# The Art of Human-Computer Interface Design

Edited by
Brenda Laurel

S. Joy Mountford,
Manager of the Human Interface Group,
Apple Computer, Inc., conceived of
and technically supported the development of
this book

ADDISON-WESLEY
An imprint of Addison Wesley Longman, Inc.
Reading, Massachusetts • Harlow, England • Menlo Park, California
Berkeley, California • Don Mills, Ontario • Sydney
Bonn • Amsterdam • Tokyo • Mexico City

# Creativity and Design
## Introduction

**Thomas D. Erickson**
*Advanced Technology Group*
*Apple Computer, Inc.*

WHY DID THEY DESIGN IT LIKE *THAT*?

Voiced in tones of baffled disbelief, this is a question that many computer users have asked. After all, there are so many bad interfaces around, with such obvious problems. Even the Macintosh—that paragon of good interface design—even the Macintosh has its warts! The infamous Font/DA Mover. The bewildering Installer Utility. Or everybody's favorite wart: ejecting a diskette by dragging its icon to the trash can. What were they thinking of when they did *that*?

Because the chapters in this section deal with how to solve interface design problems, it's important to begin with an understanding of why such obvious problems may be so difficult. Thus, I'm going to tell you how the Macintosh trash icon came to be used as a disk ejection mechanism. It's a good illustration of the difficulties involved in designing an interface for even a simple task.

### The Truth about the Trash

The use of the trash can to eject a disk was present from the very beginning of the Macintosh interface. As is usually the case in interface design, it's important to understand the situation. The original Mac had no hard disk; only a single diskette drive. When a diskette was in the machine, a diskette icon would be displayed and a list of its files would appear in a window. Because most users typically would switch back and forth between several diskettes during a session, it was deemed appropriate for the Mac to keep a memory image of the lists of files on the various disks, regardless of whether the diskette was actually inserted in the drive. Diskettes that the Mac "knew about," but that weren't actually in the drive, were represented by grayed-out diskette and file icons. Thus, a user could select a grayed-out file icon and open it, and the Mac would eject the current diskette and prompt the user to insert the proper diskette. This was all well and good, and made life easier for the user.

There was one drawback. Often, during the course of a session, the user would finish using a particular diskette. The actual diskette could be ejected, but the grayed-out diskette and file icons would remain, taking up both limited screen space and memory. To reclaim valuable space, the now-unwanted list of files represented by the grayed-out icon could be thrown away by dragging it to the trash. Thus, when users decided they were really finished with a diskette, they had to do two things: use the eject command to get rid of the physical disk, and throw away the memory image of the disk by dragging it to the trash.

This annoyed one of the programmers. Why should users have to do two things to accomplish one, frequently desired, action? Clearly, if users are going to discard the image of the diskette that's in memory, they're not interested in keeping the physical diskette in the Mac's drive. Thus, the now-infamous shortcut: dragging the icon of the currently inserted diskette to the trash can would both eject the physical desk and delete the memory image. From the programmer's viewpoint, it was a logical extension of existing functionality. And convenient. After all, wasn't making things easier for the user what the Mac was all about?

The programmer's decision caused a dispute within the design team. In particular, objections were heard from the member of the team, a non-programmer, who was responsible for testing the interface and taking the user's point of view. It was agreed that the tester try it out until the next release. After a week of using it, she couldn't give it up. The rest is history.

I like this story because it is a good example, in miniature, of the realities under which the design process must operate. Not the design process as it ought to be, perhaps, but the design process as it really is. The problem was relatively simple: give users a simple way of ejecting a disk and freeing up memory. But there was no simple, obvious solution. The possible solutions—do nothing, eject via the trash icon, or create a new command or icon for ejection—each had its own trade-offs. Doing nothing retained simplicity and ease of learning at the expense of ease of use. Ejection via the trash icon provided the desired functionality without adding a new interface element, but it compromised the consistency of the trash can's behavior. Creating a new interface object increased ease of use and maintained behavioral consistency at the expense of increasing the number of interface objects. Finally, arriving at a solution involved the interaction—requiring communication and agreement about methods of evaluation and criteria for success—of two people with very different backgrounds.

The point here is not whether the solution was a good one. Rather, the point is to highlight the multiple difficulties of interface design.

## Why Interface Design Is Hard

Interface problems are often obvious. Solutions are less obvious. It may be difficult to find a solution that solves a particular problem without creating new problems. Even then, a separate solution for every problem would result in an interface of such complexity that it would be unusable. What is really needed is a solution that elegantly solves a range of problems. Such solutions are exceptionally difficult to find.

The difficulty of interface design is compounded by the fact that virtually all solutions are compromises. Solutions are shaped by a multitude of problems that are invisible to those outside of the design process. A wonderfully intuitive solution doesn't matter if the system architecture doesn't support it, or if the resulting code takes up too much memory or runs too slowly. Other problems stem from the basic capabilities of humans, and the requirements of the tasks users wish to do. It doesn't matter if the interface responds instantly, if the user can't use it. Solutions to an interface problem involve compromise. But how do designers determine what an acceptable compromise is? How do designers figure out acceptable tradeoffs between speed and intuitiveness and other seemingly contradictory values and requirements?

Not only does the sheer number of requirements increase the difficulty of interface design, but the variety of sources from which the requirements come requires that successful interface design be a multidisciplinary process. The multidisciplinary nature of interface design introduces problems that are political in nature. Psychologists, graphic designers, writers, industrial designers, and programmers all have essential contributions to make to the design of an interface. Yet each discipline has its own priorities and perspectives, its own methods, its own criteria for success. Often these are in conflict with one another. Whose priorities are most important? Whose perspectives are most valuable? Whose criteria for success should be met? Figuring out how to resolve conflicts between differing approaches is not easy.

There are three sorts of reasons why interface design is difficult. First, quite simply, it's hard to come up with good solutions. Second, there are so many competing desiderata involved in interface problems that any solution is bound to be a compromise. The problem here is one of evaluation: how do designers figure out which compromises will fly, and which are to be avoided? The third reason for the difficulty of interface design is that it's interdisciplinary and highly political.

## Making Interface Design Less Difficult

The chapters in this section provide a look at the methods of some of the most skilled and experienced interface designers in the business. The interface designers contributing to this section come from a variety of backgrounds: industrial design, programming, writing, psychology, and graphic design. Most contributors have experience in two or more of these fields. Most contributors have also had the experience of working on interdisciplinary teams. As a result, the chapters in this section contain a broad array of ideas, methods, perspectives, and approaches. Whether you are a designer who wants to broaden your repertoire of methods, or simply a user who wants a better understanding of the design process, this section has a lot to offer.

Here's a sampling of questions addressed by the chapters in this section: What does the evolution of language suggest about ways in which the Macintosh interface might evolve? What can interface designers learn from fields like animation, theater, and architecture? Why do designers from different disciplines have such trouble communicating, and what can be done about it? What happens when you give a programmer and a graphic designer the same interface problem and let them duke it out? Why is cardboard an important interface design tool? People talk about *interface metaphors*, but what exactly are they and how do you design a metaphor? Everyone agrees that consistency is a Good Thing, but how can consistency be conserved when each release adds new features? How on earth can designers do all the testing and tweaking they advocate, when the program isn't working until the week before it ships? Everyone agrees that it's essential to involve users in the design process, but how do you do that?

If you're intrigued, read on. The chapters that follow address these questions and many others.