# Designing the Xerox “Star” User Interface

The Star user interface adheres rigorously to a small set of principles designed to make the system seem friendly by simplifying the human-machine interface.

Reprinted from Byte, issue 4/1982, pp. 242-282.

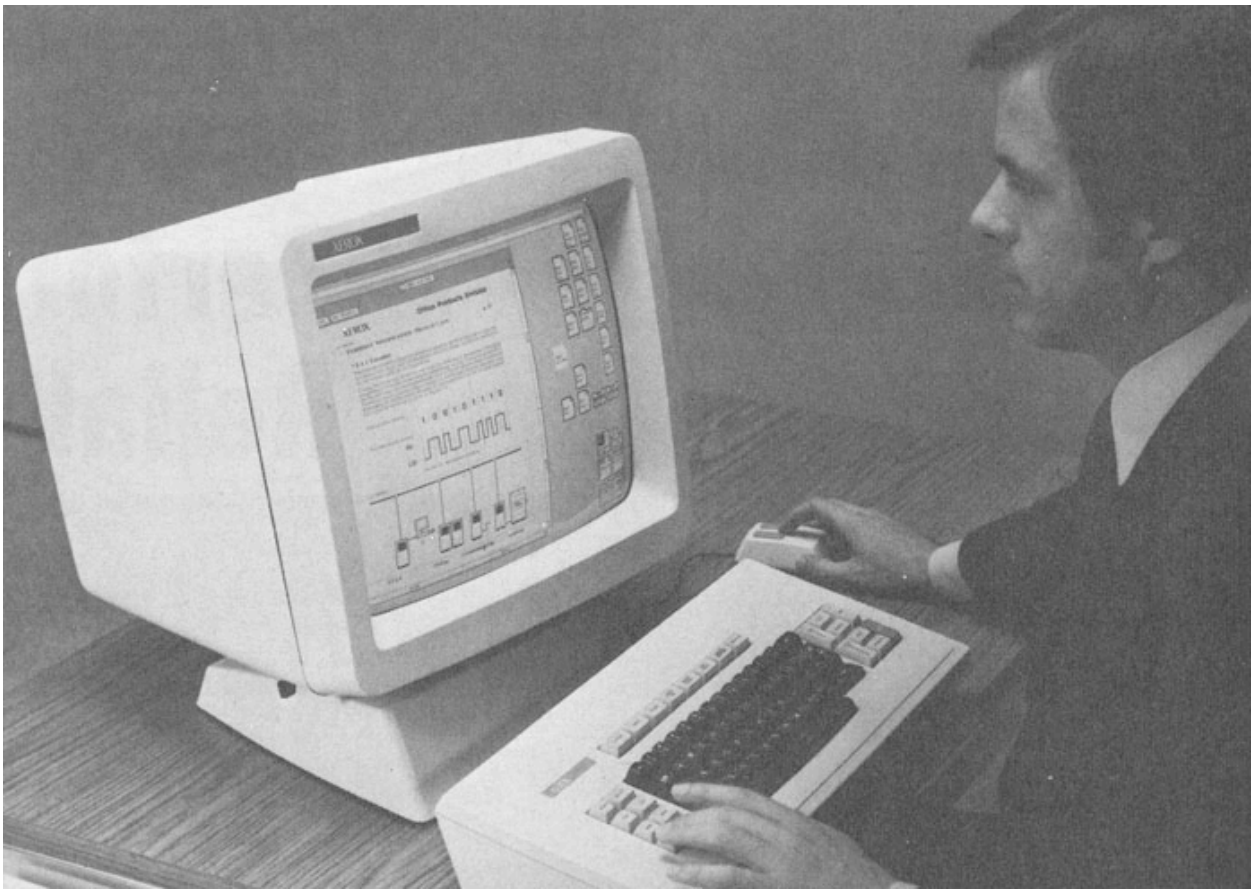

Photo 1: A Star work station showing the processor, display, keyboard and mouse.
In April 1981, Xerox announced the 8010 Star Information System, a new personal computer designed for offices. Consisting of a processor, a large display, a keyboard, and a cursor-control device (see photo 1), it is intended for business professionals who handle information.

Star is a multifunction system combining document creation, data processing, and electronic filing, mailing, and printing. Document creation includes text editing and formatting, graphics editing, mathematical formula editing, and page layout. Data processing deals with homogeneous, relational databases that can be sorted, filtered, and formatted under user control.

Filing is an example of a network service utilizing the Ethernet local-area network (see references 9 and 13). Files may be stored on a work station’s disk, on a file server on the work station’s network, or on a file server on a different network. Mailing permits users of work stations to communicate with one another. Printing utilizes laser-driven raster printers capable of printing both text and graphics.

As Jonathan Seybold has written, “This is a very different product: Different because it truly bridges word processing and typesetting functions; different because it has a broader range of capabilities than anything which has preceded it; and different because it introduces to the commercial market radically new concepts in human engineering.” (See reference 15.)

The Star user interface adheres rigorously to a small set of design principles. These principles make the system seem familiar and friendly, simplify the human-machine interaction, unify the nearly two dozen functional areas of Star, and allow user experience in one area to apply in others. In reference 17, we presented an overview of the features in Star. Here, we describe the principles behind those features and illustrate the principles with examples. This discussion is addressed to the designers of other computer programs and systems – large and small.

**Star Architecture**

Before describing Star’s user interface, several essential aspects of the Star architecture should be pointed out. Without these elements, it would have been impossible to design an interface anything like the present one.

The Star hardware was modeled after the experimental Xerox Alto computer (see reference 19). Like Alto, Star consists of a Xerox-developed, high-bandwidth, MSI (medium-scale integration) processor; local disk storage; a bit-mapped display screen having a 72-dots-per-inch resolution; a pointing device called the “mouse”; and a connection to the Ethernet network. Stars are higher-performance machines than Altos, being about three times as fast, having 512K bytes of main memory (versus 256K bytes on most Altos), 10 or 29 megabytes of disk memory (versus 2.5 megabytes), a 10½- by 13-inch display screen (versus 10½ by 8 inches), and a 10-megabits-per-second Ethernet (versus 3 megabits). Typically, Stars, like Altos, are linked via Ethernets to each other and to shared file, mail, and print servers. Communication servers connect Ethernets to one another either directly or over telephone lines, enabling internetwork communication. (For a detailed description of the Xerox Alto computer, see the September 1981 BYTE article “The Xerox Alto Computer” by Thomas A. Wadlow on page 58.)

