Usefulness and Usability

Usefulness

Upscale's department store installed a checkout machine intended for better inventory control and customer service. The screen showed a menu of items for sale. It started with a list of general categories: housewares, men's clothing, and so forth, each represented by a colorful picture, an icon. Choose an icon by touching it with your finger, and up comes a more detailed list—sport clothes, suits, shoes . . . each represented by another colorful icon. And on down to the individual item type. No longer did clerks need to know or enter the item type or number—so long as their interpretation of the icons and categories matched that of the designer. I once tried to buy some candles. Three clerks struggled for fifteen minutes to enter the sale. Candles were under vases under platters under mixers under chairs.[1]

It's easy for a computer to offer operations that don't help people. For example, computer schemes for finding books in libraries and for presenting them electronically make it harder to find the books and harder to read them. Later we'll see how dumb applications have been turned around by user-centered design. For now, let's look at some of the underlying problems.

Phase one computer applications perform the same task that a human might otherwise do. Programming for these applications demands near-perfect understanding of what the task is and how it can be accomplished. In tasks like bookkeeping and gun aiming there are well-defined

formulas by which the task can be performed, principles that humans would follow precisely if they could. These can be translated directly into computer programs. It has been said that the first industrial revolution replaced human and animal muscle with more efficient energy sources and that the information revolution is replacing human mental work with more efficient electronic processes. That's the way phase one applications works. Phase one computing has had two effects. First, it has pushed forward the replacement of human motor skills with more accurately controlled mechanical ones. Second, it has largely replaced humans in simple acts of arithmetic and filing. Arithmetic is an easily described skill and one at which no mathematical John Henry would stand a chance.

The second phase of computer application, helping people think, is surprisingly more difficult. Note how shallow are the components of writing assumed by word processors, how superficial the aspects of business decision making taken on by spreadsheets. The mental processes of composing memos and documents, of making medical and business decisions, of negotiating and persuading, of formulating plans, and communicating ideas will not soon be captured and imprisoned in a machine. No one really knows how humans do these mysterious things.[2] Humans have amazing memories from which details can be recalled in response to any hint about any part of the original experience. (What time did your childhood neighbors eat dinner on Christmas?) People have astounding visual and auditory pattern-recognizing ability, can easily identify one among a hundred thousand faces or words. In reading text, a literate adult can guess a missing from context half the . And these are paltry feats compared to what goes on in the mind during the comprehension, appreciation, or creation of a serious (or funny) verbal passage. No matter what you may have read in an airline magazine, we can't yet come close to these human abilities in any programmable process.

In services, the role of the computer is to help people help people. Thinking how to do this has proved more difficult than technologists expected. Upscale's worthless point-of-sales device is worse than most others perhaps, but so far not enough are good enough. Reservation and information retrieval systems help people do some important business

chores more easily. So do bar code inventory systems, income tax preparation programs, email, and several dozen more. But most department store service has gotten worse since computers; so has the handling of telephone inquiries and dozens more. Figuring out really good things to do with computers in service businesses is difficult.

Usability

I tried to change a character in the bibliography of this book to put in a French diacritical mark. After an hour of reading three manuals (for the text editor, the bibliography program, the bibliography program's enhancement package) and many experiments, I gave up. I hope the Académie Français will forgive me.

A phase two system is unlikely to be useful unless it is easy to operate, although the opposite is possible: a system that is easy to operate but of no value. The division between usefulness and usability is sometimes fuzzy; one can't tell whether a system flunks for one reason or the other. Was the Upscale department store's menu a failure because it was hard to use or because it wasn't useful? Or was it useless because it was hard to use?

