 101
Confirmation steps
  and fixed responses, 22–23
Conscious and unconscious, 15. *See also* Cognitive conscious; Cognitive unconscious
Consciousness
  and models of the mind, 12–13
Consciousness Explained (Dennett), 12–13
Considerate programming
  applications as visitors, 146–148
Content
  defined, 33–34
  elementary operations on, 104
  and quasimode, 59
  structure as part of, 121
Control direction checks (RPVs), 44
Control functions
  quasimodes reserved for, 59
Control principle, 183
Control sticks (RPVs), 43, 44
Copied content, 104
Copilot, 51
Copy command, 123
Cosmology, 15
Count-down clocks, 47
Crashes (computer), 147
Critical-path method GOMS (CPM-GOMS), 72, 74
Crystal, Billy, 3
Cursors, 34, 105, 106, 209, 212
  Canon Cat and placement of, 31
  design of, and making selections, 133–136
  and Fitts' Law, 93–96
  graphical design of, 133–136
  and Hick's Law, 96
  insertion/deletion portions, *137*
  and LEAP, 136–139, 212
  standard between-character, *133*
  two-part, *134*
  in Vellum, 40
Customer-centered design, 3
Customer feedback, 187–190
Customizations, 48
Cut and paste, 177–178

DataViz, 145
Decay,
  of sensory input, 18, 31
Decryption, 177
Dedicated numeric keypad, 90
Defaults, 190
Delays, 36, 184
Deleted content, 104
Delete key, 125
Deletion
  humane design for, 178
Deletion of text
  BRAVO methods for, 208
Delimited search, 125
Delimited-search dialog boxes, 132
Delimited searches, 126, 128, 129, 133
Delimiters, 76, 120
  search-pattern, 127–129
Dennett, Daniel, 13
Design cycle
  interface design in, 5–6
Design innovation
  tools not facilitating, 4–5
Desktop-based computer systems, 31

Diacritical marks, 186
Dialog boxes, 188, 189, 190
  with check boxes for searches/launches, *131*
  with different launch buttons, *131*
Digital filter program, 144n11
Dilemma of preemption, 139
Direct perceptions, 18
Disambiguate, 126
Disappearing button, 51
DISK command, 181–182, 213
Disk drives, 179
Disk formatting, 211
Displays
  design of, 203–204
  and ergonomics, 10
Display states of objects, 115–117
Distance
  and Fitts' Law, 93, *94*
Disturbance
  and absorption, 27
Divide-by-0, 179
Documentation
  importance of, in program creation, 194–195
Documents
  compound, 140
Domain experts, 4
DOS (Disk Operating System), 213
Double clicking, 34, 74–75
Double quasimode, 109
Douglass, Frederick, 33
Drag-and-drop, 114
Dragging, 34, 106, 114
Duell, D. J., 22
Dynamic memory allocation, 118

Edsel, 119
  and absorption, 26
Efficiency
  character, 89
  and information, 85
  measurement of with interfaces, 83–87
  and thermodynamics, 84
Eject (a disk), 179, 181, 182
Electronic mail (e-mail), 176
  America Online package, 41
Elementary actions, 103
  cataloged, 103–117
  commands, 109–115
  display states of objects, 115–117
  highlighting, indications, and selection, 105–109
  and uniformity, 101–103
Elementary operations, 104
EMACS, 125
E-mail. *See* Electronic mail
Embedded drawing programs, 140
Emotional needs, 58
Emperor's New Mind, The (Penrose), 12
Empirical conscious, 11
Empirical unconscious, 11
Encryption, 177
Englebart, Douglas, 207
Environmental parameters, computer, 146
Equiprobable messages, 89
Ergonomics
  and cognetics: what we can/cannot do, 9–11
Error messages, 27, 178
  transparent, *117*
Error rate
  of bifurcated converter, 92
Error reduction
  and noun-verb interaction, 59
Errors
  ambiguous notation and, 35
  with double clicking, 74
  mode, 40
  with toggles, 39
Ethics
  and management of interface design, 198, 200–204
Eudora, 100

Fast string search, 129
Feature bloat/complexity
  decreasing, 146–147
Female connectors
  on equipment/cables, 196
Female-to-female adapters, 196
Female-to-male adapters, 196
Fiber optics, 195
File names, 100
  described, 118
  eliminating, 119
  and structures, 117–123
Files
  and file names, 118