The most important ingredient of the user interface is the bit-mapped display screen. Both Star and Alto devote a portion of main memory to the screen: 100K bytes in Star, 50K bytes (usually) in Alto. Every screen dot can be individually turned on or off by setting or resetting the corresponding bit in memory. It should be obvious that this gives both computers an excellent ability to portray visual images. We believe that all impressive office systems of the future will have bit-mapped displays. Memory cost will soon be insignificant enough that they will be feasible even in home computers. Visual communication is effective, and it can’t be exploited without graphics flexibility.

There must be a way to change dots on the screen quickly. Star has a high memory bandwidth, about 90 megahertz (MHz). The entire Star screen is repainted from memory 39 times per second, about a 50-MHz data rate between memory and the screen. This would swamp most computer memories. However, since Star’s memory is double-ported, refreshing the display does not appreciably slow down processor memory access. Star also has separate logic devoted solely to refreshing the display. Finally, special microcode has been written to assist in changing the contents of memory quickly, permitting a variety of screen processing that would not otherwise be practical (see reference 8).

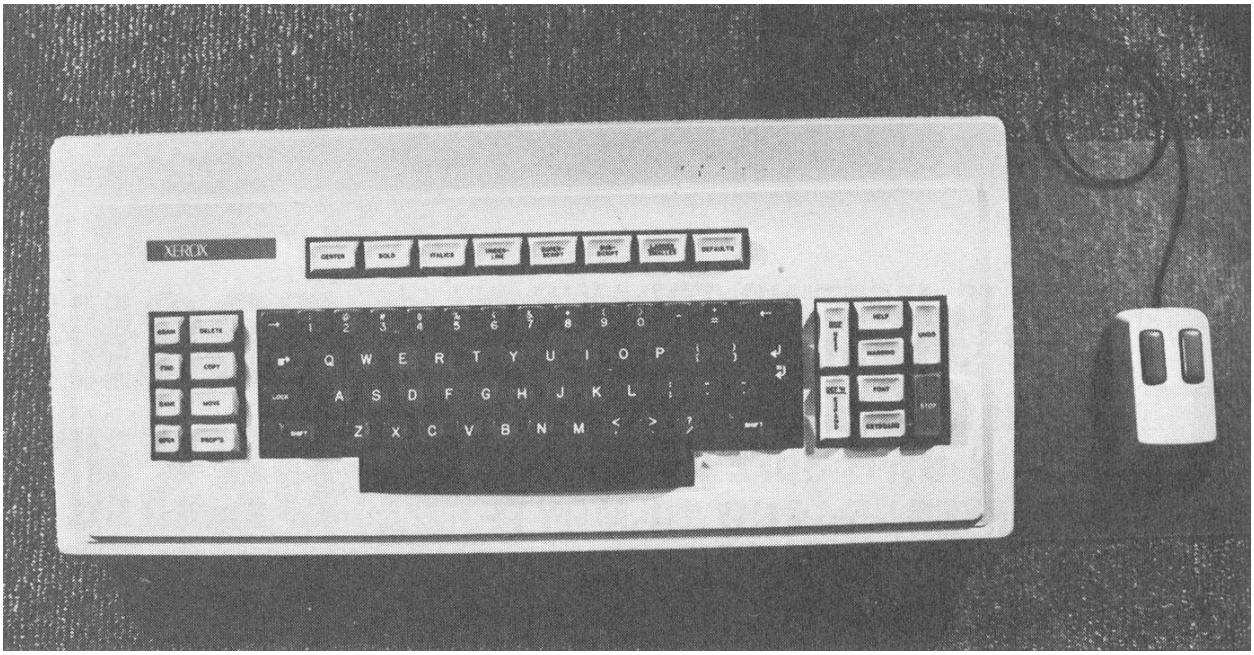


Photo 2: The Star keyboard and mouse. Note the two buttons on top of the mouse.

People need a way to quickly point to items on the screen. Cursor step keys are too slow; nor are they suitable for graphics. Both Star and Alto use a pointing device called the mouse (see photo 2). First developed at Stanford Research Institute (see reference 6), Xerox’s version has a ball on the bottom that turns as the mouse slides over a flat surface such as a table. Electronics sense the ball rotation and guide a cursor on the screen in corresponding motions. The mouse possesses several important attributes:

It is a “Fitts’s law” device. That is, after some practice you can point with a mouse as quickly and easily as you can with the tip of your finger. The limitations on pointing speed are those inherent in the human nervous system (see references 3 and 7).

It stays where it was left when you are not touching it. It doesn’t have to be picked up like a light pen or stylus.

It has buttons on top that can be sensed under program control. The buttons let you point to and interact with objects on the screen in a variety of ways.

Every Star and Alto has its own hard disk for local storage of programs and data. This enhances their personal nature, providing consistent access to information regardless of how many other machines are on the network or what anyone else is doing. Larger programs can be written, using the disk for swapping.

The Ethernet lets both Stars and Altos have a distributed architecture. Each machine is connected to an Ethernet. Other machines on the Ethernet are dedicated as “servers” – machines that are attached to a resource and provide access to that resource.

**Star Design Methodology**

We have learned from Star the importance of formulating the fundamental concepts (the user’s conceptual model) *before* software is written, rather than tacking on a user interface *afterward*. Xerox devoted about thirty work-years to the design of the Star user interface. It was designed *before* the functionality of the system was fully decided. It was even designed *before* the computer hardware was built. We worked for two years before we wrote a single line of actual product software. Jonathan Seybold put it this way, “Most system design efforts start with hardware specifications, follow this with a set of functional specifications for the software, then try to figure out a logical user interface and command structure. The Star project started the other way around: the paramount concern was to define a conceptual model of how the user would relate to the system. Hardware and software followed from this.” (See reference 15.)

In fact, before we even began designing the model, we developed a methodology by which we would do the design. Our methodology report (see reference 10) stated:

One of the most troublesome and least understood aspects of interactive systems is the *user interface*. In the design of user interfaces, we are concerned with several issues: the provision of languages by which users can express their commands to the computer; the design of display representations that show the state of the system to the user; and other more abstract issues that affect the user’s understanding of the system’s behavior. Many of these issues are highly subjective and are therefore often addressed in an *ad hoc* fashion. We believe, however, that more rigorous approaches to user interface design can be developed...