Nevertheless, usability is the key issue in thousands of cases. In a system in which you sometimes enter *K* to "keep" a program and in other cases enter *K* to "kill" the same program, there is a usability problem no matter how useful the program might otherwise be. Because different programs are written by different programmers at different times and marketed by different companies, such inconsistencies are rife. Even when companies like Apple, and more recently IBM and others, attempt to set up interface consistency standards, they don't always have the desired effect. Sometimes it's impossible to maintain consistency as the functions of a system multiply. Bruce Tognazinni, Apple's erstwhile human interface evangelist, has written, "The Macintosh promises consistency, and it is a promise that is dreadfully hard to keep" (Tognazzini 1992). The Macintosh user interface guidelines say the menu bar (a strip along the top of the screen from which available actions can be chosen by a mouse click) must always be visible. But here comes a program

intended to produce a slide show on the screen. Should we leave the menu bar visible on each slide? Of course not.

Standards often force designers to use suboptimal designs; the best way to lay out the screen or structure the user system dialog is not the same for different jobs. For consistency, interface standards say that every computer action should be invoked by the same series of user actions: for example, first click a choice from the main menu bar, then pick one of the subsequently displayed submenu options, and then, perhaps, confirm or type in needed information—like a line thickness or a search term. But use this interface to control a steel-rolling plant and you'd have consistently crumpled metal; all those steps would take much too long. As Emerson might have observed, a foolish consistency is the hobgoblin of little computers.

Another problem is the incredibly rapid change in computer technology. Last year's hot program is this year's dog. Everybody is in a constant state of computer illiteracy. The situation cries out for simplicity, but the cry goes unheard. Instead we get ever greater complexity.

Such problems are not the sole property of PCs. Consider the control of advanced telephone services by a Touch-Tone pad. Transferring a call to another telephone or adding on a third participant can be useful, but remembering to do it by punching 23# is taxing. Thus the familiar, "if this doesn't work, call me back."

The list that follows describes some major phase two applications and contrasts their amazing feats and flaws. If you're not a computer maven, this survey should give you an idea of what a wonderful whiz the computer that's in trouble is.

Two Tales from DEC

A group of researchers led by John Whiteside at Digital Equipment Corporation (DEC) has contributed greatly to the development of UCD methods. One of their techniques, a user-derived interface, illustrates the value of gathering data on what users really do as a foundation of design. In a delightful example, Dennis Wixon pretended to be the computer as test users in an adjoining room struggled with a series of miniversions of an electronic mail system. The first version, a better-than-average first stab, was based on a melding of features and commands found in popular email systems. Test users, professionals or professionals in training, were

given a brief tutorial and a set of eight common email tasks to accomplish in an hour. For example they were to "see that Mike Good gets the message about the keyboard study from Crowling." With the initial version, none of the four test users was able to do any of the tasks without help; Mike never got his message. By the end of the project, many novices could do all eight on their own (Good et al. 1984).[2]

What happened in between? The big problem at the beginning was that users tried to type words to the computer that it didn't understand. For the test problem about sending messages to Mike Good, users tried commands containing the words *memo*, *from*, *about*, and *copy*, none of which the computer understood. Here's how that was fixed. After every test session, Mike, who was actually the project's chief designer-programmer, did get one important message: a list of all the words and phrases users had tried. He analyzed these and made changes to the system's parser, the part of the program that interprets what users type. The majority of changes were just adding synonyms for the words needed in commands—*memo* and *note* for *message*, for example. The team kept track of how many of the users' commands got by the parser. At the outset, the system recognized only one of fourteen inputs; the rest were turned back with reports like "Stopped understanding at 'memo' in message." Figure 11.1 shows how the proportion of successes grew as more changes based on user behavior were added. At the end of the project, three-quarters of the commands offered up by inexperienced users were handled correctly. Additional tests showed that the ultimate design was easy to use for both novices and expert users and that it was more favorably rated than the standard system used at the outset.

Edison read every scrap of prior science and technology possibly relevant to a problem before stepping into his lab. Then he did literally thousands of experiments on the way to each major invention. So did Bell, and the Wright brothers, who built a novel wind tunnel where they performed over 200 experiments on wing lift and drag, to add to their hundreds of experiments with kites and gliders, before they did scores of experiments with powered airships. Incandescence, speech acoustics, and aerodynamics are certainly no more complex and mysterious than human thought. We need wind tunnels for mindships too.