Filters, 144n11
Fingerprints, 183
FireWire standards, 196
Fitts' Law, 71, 93–96, 97
  and penalty with small targets, 138
  and scrolling, 115
  and standard text cursor, 133
Fixed delays, 184
Fixed responses
  and confirmation steps, 22
Flashlights, 37–38, 46
Floppy disks, 182
Fluke oscilloscope, *52*, 53
Focus, 108–109
  versus locus, 17
Font size
  and icons, 170
Footnotes
  zooming in on, 156
Forward erase, 134
Four-conductor linear hermaphroditic connector, 197, *198*
Foveal, 17, 40, 135
Fowles, Bob, 47
Freud, 11
Functionality
  and simplicity, 202
Function buttons, 52
Fundamentals
  importance of, xvii–xix

Games, 146
Generated content, 104
Gestures, 139
  and comparative task analysis, 73
  defined, 37
  modal, 42
  and monotony, 67
GIDs. *See* Graphical input devices
Global Village, 129
Goals, objects, methods, and selection rules. *See* GOMS model
GOMS analysis, 206
  and autorepeat, 185
GOMS calculation examples
  Hal's interface: solution 2, GUI, 80–83
  Hal's interface: Solution 1, dialog box, 78–80
GOMS keystroke-level analysis
  for comparing GID and LEAP timings, 136
GOMS keystroke-level model, 72–83
GOMS calculation, 76–77
GOMS calculation examples, 77–83
  interface timings, 73–75
GOMS model, 71
Grab cursor, 115
Graphical input devices, 34, 101, 103, 212
  buttons, 34, 143
  comparative task analysis, 73
  cursor moves with, 135
  and double clicking, 74–75
  and one-button mouse, 209
  with two-part cursor, 138
  and zooming, 164
  and zoom quasimodes, 154, 155. *See also* Cursors; Mouse
Graphical user interfaces, 2, 49, 122, 123
  for Hal's interface, *81*
  Hal's interface: solution 2, 80–83
  and icons, 170
  searches initiated with, 129
Graphics, 119
Grey, C. G., 205
Grimes, Martha, 71
Grudin, J., 189
GUIs. *See* Graphical user interfaces

Habit formation, 18–20, 21, 22, 213
Habituation, 205
  and buttons, 52
  and Hick's Law, 96
Hal, 77–78
Hal's interfaces
  efficiency of, 87–90
  other solutions for, 90–92
Handheld computers, 7
Handheld personal data managers, xi
Handwriting tablet, 101
Hashing, 118
Heftman, Rex, 143
Heisenberg, Werner Karl, 83
Help systems, 175
Hermaphroditic coaxial connector
  conductors for, 197, *200*
Hermaphroditic connector-equipped cables, *199*
Hermaphroditic four-conductor connector, 197, *198*

Heuristics, 4
Hick's Law, 71, 96–97, 164
Hierarchical file structures
  eliminating, 121
Highlighting, 105
Horn, Bob, 170
Horton, William, 169, 170, 172
Hospitals
  ZoomWorld images of, *162*, *163*
Howard, Brian, 209
Human-centered design
  and user-centered design, 3–4
Human-computer interfaces, 2. *See also* Interfaces
Humane interfaces, 1
  definition of, 6–8
  ideal, 20
  single ranges in, 46
  techniques and Help facilities in, 174–177
  and transformers, 144
  typing in, 113
Humane programming language environments, 192–195
  importance of documentation in program creation, 194–195
  system and development environment, 192–194
Human-machine design
  and habit formation, 20
Human-machine interaction, xi
  and quantitative methods, 72
Human-machine interfaces, xviii, 2
  Fitts' Law applied to, 94
  and locus of attention, 9
  modal, 42
  revolution in design of, 206
  surmounting problems in, xii