These design methodologies are all unsatisfactory for the same basic reason: they all omit an essential step that must precede the design of any successful user interface, namely task analysis. By this we mean the analysis of the task performed by the user, or users, prior to introducing the proposed computer system. Task analysis involves establishing who the users are, what their goals are in performing the task, what information they use in performing it, what information they generate, and what methods they employ. The descriptions of input and output information should include an analysis of the various objects, or individual types of information entity, employed by the user...

The purpose of task analysis is to simplify the remaining stages in user interface design. The *current task description*, with its breakdown of the information objects and methods presently employed, offers a starting point for the definition of a corresponding set of objects and methods to be provided by the computer system. The idea behind this phase of design is to build up a new

*task environment* for the user, in which he can work to accomplish the same goals as before, surrounded now by a different set of objects, and employing new methods.

Prototyping is another crucial element of the design process. System designers should be prepared to implement the new or difficult concepts and then to *throw away* that code when doing the actual implementation. As Frederick Brooks says, the question “is not *whether* to build a pilot system and throw it away. You *will* do that. The only question is whether to plan in advance to build a throwaway, or to promise to deliver the throwaway to customers... Hence *plan to throw one away; you will, anyhow.”* (See reference 2.) The Alto served as a valuable prototype for Star. Over a thousand Altos were eventually built. Alto users have had several thousand work-years of experience with them over a period of eight years, making Alto perhaps the largest prototyping effort ever. Dozens of experimental programs were written for the Alto by members of the Xerox Palo Alto Research Center. Without the creative ideas of the authors of those systems, Star in its present form would have been impossible. In addition, we ourselves programmed various aspects of the Star design on Alto, but all of it was “throwaway” code. Alto, with its bitmapped display screen, was powerful enough to implement and test our ideas on visual interaction.

Some types of concepts are inherently difficult for people to grasp. Without being too formal about it, our experience before and during the Star design led us to the following classification:

| Easy | Hard |
|---:|---|
| concrete | abstract |
| visible | invisible |
| copying | creating |
| choosing | filling in |
| recognizing | generating |
| editing | programming |
| interactive | batch |

The characteristics on the left were incorporated into the Star user’s conceptual model. The characteristics on the right we attempted to avoid.

**Principles Used**

The following main goals were pursued in designing the Star user interface:

- familiar user’s conceptual model
- seeing and pointing versus remembering and typing
- what you see is what you get
- universal commands
- consistency
- simplicity
- modeless interaction
- user tailorability

We will discuss each of these in turn.

• Familiar User’s Conceptual Model

A *user’s conceptual model* is the set of concepts a person gradually acquires to explain the behavior of a system, whether it be a computer system, a physical system, or a hypothetical system. It is the model developed in the mind of the user that enables that person to understand and interact with the system. The first task for a system designer is to decide what model is preferable for users of the system. This extremely important step is often neglected or done poorly. The Star designers devoted several work-years at the outset of the project discussing and evolving what we considered an appropriate model for an office information system: the metaphor of a physical office.

The designer of a computer system can choose to pursue familiar analogies and metaphors or to introduce entirely new functions requiring new approaches. Each option has advantages and disadvantages. We decided to create electronic counterparts to the physical objects in an office: paper, folders, file cabinets, mail boxes, and so on – an electronic metaphor for the office. We hoped this would make the electronic “world” seem more familiar, less alien, and require less training. (Our initial experiences with users have confirmed this.) We further decided to make the electronic analogues be concrete objects. Documents would be more than file names on a disk; they would also be represented by pictures on the display screen. They would be selected by pointing to them with the mouse and clicking one of the buttons. Once selected, they would be moved, copied, or deleted by pushing the appropriate key. Moving a document became the electronic equivalent of picking up a piece of paper and walking somewhere with it. To file a document, you would move it to a picture of a file drawer, just as you take a physical piece of paper to a physical file cabinet.

The reason that the user’s conceptual model should be decided *first* when designing a system is that the approach adopted *changes the functionality of the system*. An example is electronic mail. Most electronic-mail systems draw a distinction between *messages* and *files* to be sent to other people. Typically, one program sends messages and a different program handles file transfers, each with its own interface. But we observed that offices make no such distinction. Everything arrives through the mail, from one-page memos to books and reports, from intraoffice mail to international mail. Therefore, this became part of Star’s physical-office metaphor. Star users mail documents of any size, from one page to many pages. Messages are short documents, just as in the real world. User actions are the same whether the recipients are in the next office or in another country.

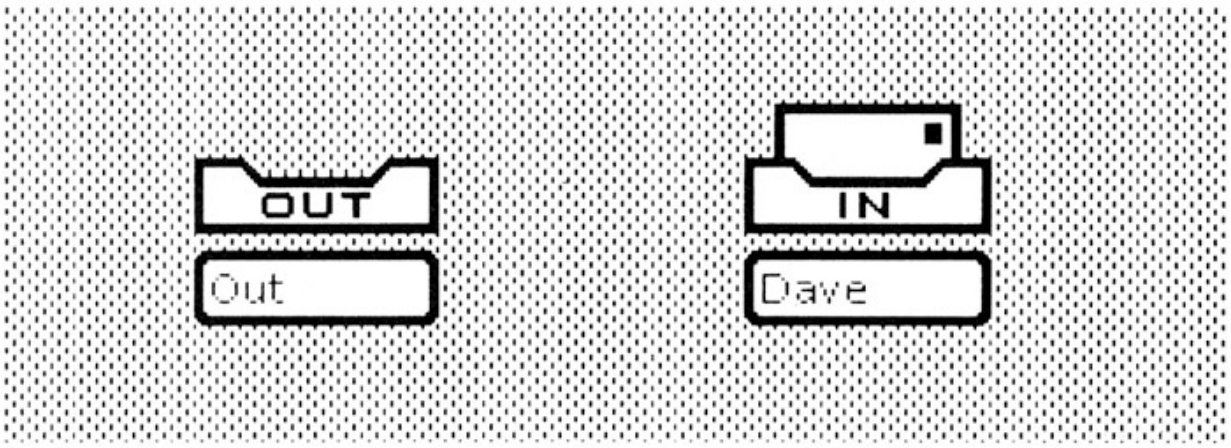


Figure 1: In-basket and out-basket icons. The in-basket contains an envelope indicating that mail has been received. (This figure was taken directly from the Star screen. Therefore, the text appears at screen resolution.)