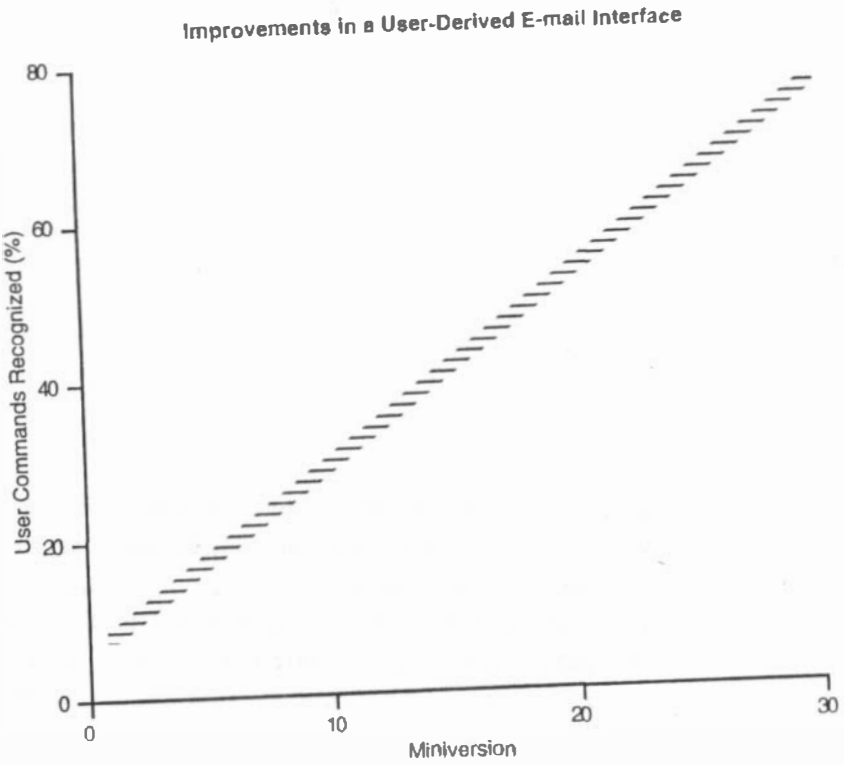

Figure 11.1.
At DEC, the interface to an email system was improved by changes that made it accept commands in the way test users spontaneously entered them. The initial version was similar to existing programs. Changes were made in a series of thirty slightly altered versions as data were collected. (Good et al. 1984.) Note: The authors do not provide data separately for each miniversion and probably did not test sufficiently to plot the percentage recognized for each. The figure depicts the overall improvement from the first to the thirtieth version, which they do report.

The second DEC story is from the same group and is about the development of the text editor EVE. The email program was an experimental prototype explicitly developed for the purposes of designing a single product. The UCD effort for the text editor was aimed at improving its usability in parallel with a larger project of which it was a component. The UCD activity was constrained by both the overall development schedule and the congealed functionality of the larger system. The strategy was to make an early version of the editor available within the company, to seek users and use, complainrs and suggestions. As usability problems were identified, they were corrected. When suggestions for changes looked valuable to the usability specialists and if they were not contradicted by other suggestions, did not overload the complexity of the interface, and were technically feasible, they were tried. Over 200 suggestions were received in the first four months, almost half of them in the first month after wide distribution. Of the 200, about 140 were implemented. After the fifth month many fewer suggestions were made and almost none implemented. The system's usability was measured in the eighth month and found adequate, well before the project's final deadline. This was the editor that, despite being nongraphical, tied for second place against those in the Roberts and Moran tests.

These three examples illustrate a kind of progression in the placement of UCD in the design life cycle and correspondingly different purposes. The map story shows how first studying the intellectual difficulties of a practical user problem—in this case way-finding—and evaluating possible solutions can help to formulate the *what* of a computer system, and steer design toward the right functions.