iBook, 31
Icon Book, The (Horton), 169
Icons, 34, 63, 122, 168–174, 189
  bibliographic, *173*
  particularly obscure, *169*
ICU. *See* Intensive care unit
IDE (Borland), 129
IEEE. *See* Institute for Electrical and Electronic Engineers
Implementation
  influences on, 5
Incremental searches, 125–126, 128, 129, 132, 133
Indicated content, 104
Indication, 105–106
Information Appliance, 130, 133, 185
Information display
  Tufte's principles of, 203
Information efficiency, 84
Information-theoretic efficiency, 84, 90
  dialog box, *84*
Information theory, 83
  interface design related to, 87
Input devices, computer, 101. *See also* Graphical input devices
Insertion of text
  BRAVO methods for, 208
Institute for Electrical and Electronic Engineers, 178
Intensive care unit
  medical care chart used in, 157
  ZIP view of, *158*
  zooming out beyond, *161*
Interface-building tools, 4
Interface design, xi, 1, 71, 151
  accommodating of human universalities, 68–69
  Asimov's first law of robotics applied to, 6
  and cognetic technology, 168
  in design cycle, 5–6
  estimating error rates from, 80
  ethics and management of, 198, 200–204
  and habit formation, 22, 23
  information theory related to, 87
  and modes, 33
  quantitative laws relating to, 93
  ranking of performance times, 74
  and sense perception, 18
  simplifying, 100
  and standards of practice, 201
  without modes, 50
  and ZUIs, 166
Interface designers, 3, 72
  and beginner-expert dichotomy, 69
  and development cycle, 200
  and universal psychological facts, 4
  and visibility, 63
Interface efficiency
  measurement of, 83–87
Interface operation errors
  preventing, 27

Interfaces
and absorption, 26
definition of, 2
definition of humane, 6–8
and delays, 36
designing of, before implementing interface design, 5
and ergonomics of the mind, 10
and gesture, 37
and monotony, 67
nature of, 1
nonmodal, 139
personality of user tailored to, 58
problems with, 187–188
quantitative analyses of, 71–72
to searches, 124–127
self-teaching, 175
simplicity of, 2–3
unified, 100. *See also* Humane interfaces; Human-machine interfaces; Interface design
Interface timings, 73–75
Interference, 21
Internet, xix, 12, 99, 153. *See also* World Wide Web
Interrupted work
resumption of, 31–32
Intuitive interface, 149, 150, 151
Invisible Computer, The (Norman), 204
Invisible interface feature, 63

J, 195
JavaScript, 126
Alert command in, 84
Jazz, 156n3
Jefferson, Thomas, 117
Jennings, Karla, 37
John, Bonnie E., 73
Joysticks, 34, 43, 103

Karpinski, Dick, 18
Kay, Alan, 141
Keyboards, 101, 109–111
design, 11
and ergonomics, 10
full-chord, 109
interfaces, 36
shortcuts based on, 67
time delays and tricks with, 184–187. *See also* Cursors; Mouse
Keystroke-level model (GOMS), 72
Keyword, 174, 175
Knobs, 63
Knuth, Donald E., 194, 195

Labels
clarity preferred over brevity in, 167
icons *versus*, 170
on mouse buttons, 209
visibility enhanced by, 51
Landauer, Thomas K., 139
Landmarks, 152, 153, 154
Language conventions
bending machine to, 113
Launch buttons, 130
dialog box with different, *131*
Lazar, David, 11
LEAP, 129, 135, 155, 168, 175, 204
and cursor position, 136–139
document selection with, 122
superiority of, 139
LEAP AGAIN, 138
LEAPing, 138
Learning times
and natural GOMS language, 72
Left-to-right languages
delete cursor for, 137–138
Legended buttons, 51
Lightpens, 34
LISP, 195
"Local Leap," 138
Lock command, 116
Locked text
and typing, 175–176
Lock with Password command, 116
Locus of attention, 17–32, 43, 109, 205
and choice of methods, 67
execution of simultaneous tasks, 20–23
exploitation of single, 29–30
focus collocated with, 108
and formation of habits, 18–20
and habituating features, 57
and launch buttons, 131
and modality of interface, 42
and mode, 40, 41
and noun-verb interaction, 59, 60
origins of, 27–29

resumption of interrupted work, 31–32
and side effect, 113
singularity of, 24–27
Lower-cost software packages
failure of, 145