A physical metaphor can simplify and clarify a system. In addition to eliminating the artificial distinctions of traditional computers, it can eliminate commands by taking advantage of more general concepts. For example, since moving a document on the screen is the equivalent of picking up a piece of paper and walking somewhere with it, there is no “send mail” command. You simply move it to a picture of an out-basket. Nor is there a “receive mail” command. New mail appears in the in-basket as it is received. When new mail is waiting, an envelope appears in the picture of the in-basket (see figure 1). This is a simple, familiar, nontechnical approach to computer mail. And it’s easy once the physical-office metaphor is adopted!

While we want an analogy with the physical world for familiarity, we don’t want to limit ourselves to its capabilities. One of the raisons d’être for Star is that physical objects do not provide people with enough power to manage the increasing complexity of the “information age.” For example, we can take advantage of the computer’s ability to search rapidly by providing a search function for its electronic file drawers, thus helping to solve the long-standing problem of lost files.

• The “Desktop”

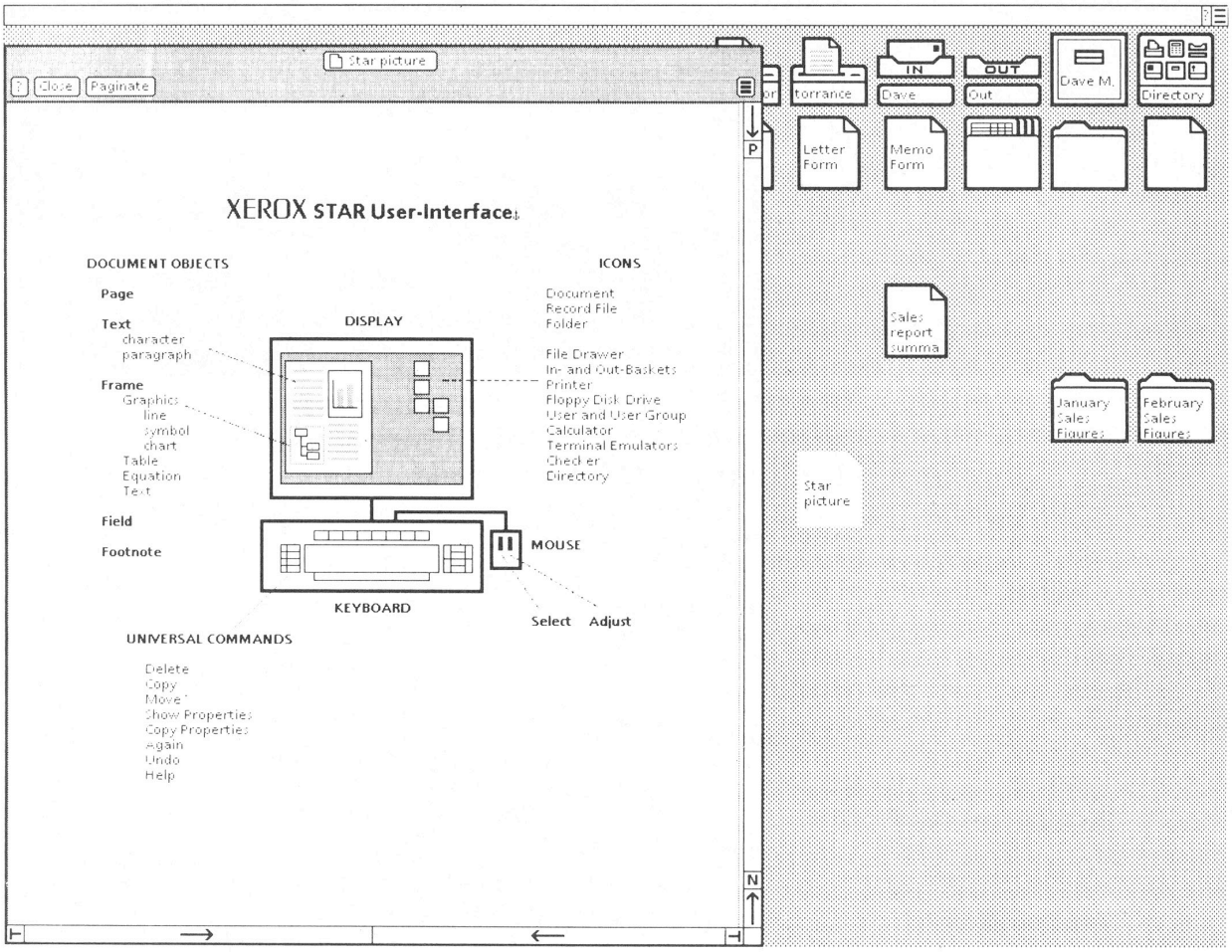


Figure 2: A Desktop as it appears on the Star screen. Several commonly used icons appear across the top of the screen, including documents to serve as “form-pad” sources for letters, memos, and blank paper. An open window displaying a document containing an illustration is also shown.

Every user’s initial view of Star is the “Desktop,” which resembles the top of an office desk, together with surrounding furniture and equipment. It represents your working environment – where your current projects and accessible resources reside. On the screen are displayed pictures of familiar office objects, such as documents, folders, file drawers, in-baskets, and out-baskets. These objects are displayed as small pictures or “icons,” as shown in figure 2.

You can “open” an icon to deal with what it represents. This enables you to read documents, inspect the contents of folders and file drawers, see what mail you have received, etc. When opened, an icon expands into a larger form called a “window,” which displays the icon’s contents. Windows are the principal mechanism for displaying and manipulating information.

The Desktop “surface” is displayed as a distinctive gray pattern. This restful design makes the icons and windows on it stand out crisply, minimizing eyestrain. The surface is organized as an

array of one-inch squares, 14 wide by 11 high. An icon can be placed in any square, giving a maximum of 154 icons. Star centers an icon in its square, making it easy to line up icons neatly. The Desktop always occupies the entire display screen; even when windows appear on the screen, the Desktop continues to exist “beneath” them.