The email example shows how, given a decision of what the system is for-—electronic text message exchange, in this case—the details of design can be greatly improved by careful study of the behavior of users trying to use a mock-up or prototype.

The text editor example illustrates the perfection of a system that already exists, albeit in a still malleable form. The text editor was put out for real use and the experience of users harnessed systematically to pull its usability forward.

The IBM 1984 Olympic Message System

At IBM's Yorktown Heights, New York, research lab in late 1983, a team of five, led by Steve Boise and John Gould, were handed a formidable job: to design a totally new communications system for the Summer Olympics to be held nine months later in Los Angeles (Gould et al. 1987b). The system would make it possible for athletes housed in Olympic Village dorms to send and receive recorded voice messages to and from each other, to coaches and staff outside the villages, and to friends and relatives in all parts of the world. The system would have to accept input from any kind of telephone anywhere. Arabic-speaking parents in Marrakech, Mandarin-speaking friends in Hong Kong, Norwegian coaches in Los Angeles, as well as the Olympians, would have to learn how to use the system.

In 1984 there were almost no public voice message systems operating anywhere. Not everyone in the United States, and hardly anyone in Kiev, had used an answering machine. IBM had built and experimented with a voice store and forward system in the 1970s but had not imagined using it in this way. In Los Angeles, the system would be set up at two dozen kiosks in the villages connected to a dozen national Olympic Committee headquarters. Olympians would operate the system themselves from the kiosks. Non-Olympians would leave messages by calling into committee headquarters, where an operator would connect them to the system.

Design of the system was wide open and daunting. What functions should it provide: forwarding, message editing, distribution lists, nicknames, or just bare bones? What kind of interface and what sorts of instructions could make the system easy enough to use for such a constituency under such conditions? The stakes were high: the competitors would care a great deal; the prestigious Olympic Committee, the press, and the public would be watching. The system would operate for just four weeks; if it didn't succeed at first, there would be no try again.

The team started by composing a set of scenarios—detailed descriptions of imaginary users interacting with the system, engaging in dialogs with its prompts, exercising its intended functions. These documents provided concrete examples for potential users—the Olympic Committee,

parents, programmers, and operators—to think about, discuss, and criticize. On the basis of these comments, the distribution list feature was dropped before ever being designed, as was a facility to verify that a message had been received and another to place calls to Olympians with waiting messages. Simplicity and manageability began to take center stage.

The next step was the production and test of user guides. Versions of these instructions for both the athletes and for their families and friends were produced. They were written *before* any code was programmed and shown to potential users. Comments and questions about the instructions shaped the features and functions of the system, not vice versa; for example, an easy method for headquarters operators to switch the language of the prompts was discovered to be necessary. In response to suggestions from readers and tests with system simulations, 200 modifications were made to the Olympian version of the guide, over fifty to the family and friends version. The brief user guides became the definitive design requirement documents for the project. The fact that the guides had already settled a vast number of design questions meant that many fewer changes would be made in the working prototypes.

Only a few weeks after the project was begun, computer simulations of the system and interface were tested. An experimenter read the prompts—instead of the eventual recorded-speech module—and typed the received voice messages into a file for later reading aloud to recipients. The researchers used a rapid prototyping and interface building kit that made it easy to transfer successful trial programs to the final system. In the early stages, lab personnel and visitors were the test users, not world-class athletes or their cousins, but they were nevertheless able to show up glaring errors: that four alternatives in an audio menu were too many—"Press 1, listen again; 2, listen to another new message; 3, send a message; 4, hang up"—and stated backward, "To listen again, press 1 . . ." is better.

Demonstrations and test trials with users from outside the United States began as soon as a working audio prototype was available and turned up many problems. The foreign users, less accustomed to computers and electronic interfaces, were more easily overwhelmed. Some functions that had been integrated already were dropped, for example, a

facility to insert material in the middle of a message. The team also interviewed Olympians to see what life, schedules, and desires would be like. Over objections of the officials, who thought maps and expert opinions should suffice, they toured the site, making critical discoveries, for example, that classroom training would be impractical—users would have to learn as they used—and that smog, heat, and desert bugs would create hardware problems.