Machine response times
and user actions, 75
Macintosh computer, 172, 200, 207
and one-button mouse, 209
resetting RAM parameter on, 36
Macintosh interface
delays with, 184
guidelines, 4
Macintosh operating system, xi
Macro, 46
Macromedia Flash, 167–168
Magnetic-resonance imaging, 15
Male connectors
on equipment/cables, 196
Male-to-female adapters, 196
Male-to-male adapters, 196
Management
of interface design, 198, 200–204
Manhole, The, 150
Markoff process, 87
Martin, Pam, 75n1
Masking, 30
Mathematica, 195
Maxwell, James Clerk, 72
Mayhew, Deborah, 72, 170
Mazes
antithesis of, 152
Meaning
and information, 86–87
Measures of efficiency, 71
Medical care charts
zooming in on, *159*, *160*
Memory, 14, 32. *See also* Locus of attention
Mental operators
heuristics for placing, 76–77 (table)
Menus, 34, 50, 110, 112, 190, 212
adaptive, 57
experiments on structures of, 97
systems based on, 67
Merchantability
denials of, 201
Messages
measurement of information embodied in, 86
user, 178–183
Messerschmidt, Willy, 6
Microphones, 101, 135
Microsoft Office, 143
Microsoft Windows menu
and Fitts' Law, 94–96, *95*
Microsoft Word, 23, 47, 49, 100
search box, *128*
set-up-then-launch dialog box in, 131
Mill, John Stuart, 150
Mind
consciousness of, 12–13
ergonomics of, 10
Mobile Office magazine, 2
Modal design, 50
Mode errors, 40, 55
frequency of, 40
minimizing, 41
Modeless interfaces, 67, 68, 206
Modelessness, 46
Model-search dialog box, *124*
Modes, 33, 37–42, 205
aircraft crash prevention through elimination of, 43–45
and cables, 195–198
definition of, 42–43
elimination of, 212
error reduction and elimination of, 209
and quasimodes, 55–59
Modified content, 104
Monotonous interface, 67, 68
Monotony, 33, 66–68, 206, 213
Moran, Thomas P., 71, 76, 97
Mouse, 34, 151
double clicking, 74–75
grab function on, 115
for working with ZIP, 168
Move command, 123
Moved content, 104
MRI. *See* Magnetic-resonance imaging
Multiple-character commands, 50
Multiple-line comments, 194
Multiple-option switches, 46
Multiple-pin connectors, 197

Nass, Clifford, 68
Natural GOMS language, 72
Natural interfaces, 150, 151
Natural language, 35, 120, 194
conventions, 113

Navigation, 149
  white space *versus*, 202–204
  ZoomWorld, 152–168
Newell, Allen, 76, 97
NextWave (HP), 140
NGOMSL. *See* Natural GOMS language
N-key rollover, 186, 187
Noik, Emanuel, 164
Nomenclature
  and notations, 33–36
Nonkeyboard devices, 87
Nonmodal interfaces, 139
Nonvolatile memory, 32
Norman, Donald, 12, 40, 41, 72, 156n3, 204
Notations
  ambiguous, 35
  and nomenclature, 33–36
Noun-verb constructions
  versus verb-noun constructions, 59–62
Noun-verb design of commands, 213

Oberon, 194
Objects
  actions applied to, 59
  display states of, 115–117
  highlighting of, 105
  and locus of attention, 109
  sameness for, 102
OCR programs. *See* Optical character recognition programs
Old selection, 106
OLE software (Microsoft), 140
One-button mouse, 34
  history and future, 207–209
One-on-one interaction with human user
  ease and pleasantness of, xix
OpenDoc (Apple), 140
Operating system, xviii
  elimination of, 212
Optical character recognition programs, 144
Oscilloscopes, 169
Overlapping windows, 141
Overstriking, 187
  with quasimode, 186