The Desktop is the principal Star technique for realizing the physical-office metaphor. The icons on it are visible, concrete embodiments of the corresponding physical objects. Star users are encouraged to think of the objects on the Desktop in physical terms. Therefore, you can move the icons around to arrange your Desktop as you wish. (Messy Desktops are certainly possible, just as in real life.) Two icons cannot occupy the same space (a basic law of physics). Although moving a document to a Desktop resource such as a printer involves transferring the document icon to the same square as the printer icon, the printer immediately “absorbs” the document, queuing it for printing. You can leave documents on your Desktop indefinitely, just as on a real desk, or you can file them away in folders or file drawers. Our intention and hope is that users will *intuit* things to do with icons, and that those things will indeed be part of the system. This will happen if:

a. Star models the real world accurately enough. Its *similarity* with the office environment preserves your familiar way of working and your existing concepts and knowledge.
b. Sufficient *uniformity* is in the system. Star’s principles and “generic” commands (discussed below) are applied throughout the system, allowing lessons learned in one area to apply to others.

The model of a physical office provides a simple base from which learning can proceed in an incremental fashion. You are not exposed to entirely new concepts all at once. Much of your existing knowledge is embedded in the base.

In a functionally rich system, it is probably not possible to represent everything in terms of a single model. There may need to be more than one model. For example, Star’s records-processing facility cannot use the physical-office model because physical offices have no “records processing” worthy of the name. Therefore, we invented a different model, a record file as a collection of *fields*. A record can be displayed as a row in a table or as filled-in fields in a *form*. Querying is accomplished by filling in a blank example of a record with predicates describing the desired values, which is philosophically similar to Zloof’s “Query-by-Example” (see reference 21).

Of course, the number of different user models in a system must be kept to a minimum. And they should not overlap; a new model should be introduced only when an existing one does not cover the situation.

• Seeing and Pointing

A well-designed system makes everything relevant to a task visible on the screen. It doesn’t hide things under CODE+key combinations or force you to remember conventions. That burdens your memory. During conscious thought, the brain utilizes several levels of memory, the most important being the “short-term memory.” Many studies have analyzed the short-term memory

and its role in thinking. Two conclusions stand out: (1) conscious thought deals with concepts in the short-term memory (see reference 1) and (2) the capacity of the short-term memory is limited (see reference 14). When everything being dealt with in a computer system is visible, the display screen relieves the load on the short-term memory by acting as a sort of “visual cache.” Thinking becomes easier and more productive. A well-designed computer system can actually improve the *quality* of your thinking (see reference 16). In addition, visual communication is often more efficient than linear communication; a picture is worth a thousand words.

A subtle thing happens when everything is visible: *the display becomes reality*. The user model becomes identical with what is on the screen. Objects can be understood purely in terms of their visible characteristics. Actions can be understood in terms of their effects on I the screen. This lets users *conduct experiments* to test, verify, and expand their understanding – the essence of experimental science.

In Star, we have tried to make the objects and actions in the system *visible*. Everything to be dealt with and all commands and effects have a visible representation on the display screen or on the keyboard. You never have to remember that, for example, CODE+Q does something in one context and something different in another context. In fact, our desire to eliminate this possibility led us to abolish the CODE key. (We have yet to see a computer system with a CODE key that doesn’t violate the principle of visibility.) You never invoke a command or push a key and have nothing visible happen. At the very least, a message is posted explaining that the command doesn’t work in this context, or it is not implemented, or there is an error.. It is disastrous to the user’s model when you invoke an action and the system does nothing in response. We have seen people push a key several times in one system or another trying to get a response. They are not sure whether the system has “heard” them or not. Sometimes the system is simply throwing away their keystrokes. Sometimes it is just slow and is queuing the keystrokes; you can imagine the unpredictable behavior that is possible.

