CHAPTER 1

USABILITY VERSUS PLEASANTNESS-OF-USE

So You Think Computers Are Easy to Use

If you think that now that we have windowing environments, with menus and dialogue boxes and various buttons, all controllable by a mouse, plus sophisticated computer graphics, not to mention multimedia, computers have at last become easy to use, make the following experiments.

Experiment 1: Transferring a file from a PC to a Unix server

Find a person who works on a PC which is connected to one or more Unix servers via a network, but who has not had occasion ever to copy a file from the PC to one of the servers. Be sure that the person has all relevant PC, network, and Unix manuals on hand, and tell him or her that full use of any on-line help facilities is allowed. Now ask the person to select any small text file in any directory on the PC's hard disk, and copy it to a specified directory on the server. Time how long it takes the person *to find out how to perform this task*. (You will not need a watch with a second hand.) Then ask yourself how long it should have taken.

The answer is less than 25 seconds, with or without on-line help, and this book shows you how to create documentation that makes that possible.

Experiment 2: Inserting a bullet (•) in a FrameMaker document

Find a person who uses FrameMaker on the PC but who has never had occasion to insert a bullet at an arbitrary point in a FrameMaker text. (This is not the same thing as creating a bulleted list.) Be sure that the person has all relevant PC and Windows documentation, and tell him or her that full use of on-line help facilities is allowed. Now ask the person to insert a bullet at an arbitrary point in a document. Time how long it takes the person *to find out how to perform the task*. (You will not need a watch with a second hand.) Then ask yourself how long it should have taken.

The answer is less than 25 seconds, with or without on-line help, and this book shows you how to create documentation that makes that possible.

Everyone who uses computers on a daily basis can give numerous equivalent examples of simple tasks that he or she needed to perform in the course of daily work, and the extraordinary length of time, cleverness, and memorizing that was necessary to figure out how to perform these tasks if no one was on hand to answer questions, or if the person (probably with justification) felt that asking would lower his or her worth in the eyes of coworkers. To give just a few additional examples:

• On a PC, finding out how to place an icon representing an application, on the desktop.

• In FrameMaker, finding out how to change all the margins for a set of documents.

• In Unix, finding out how to cause a command, or a script, to be executed at some specified time in the future; or finding out how to copy a directory to floppy disks if the directory requires more than one disk.

Yet there is no need for computer manufacturers to continue to force their customers, and their own employees, to go on wasting time like this. And, sooner or later, a smart manufacturer is going to figure out just how much time *is* being wasted, and realize that if (s)he can reduce that time by a factor of ten or twenty or more, that customers will want to buy his or her products *even if the computation speed is less than that of the competition's product!*

Remarkably, the answer does not depend on more sophisticated computer graphics or windows environments or multimedia. The answer depends solely on a new approach to documentation, and that is what this book is about. That approach requires that we begin by thinking straight about the difference between usability and what we might call *pleasantness-of-use*.

The Difference Between Usability and Pleasantness-of-Use

There are many definitions of *usability*, some of them vague, and all that I know of indiscriminately mixing two separate and by no means necessarily related characteristics of a computer system, namely:

(1) the degree to which a user can accomplish any of the tasks made possible by the system using *only* the on- and/or off-line documentation provided, regardless of how tedious the accomplishment of these tasks may be (this is what I am calling *usability*); and

(2) the degree to which the accomplishing of these tasks is pleasant — or at least not unpleasant — for the user.

Thus, a system can have a high degree of usability and a low degree of pleasantness-of-use.

The failure — even by designers of computer-human interfaces — to recognize the importance of the difference is one reason why computers remain difficult to use, despite all the hoopla about friendliness and despite the explosion of new graphics technologies. I have heard project managers proclaim that a new product will have increased usability because the terminal display will be in four colors, rather than merely in black and white; or that a system has much better usability because it uses this or that windows facility, or because it incorporates advanced graphics, etc.

You would think that if anyone were fully appreciative of the difference between usability and pleasantness-of-use, it would be human factors experts. But in my experience this has not been the case. Here again we see the same concentration on pleasantness-of-use (and of course, where required, on safety): the concentration on designing pretty terminal displays, nonglare screens, comfortable chairs, in the naive belief that, with enough psychology and enough art, usability (as I am defining it) will result.

Let me repeat: Pleasantness-of-use is not at all equivalent to usability. You can write a correct procedure that enables, say, 90% of the intended users of a computer system to accomplish a given task — e.g., transfer a file from one disk on a network to another — and at the same time you can know throughout the writing, and have that knowledge confirmed by user experience, that the procedure is tedious and time-consuming. You have delivered on usability but not on pleasantness-of-use.