Pace of interaction
  user setting of, 8

PAD++, 156n3
Paddle switches, 45
Page breaks, 120
Paint programs
  and noun-verb interaction, 60
Palm Pilot, 7, 102
Parallel processing, 28
PARC cursor, 138
PARC (Palo Alto Research Center), 141, 207, 209
Passwords, 183–184
Patterns, 124
  in searches, 133
Penrose, Roger, 12, 24
Perceptual memory, 18
Perl, 167
Persistence of vision, 18
Personalization features
  time wasted in learning/operating, 48–49
PET. *See* Positron-emission tomography
Photo-processing programs, 104
PL/I, 195
Positional cues, 152
Positron-emission tomography, 15
Pound, Ezra, 66
Power connectors, 197
Preemption dilemma solution, 139–140
Preferences, 47
Pressure-sensitive graphic tablets, 103
Product
  interface as, 5
Productivity
  designing for, 5
Programmers, 193
  and documentation, 194–195
Programming
  considerate, 146–148. *See also* Humane programming language environments
Progress bar, 76
Psi effect, 153
Psychology of Computer Programming (Weinberg), 195
Pushbuttons, 63

Quantification, 71–97
  Fitts' Law and Hick's Law, 93–97
  GOMS keystroke-level model, 72–83
  measurement of interface efficiency, 83–92
  quantitative analyses of interfaces, 71–72

Quantitative guide
for improvement, 92
Quasimodal, 55
Quasimodes
and modes, 55–59
overstriking with, 186
for searches, 129
Querying, 104

Radio
truck, *53*, 54
Radio buttons, 38–39
dialog box solution with, *79*
labeling with adjectives, *39*
Radio-controlled model aircraft, 43
Ranking
of performance times of interface designs, 74
Rating systems
interface-quality, 202
Redo command, 107, 108
Redo key, *108*
Remote-piloted vehicles, 43
commercial controller, *43*
switch handles on, *45*
Repetitive stress injuries, 201
Resetting computer programs, 148
Reversibility
and noun-verb interaction, 60
Right-to-left languages
delete cursor for, 138
Robinson, Spider, 109
Robotics
Asimov's first law of, 6
Rollover, 186, 187
RPVs. *See* Remote-piloted vehicles
RSI. *See* Repetitive stress injuries

Saving work, 5
Scanning, 144
Scope of activity
and modes, 45
Screen Lock command, 116
Screen Unlock command, 116
Scrolling, 114, 133, 164, 209
replacement for, 115
SCSI connections, 196
Searches
interface to, 124–127
Search-pattern delimiters, 127–129
Search quasimodes, 129
Security
and sign-ons, 183–184
Selected content, 104
Selecting, 106
Selection, 106–107
separating drag gestures from, 114
transparency of, 116–117
Self-teaching interfaces, 175
Separators. *See* delimiters.
Separator characters, 120
Servo
in RPV, 43, 44, 45
Sex-changing adapters, 196
Shakespeare, William, xiii
Shannon, Claude E., 86
Shaw, George Bernard, 198
Shift key, 35
Shneiderman, Ben, 72
Short term memory, 28n4, 63
and toggles, 39
Side effect
eliminating, 113
Sign-ons
simplified, 183–184
Simplicity, 70, 100, 205
factors supporting/working against, 202
and noun-verb interaction, 60
with tasks, 2–3
Sims, Admiral William S., 149
Simultaneous tasks
execution of, 20–23
Single personhood, 28
Single-use interface design, xix
Sleep mode, 7, 50
Smalltalk, 167, 195
Smith, S. F., 22
SNOBOL, 195
Snow, C. P., 2
Software
licenses, 201
selling on command-by-command basis, 144
unused features of, 145
Sony, 182
2010 short-wave radio, *54*
Sound files, 119
Space bar, 35
Space-making tactics
with ZoomWorld, 155

Special characters, 128
Spector, Lincoln, 139
Speech recognizer, 101
Speed
  of incremental searches, 129
  and information efficiency, 85
  with LEAP, 135
  and noun-verb interaction, 59
Spell checking, 102
Spreadsheets, 104, 177
Spring-loaded mode, 55
Spring-locked mode, 55
SRI. *See* Stanford Research Institute
Standards of practice, 201
Stanford Research Institute, 207
Stanley Steamer, 7
Steam-powered automobile, 7
STM. *See* Short term memory
Strachey, C., 99
String, 76
  defined, 124
String searches, 124, 133
  fast, 129
  and find mechanisms, 124–132
Structures
  and file names, 117–123
Subjective ratings
  and interfaces optimizing productivity, 49
Swinehart, Dan, 139
SwyftCard
  interface theory of operation, 211–213
SwyftWare, 121, 129, 138, 174
Syntax
  for commands, 113
System and development environment, 192–194