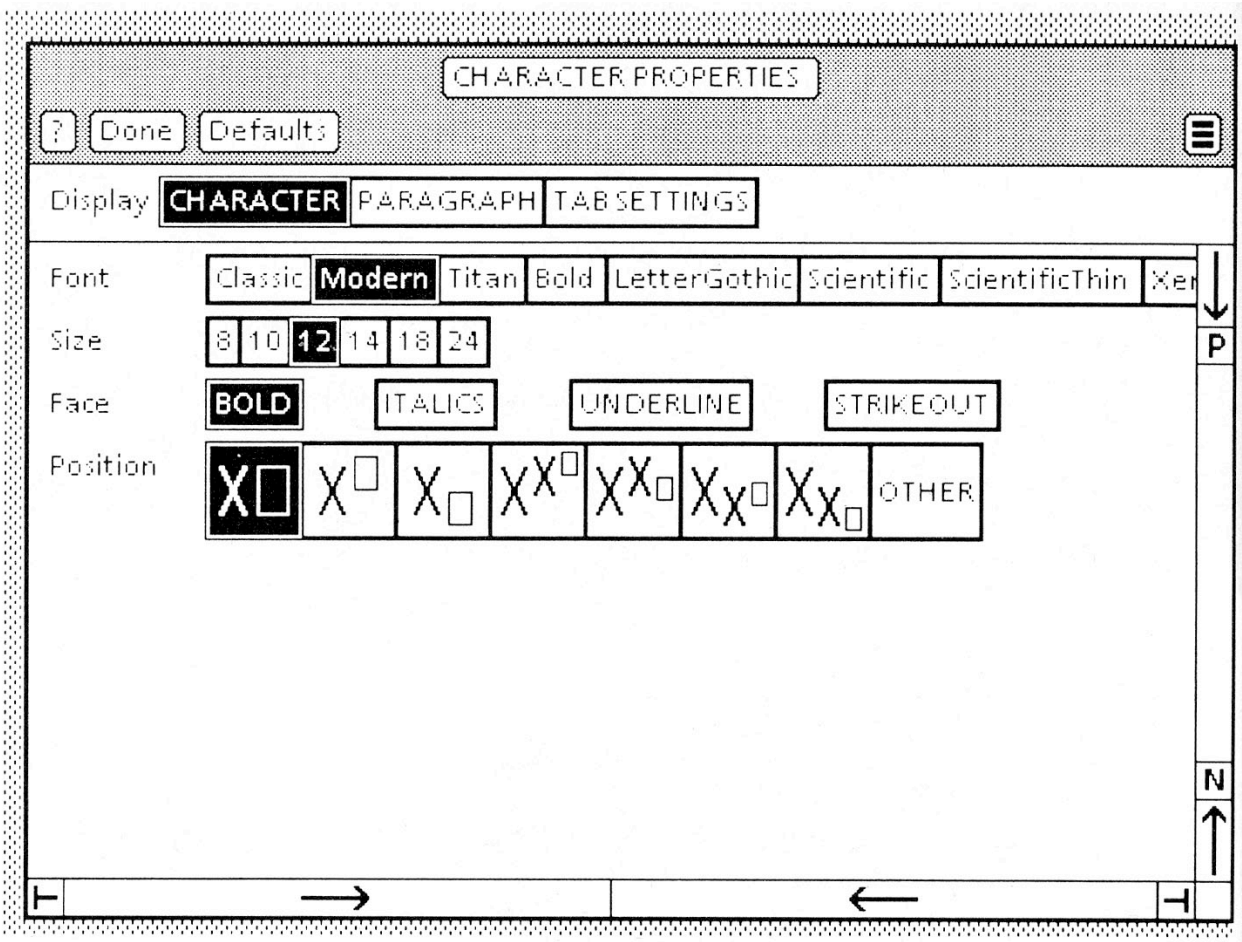


Figure 3: The property sheet for text characters.

We have already mentioned icons and windows as mechanisms for making the concepts in Star visible. Other such mechanisms are Star’s *property and option sheets*. Most objects in Star have properties. A property sheet is a two-dimensional, form-like environment that displays those properties. Figure 3 shows the character property sheet. It appears on the screen whenever you make a text selection and push the PROPERTIES key. It contains such properties as type font and size; bold, italic, underline, and strikeout face; and superscript/subscript positioning. Instead of having to remember the properties of characters, the current settings of those properties, and, worst of all, how to change those properties, property sheets simply *show everything on the screen*. All the options are presented. To change one, you point to it with the mouse and push a button. Properties in effect are displayed in reverse video.

This mechanism is used for *all* properties of *all* objects in the system. Star contains a couple of hundred properties. To keep you from being overwhelmed with information, property sheets display only the properties relevant to the type of object currently selected (e.g., character, paragraph, page, graphic line, formula element, frame, document, or folder). This is an example of “progressive disclosure”: hiding complexity until it is needed. It is also one of the clearest examples of how an emphasis on visibility can reduce the amount of remembering and typing required.

Property sheets may be thought of as an *alternate representation* for objects. The screen shows you the visible characteristics of objects, such as the type font of text characters or the names of icons. Property sheets show you the underlying structure of objects as they make this structure visible and accessible.

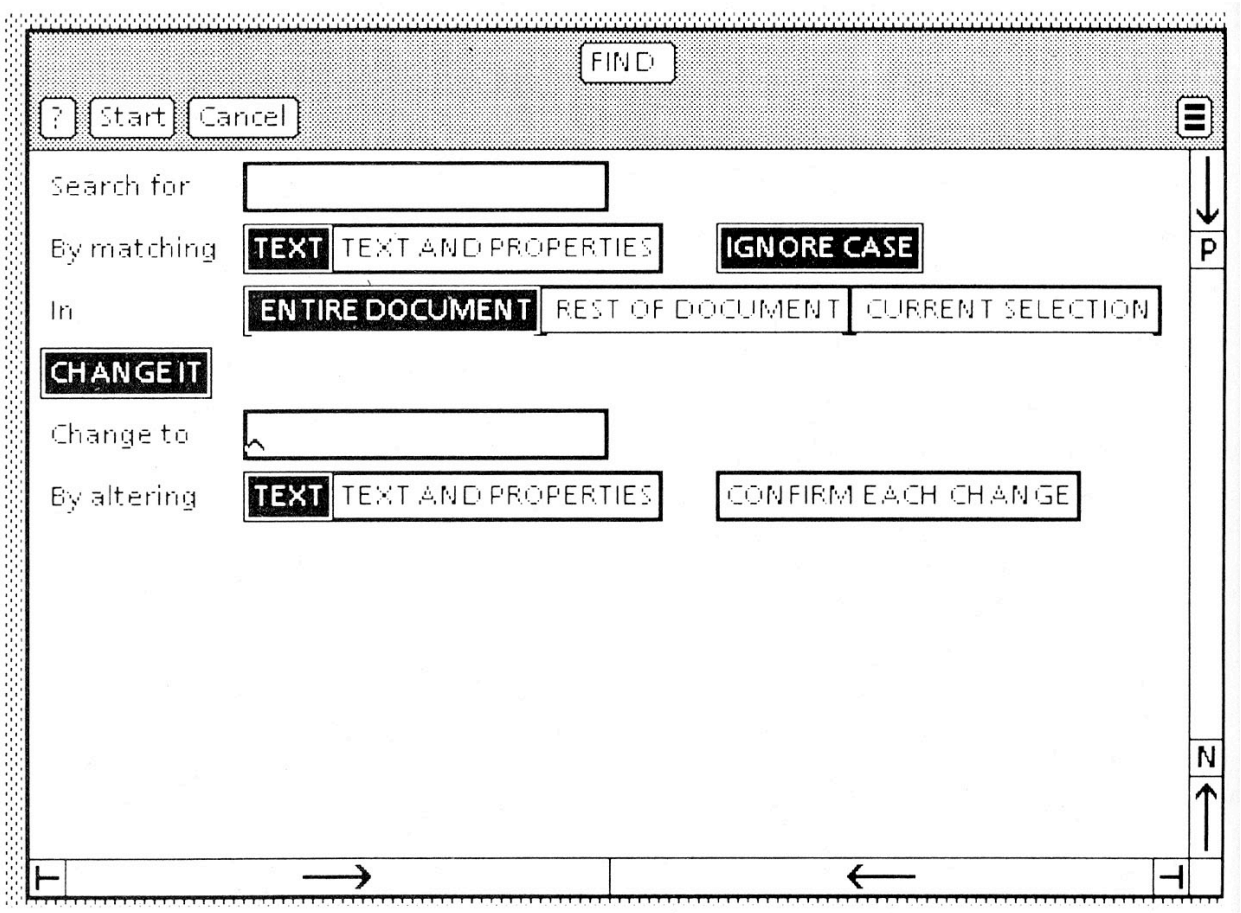


Figure 4: The option sheet for the Find command showing both the Search and Substitute options. The last two lines of options appear only when CHANGE IT is turned on.