Of course, pleasantness-of-use is always a desirable characteristic of a computer system, but in my opinion we are not going to develop a method for consistently providing this characteristic until we learn how to make systems usable according to the above definition. This book tells you how to do just that. It gets you to first base.

Lest We Forget...

No matter how interesting these new developments may be to us in the interface design field, we must always keep in mind that the goal of *both* usability and pleasantness-of-use is to sell more products. Is there any reason to believe that they achieve this goal? Good data is still hard to come by, but the major computer companies — Hewlett-Packard, IBM, Apple — are already spending millions of dollars annually on improving documentation and human factors for their products. So at least we can say that the management of these companies *believes* that usability and pleasantness of use pay off. Furthermore, articles in trade and business journals continue to emphasize the importance of usability to sales. See, for example,

- "I Can't Work This Thing!", editorial, *Business Week*, (April 29, 1991): pp 58 ff.

- Nakamura, Roxanna Li. “The X Factor.” *Infoworld*, (Nov. 19, 1990): pp 51 ff.
- Rettig, Marc. “Nobody Reads Documentation.” *Comm. of the ACM*, (July, 1991): pp 19 ff.

Probably the best argument at present for improving usability is that it makes lower prices possible because it allows a company to decrease the amount of training and support which customers need.

Current State of the Art

Progress in computer-human interface design has run along two lines: First, the development of ever more advanced menu and graphic technologies so that, at the time of this writing, multimedia systems are available which enable authors to combine text, sound, video, and animation. The other line has been that of task-orientation, a break with the old idea that, in order to use a system, you first have to understand how it works. The old idea was natural, given that manuals were written by, or under the supervision of, the engineers and programmers who created the software. Having spent months or years writing the software, engineers and programmers naturally felt that the center of interest for any user was how the software worked. Unfortunately, as the number of programs grew it became more and more difficult for even the most dedicated users to pay the price of, in essence, always having to learn about *everything* before they could do *anything*. Also, of course, the community of users expanded far beyond software professionals who were capable of understanding the software in order to learn how to use it.

A task-oriented manual explains how to carry out the various tasks that are made possible by the system, and typically explains very little, if anything, about how the software which makes those tasks possible is constructed. Task-orientation led directly to the idea of usability testing, i.e., testing whether users could actually perform the tasks using only the documentation, screen help, and other aids provided with the system.

Yet the old paradigm dies hard. In the mid eighties, I heard a reliable report of a manager who expected a technical writer to write a manual working from the software alone, without interrupting the programmers — i.e., to figure out the use from the poorly commented, or uncommented, programs alone: an expectation which reveals just how out of touch with reality some managers were and, I regret to say, still are. I can still remember the shock that greeted the first suggestions that users might be able to use programs without knowing how they were constructed — such suggestions could only come from one who was the worst possible thing one could be: Not a Team Player.

At present, computer-human interface design is carried out in three disciplines: technical writing, human factors, and programming. Unfortunately, each continues to stubbornly defend its own territory. The technical writers, most of whom majored in English or other humanities disciplines and are not technically educated, fight for improved usability and pleasantness of use through better writing. The human factors people hold to ergonomics, and continue to confuse pleasantness-of-use with usability. The programmers continue to see the answer in ever more dazzling technology.

The Birth of a New Profession

I once heard a middle manager say, regarding a software product under development in his department, "in the final weeks, we'll just call in a tech writer to pull it all together."

If we built houses that way, we would hire carpenters, plumbers, electricians, and masons, tell them what kind of a building they were to build, have them go to work, then, a few weeks before tenants were to move in, call in an architect to "pull it all together," i.e., to do the plans which will tell the tenants where the various rooms are, and how to get from one to the other.

The analogy to a building, in software, is *not* the software itself, still less the hardware, but the *use structure* — what in this book I am calling the *Environment*. In good software design of the future, designers will structure the use *first*, or at least they will always put use structure before software and hardware structure; the latter two are simply means of achieving the first. They will decide how they want the user to be able to use the system, *then* write the software.

I know this prediction will go down hard with most programmers and, in particular, with hackers, who always want to think in terms of *things* — programs — first. But I hope this book will convince at least a few of them that progress does not lie in that direction.

I don't know what titles such use designers will have. In this book, I have used *Environment designer.* Maybe one of the currently used titles will eventually dominate: *technical writer*, *documenter*, *learning products developer*, *computer-*

*human interface designe*r, *human factors expert*. At this point, I think it is a waste of time to try to decide on the best title. But there is no question that a new and very exciting profession is being born. This profession will be to the design of software what architecture is to the design of buildings. It will incorporate the skills of documenters, human factors experts, and programmers working in computer-human interface design. It will differ from the work of these groups today in that it will be much more methodical in its approach, and in that the effectiveness of its product will be much more easily measurable. Its guiding rule will be, Design the *use*!

I believe that this book will be the first manual for that profession.