Tables
  and ergonomics, 10
Tablet pens, 34
Tap, 34, 35
Target, 124
  and Fitts' Law, 93, 94
Task analysis
  comparative, 73
Task definition, 5
Task domain experts, 4
Tektronix oscilloscope, *53*
Telephone voice-response system, 2
Television sets, 7
Temperature-conversion, 77–83, 87–92
Temperature converter, *81*
Temporary modes, 49
Tesler, Larry, 141
Testing, 170, 213
Text, 124
  BRAVO and insertion/deletion of, 208
  commands for changing, 116
  and cut and paste, 177–178
  drag-and-drop in, 114
  encrypted, 177
  as visual cue, 170, 189. *See also* Visibility
Text-editing rules
  universal, 99
Text editors, 140
Text file
  content of, as best name for, 119
Text pointing
  with GIDs, 135
Thermodynamics
  efficiency defined in, 84
Thermometer, 80–83, *81*
Thomas, Lewis, 19
Thompson, D'Arcy Wentworth, 71, 73
Ticket machines, 64–65
Time
  sacredness of, 5–6
Time delays, 30
  and keyboard tricks, 184–187
Toggles, 39, 46, 108
  labeling difficulties with, 38
Touchpads, 34, 209
Touch screen, 101
Tracer
  in Vellum, 40
Trackballs, 34, 209
Transformed content, 104
Transformers
  and commands, 143–148
  and vendors, 145
Transparent dialog box, *117*
Truck radio, *53*, 54
Tufte, Edward, 203
Tullis's restrictions, 204
Twain, Mark, 178
Two-part cursor, *134*, 138
Typing
  accents, 186
  commands, 116

and locked text, 175–176
and selection, 110–113
speed, 74. *See also* Text

UCSD Pascal, 194
UML Toolkit, 5
Unamuno, Miguel de, 19
Unconscious
and conscious, 15
Undefined value, 178–179
Undo command, 107, 108, 109
Undo key, *108*
Unification, 99–148
applications abolished, 139–142
commands and transformers, 143–148
cursor design and strategy for making selections, 133–136
cursor position and LEAP, 136–139
elementary actions cataloged, 103–117
file names and structures, 117–123
string searches and find mechanisms, 124–132
uniformity and elementary actions, 101–103
Uniform Commercial Code, 201
Uniformity
and elementary actions, 101–103
Unit of input
program interaction with smallest, 129–130
Units of interaction, 129–132
Unlock command, 116
Unlock with Password command, 116
URLs, 206
USB standards, 196
User-centered design
and human-centered design, 3–4
User interfaces, 2
fundamental flaws in, xi. *See also* Interfaces
User interface theory of operation
and SwyftCard, 211
User-maintained mode, 55
User messages, 178–183
User preference settings, 47, 48
User(s)
letter from, 187–188
pace of interaction set by, 8

Variable-legend buttons, 52
VCR
easy-to-set digital clock on, *3*
Vellum, 40, 49
adaptive palette for, 57
Vending machines, 64
Vendors
and transformers, 145
Verb-noun constructions
versus noun-verb constructions, 59–62
Vertically oriented languages
delete cursor for, 138
Veteran's Administration Hospital (Palo Alto)
LEAP testing at, 135
Video inputs, 101
Visibility, 70
and affordances, 62–65
and commands, 112
and icons, 168, 174
labels enhancing, 51
lack of, 107
Visible interface feature, 62
Visual Basic, 4, 126, 193, 194–195
Visual C++, 4
Voice-controlled systems
LEAP usefulness in, 135
Voiceprinting, 183

Weaver, 86
WEB, 195
Web pages, 31
and higher-bandwidth communication channels, 8
programs, 104
Web sites, xi
Weinberg, Gerald M., 195
Whitehead, Alfred North, 191
White space
versus navigation, 202–204
Whole-text searches, 119
Wildstrom, Steve, 201
William of Occam, 101
Windowing paradigm, 141
Windows interface guidelines, 4
Windows operating system, xi
Windows 2000 (Microsoft)
adaptive menus in, 70