They tested the system overseas with overseas users and telephone systems, learning yet more lessons about the virtue of simplicity: functions for reviewing and revising messages were dropped, an example interaction added to the guide pamphlet. They tested the kiosk design in a hallway of the IBM research center, starting with a huge hollow cylinder with pasted-on concept drawings of the interface. Passersby edited and commented, and changes were made. Plans posted on the wall led to suggestions from strolling craftspeople. Moving on to a working prototype system in the still-changeable shell, they posted received messages for fellow workers on the computer screen, much as they would later for competitors. Layout, scanning rate, colors, translations, wordings of the official French and English instructions were improved. Intensive, systematic usability tests were conducted with about a hundred people. Interactive interface bugs were caught: prompts that misled, improper handling of time-outs for pauses. They held contests for employees to see who could find a problem, who could sign on most and send the most messages. More bugs were caught. They invited local computer science students to come by in the evenings and try to destroy the system.

Next came field tests at a pre-Olympic event with competitors from sixty-five countries. In five days the team discovered fifty-seven more usability items too severe to ignore. Despite the translated pamphlets, the videotaped mime demonstration, the international symbols, and the instructions in the kiosk in French and English, competitors from Oman, Pakistan, Japan, and Korea were helpless. Two new translations were added, then eight more. The problem of names from other cultures, where the first, middle, last name convention is not honored and for which equivalent English spellings are uncertain, was addressed by referring to badges and providing a reply function. A nest of problems with logging in and passwords was uncovered and beheaded.

And then 2,800 people were connected to the system at Yorktown and another 1,000 in Los Angeles. They used the system in their work and tested the robustness of the hardware and software in the face of connections to four different telephone companies and the stress of real loads.

Was the project a success? The system, with its six central computers and three dozen user stations, was never down entirely. Used at least once by 4,000 of the 10,000 Olympians, it carried one to two messages per minute day and night. Competitors received almost 12,000 greetings and good wishes.

This case powerfully illustrates the variety of ways to do UCD. There are many ways to maintain user focus and to obtain empirical data on what does and doesn't work. UCD consists in doing some of them. In table 10.1, almost no two projects did their UCD the same way, and, unlike this example, none used more than one or two techniques.

A Scene from Xerox

In the early 1980s, Xerox's copier business was slipping (Brown 1991). Strong new competition from Japanese manufacturers like Canon was only part of the problem. Service calls and customer complaints were up and rising. Customer interviews and service personnel reports suggested that the machines were not breaking down any more often than before, but that customers were having more difficulty operating them. A user would attempt some job and run into an obstacle—perhaps a paper jam or low toner or, worse, a complete mystery. The user might try to fix the problem—or not—and give up, abandoning the machine in a nonfunctioning condition. The next user would find it "out of order" and call for repair.

To find out why customers were having such difficulty, Lucy Suchman, a Xerox researcher, put a video camera over one of the copiers in the lab building and recorded interactions between people and machine. To the amazement and alarm of executives who later saw the tapes, their own employees had enormous trouble operating the device. In one segment, two renowned computer scientists stand by the machine in obvious frustration, try one thing, then another, reason together, then give up.

Instructions and trouble-shooting procedures for Xerox copiers of the era were contained in packs of flip cards attached to the machine. When the machine failed, a code would light on the machine, and the user was supposed to find the corresponding flip card and follow its directions. As machines added more features, the flip card packs waxed, and the users' temptation to consult them waned. Based on analyses of the videos, the copiers were redesigned. In new versions, the flip cards are gone. Instead, a small computer-driven screen displays instructions related to the function being used and shows icons and diagrams. When something goes wrong, a picture of the machine appears, showing where the problem is and illustrating how to get at it. Typical downtime for a paper jam plunged from twenty-eight minutes to twenty seconds because customers were much more willing and able to fix their own problems.