Invisibility also plagues the commands in some systems. Commands often have several arguments and options that you must remember with no assistance from the system. Star addresses this problem with *option sheets* (see figure 4), a two-dimensional, form-like environment that displays the arguments to commands. It serves the same function for command arguments that property sheets do for object properties.

• What You See Is What You Get

“What you see is what you get” (or WYSIWYG) refers to the situation in which the display screen portrays an accurate rendition of the printed page. In systems having such capabilities as multiple fonts and variable line spacing, WYSIWYG requires a bit-mapped display because only that has sufficient graphic power to render those characteristics accurately.

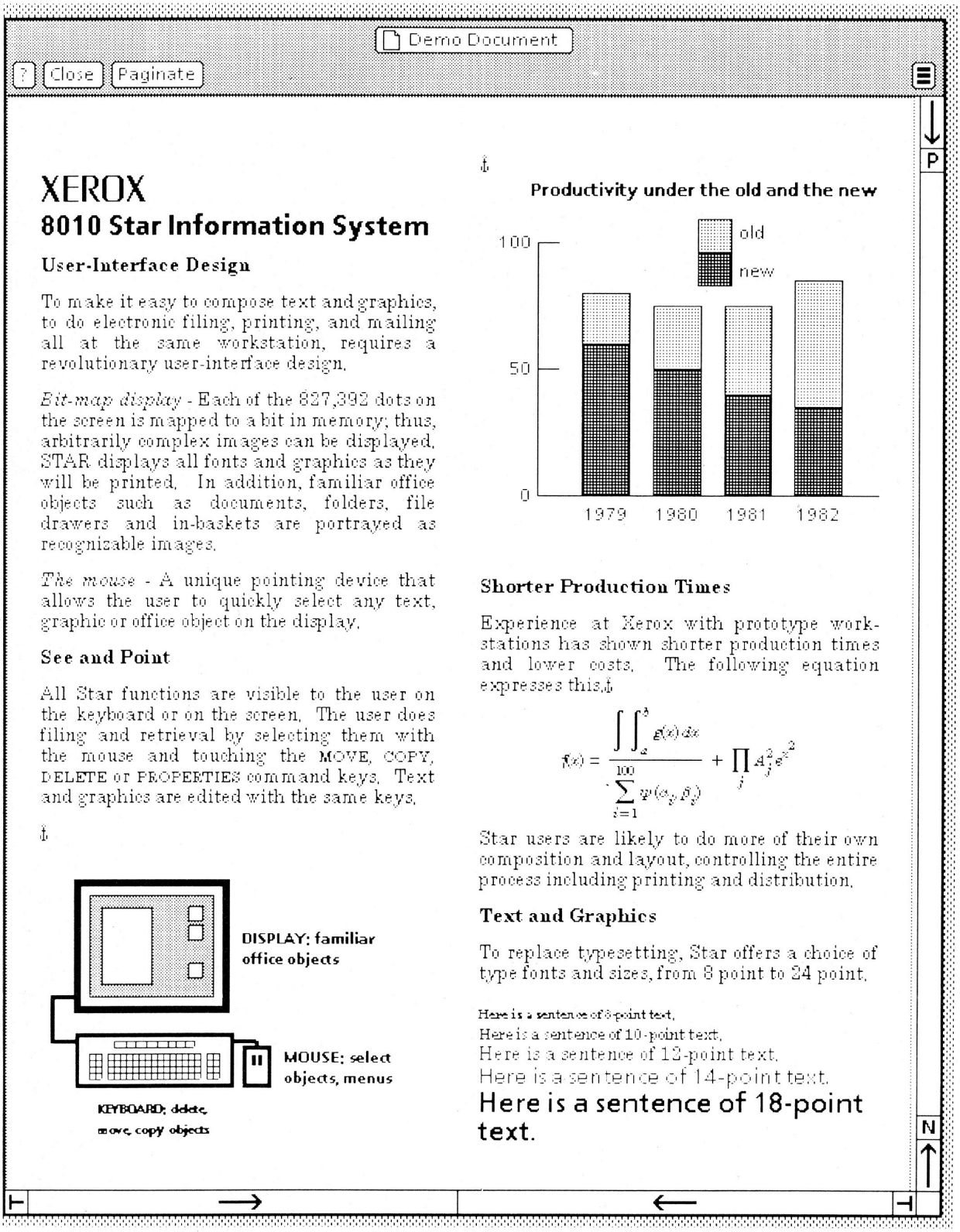


Figure 5: A Star document showing multicolumn text, graphics, and formulas. This is the way the document appears on the screen. It is also the way it will print (at higher resolution, of course).

WYSIWYG is a simplifying technique for document-creation systems. All composition is done *on the screen*. It eliminates the iterations that plague users of document compilers. You can examine the appearance of a page *on the screen* and make changes until it looks right. The printed page will look the same (see figure 5). Anyone who has used a document compiler or post-processor knows how valuable WYSIWYG is. The first powerful WYSIWYG editor was Bravo, an experimental editor developed for Alto at the Xerox Palo Alto Research Center (see reference 12). The text-editor aspects of Star were derived from Bravo.

Trade-offs are involved in WYSIWYG editors, chiefly having to do with the lower resolution of display screens. It is never possible to get an exact representation of a printed page on the screen since most screens have only 50 to 100 dots per inch (72 in Star), while most printers have higher resolution. Completely accurate character positioning is not possible. Nor is it usually possible to represent shape differences for fonts smaller than eight points in size since there are too few dots per character to be recognizable. Even 10-point (“normal” size) fonts may be uncomfortably small on the screen, necessitating a magnified mode for viewing text. WYSIWYG requires very careful design of the screen fonts in order to keep text on the screen readable and attractive. Nevertheless, the increase in productivity made possible by WYSIWYG editors more than outweighs these difficulties.

• Universal Commands

Star has a few commands that can be used throughout the system: MOVE, COPY, DELETE, SHOW PROPERTIES, COPY PROPERTIES, AGAIN, UNDO, and HELP. Each performs the same way regardless of the type of object selected. Thus, we call them “universal” or “generic” commands. For example, you follow the same set of actions to move text in a document and to move a line in an illustration or a document in a folder: select the object, push the MOVE key, and indicate a destination. (HELP and UNDO don’t use a selection.) Each generic command has a key devoted to it on the keyboard.