Winter, James, 180, 181, 182
Word processors, 31, 104, 140, 212
Work
  sacredness of, 5–6
World Wide Web, 12
  surfing in ZoomWorld, 164
  zooming in to one area of, *166*
  ZoomWorld portal to, *165*

Xerox PARC. *See* PARC

ZIP. *See* Zooming interface paradigm
Zooming in, 154
Zooming interface paradigm, 153
  browser replaced by, 164
  working effectively in, 156
Zooming user interfaces, 166, 206
Zoom quasimodes, 154
ZoomWorld, 152–168
ZUIs. *See* Zooming user interfaces

This book is published as part of ACM Press Books—a collaboration between the Association for Computing (ACM) and Addison Wesley Longman. ACM is the oldest and largest educational and scientific society in the information technology field. Through its high-quality publications and services, ACM is a major force in advancing the skills and knowledge of IT professionals throughout the world. For further information about ACM, contact:

ACM Member Services
1515 Broadway, 17th Floor
New York, NY 10036-5701
Phone: 1-212-626-0500
Fax: 1-212-944-1318
E-mail: ACMHELP@ACM.org

ACM European Service Center
108 Cowley Road
Oxford OX4IJF
United Kingdom
Phone: +44-1865-382338
Fax: +44-1865-381338
E-mail: acm.europe@acm.org
URL: http://www.acm.org

The Humane Interface *was composed in 11-point Bembo with QuarkXpress by Stratford Publishing Services of Brattleboro, Vermont. Karen Savary designed the text. Jennifer L. Collins designed the cover (with help from Aza Raskin). The text was printed on sixty-pound recycled paper by Courier in Stoughton, Massachusetts.*

Interface Design/Graphics

THE HUMANE INTERFACE

"Deep thinking is rare in this field where most companies are glad to copy designs that were great back in the 1970s. *The Humane Interface* is a gourmet dish from a master chef. Five mice!"

—Jakob Nielsen, Nielsen Norman Group
Author of *Designing Web Usability: The Practice of Simplicity*

This unique guide to interactive system design reflects the experience and vision of Jef Raskin, the creator of the Apple Macintosh project. Other books may show how to use today's widgets and interface ideas effectively. Raskin, however, demonstrates that many current interface paradigms are dead ends, and that making computers significantly easier to use requires new approaches. He explains how to effect desperately needed changes, offering a wealth of innovative and specific interface ideas for software designers, developers, and product managers.

The Apple Macintosh helped to introduce a previous revolution in computer interface design, drawing on the best available technology to establish many of the interface techniques and methods now universal in the computer industry. With this book, Raskin proves again both his farsightedness and his practicality. He also demonstrates how design ideas must be built on a scientific basis, presenting just enough cognitive psychology to link the interface of the future to the experimental evidence and to show *why* that interface will work.

Raskin observes that our honeymoon with digital technology is over: We are tired of having to learn huge, arcane programs to do even the simplest of tasks; we have had our fill of crashing computers; and we are fatigued by the continual pressure to upgrade. ***The Humane Interface*** delivers a way for computers, information appliances, and other technology-driven products to continue to advance in power and expand their range of applicability, while becoming free of the hassles and obscurities that plague present products.

Jef Raskin (www.jefraskin.com) is a user interface and system design consultant based in Pacifica, California. He invented the Apple Macintosh and the Canon Cat computers. His clients range from start-ups to multinationals such as Hewlett-Packard, IBM, Motorola, NCR, Xerox, Ricoh, Canon, McKesson, and AT&T. Raskin's extensive publications include some 500 articles that have been published in over forty periodicals including *Wired*, *Forbes ASAP*, *IEEE Spectrum*, *IEEE Computer*, *Nature*, *Quantum*, newspapers, and the *Communications of the ACM*. Raskin, who has taught at the University of California, Stanford University, and elsewhere, often leads seminars and has made many radio and TV appearances, as well as speaking at numerous seminars and conferences.

Cover design by Jennifer L. Collins (with Aza Raskin)

The cover illustration was drawn by the author using Ashlar Vellum, a product discussed in the book.

acm PRESS

Text printed on recycled paper

ADDISON–WESLEY
Pearson Education

X00455S0AT
Used - Good - The Humane Interface... for Designing Interactive Systems
$24.95 US $37.50 CANADA