These commands are far more basic than the commands in other computer systems. They strip away the extraneous application-specific semantics to get at the underlying principles. Star’s generic commands derive from fundamental computer-science concepts because they also underlie operations in programming languages. For example, much program manipulation of data structures involves moving or copying values from one data structure to another. Since Star’s generic commands embody fundamental underlying concepts, they are widely applicable. Each command fills a variety of needs, meaning fewer commands are required. This simplicity is desirable in itself, but it has another subtle advantage: it makes it easy for users to form a model of the system. People can use what they understand. Just as progress in science derives from simple, clear theories, progress in the usability of computers is coming to depend on simple, clear user interfaces.

MOVE is the most powerful command in the system. It is used during text editing to rearrange letters in a word, words in a sentence, sentences in a paragraph, and paragraphs in a document. It is used during graphics editing to move picture elements, such as lines and rectangles, around in an illustration. It is used during formula editing to move mathematical structures, such as summations and integrals, around in an equation. It replaces the conventional “store file” and “retrieve file” commands; you simply move an icon into or out of a file drawer or folder. It

eliminates the “send mail” and “receive mail” commands; you move an icon to an out-basket or from an in-basket. It replaces the “print” command; you move an icon to a printer. And so on. MOVE strips away much of the historical clutter of computer commands. It is more fundamental than the myriad of commands it replaces. It is simultaneously more powerful and simpler.

Much simplification comes from Star’s object-oriented interface. The action of setting properties also replaces a myriad of commands. For example, changing paragraph margins is a command in many systems. In Star, you do it by selecting a paragraph object and setting its MARGINS property. (For more information on object-oriented languages, see the August 1981 BYTE.)

• Consistency

Consistency asserts that mechanisms should be used in the same way wherever they occur. For example, if the left mouse button is used to select a character, the same button should be used to select a graphic line or an icon. Everyone agrees that consistency is an admirable goal. However, it is perhaps the single hardest characteristic of all to achieve in a computer system. In fact, in systems of even moderate complexity, consistency may not be well defined.

A question that has defied consensus in Star is what should happen to a document after it has been printed. Recall that a user prints a document by selecting its icon, invoking MOVE, and designating a printer icon. The printer absorbs the document, queuing it for printing. What happens to that document icon after printing is completed? The two plausible alternatives are:

1. The system deletes the icon.
2. The system does not delete the icon, which leads to several further alternatives:
    a. The system puts the icon back where it came from (i.e., where it was before MOVE was invoked).
    b. The system puts the icon at an arbitrary spot on the Desktop.
    c. The system leaves the icon in the printer. You must move it out of the printer explicitly.

The consistency argument for the first alternative goes as follows: when you move an icon to an out-basket, the system mails it and then deletes it from your Desktop. When you move an icon to a file drawer, the system files it and then deletes it from your Desktop. Therefore, when you move an icon to a printer, the system should print it and then delete it from your Desktop. Function icons should behave consistently with one another.

The consistency argument for the second alternative is: the user’s conceptual model at the Desktop level is the physical-office metaphor. Icons are supposed to behave similarly to their physical counterparts. It makes sense that icons are deleted after they are mailed because after you put a piece of paper in a physical out-basket and the mailperson picks it up, it is gone. However, the physical analogue for printers is the office copier, and there is no notion of deleting a piece of paper when you make a copy of it. Function icons should behave consistently with their physical counterparts.

There is no one right answer here. Both arguments emphasize a dimension of consistency. In this case, the dimensions happen to overlap. We eventually chose alternative 2a for the following reasons:

1. *Model dominance* – The physical metaphor is the stronger model at the Desktop level. Analogy with physical counterparts does form the basis for people's understanding of what icons are and how they behave. Argument 1 advocates an *implicit* model that must be learned; argument 2 advocates an *explicit* model that people already have when they are introduced to the system. Since people do use their existing knowledge when confronted with new situations, the design of the system should be based on that knowledge. This is especially important if people are to be able to *intuit* new uses for the features they have learned.
2. *Pragmatics* – It is dangerous to delete things when users don't expect it. The first time a person labors over a document, gets it just right, prints it, and finds that it has disappeared, that person is going to become *very nervous*, not to mention angry. We also decided to put it back where it came from (2a instead of 2b or 2c) for the pragmatic reason that this involves slightly less work on the user's part.
3. *Seriousness* – When you file or mail an icon, it is not deleted entirely from the system. It still exists in the file drawer or in the recipients' in-baskets. If you want it back, you can move it back out of the file drawer or send a message to one of the recipients asking to have a copy sent back. Deleting after printing, however, is final; if you move a document to a printer and the printer deletes it, that document is gone for good.

One way to get consistency into a system is to adhere to *paradigms* for operations. By applying a successful way of working in one area to other areas, a system acquires a unity that is both apparent and real. Paradigms that Star uses are:

*Editing* – Much of what you do in Star can be thought of as editing. In addition to the conventional text, graphics, and formula editing, you manage your files by *editing filing windows*. You arrange your working environment by *editing your Desktop*. You alter properties by *editing property sheets*. Even programming can be thought of as *editing data structures* (see reference 16).

*Information retrieval* – A lot of power can be gained by applying information-retrieval techniques to information wherever it exists in a system. Star broadens the definition of "database." In addition to the traditional notion as represented by its record files, Star views file drawers as databases of documents, in-baskets as databases of mail, etc. This teaches users to think of information retrieval as a general tool applicable throughout the system.

*Copying* – Star elevates the concept of “copying” to a high level: that of a paradigm for creating. In all the various domains of Star, you *create by copying*. Creating something out of nothing is a difficult task. Everyone has observed that it is easier to modify an existing document or program than to write it originally. Picasso once said, “The most awful thing for a painter is the white canvas... To copy others is necessary.” (See reference 20.) Star makes a serious attempt to alleviate the problem of the “white canvas” by making copying a practical aid to creation. For example, you create new icons by copying existing ones. Graphics are created by copying existing graphic images and modifying them. In a sense, you can even type characters in Star’s 216-character set by “copying” them from keyboard windows (see figure 6).

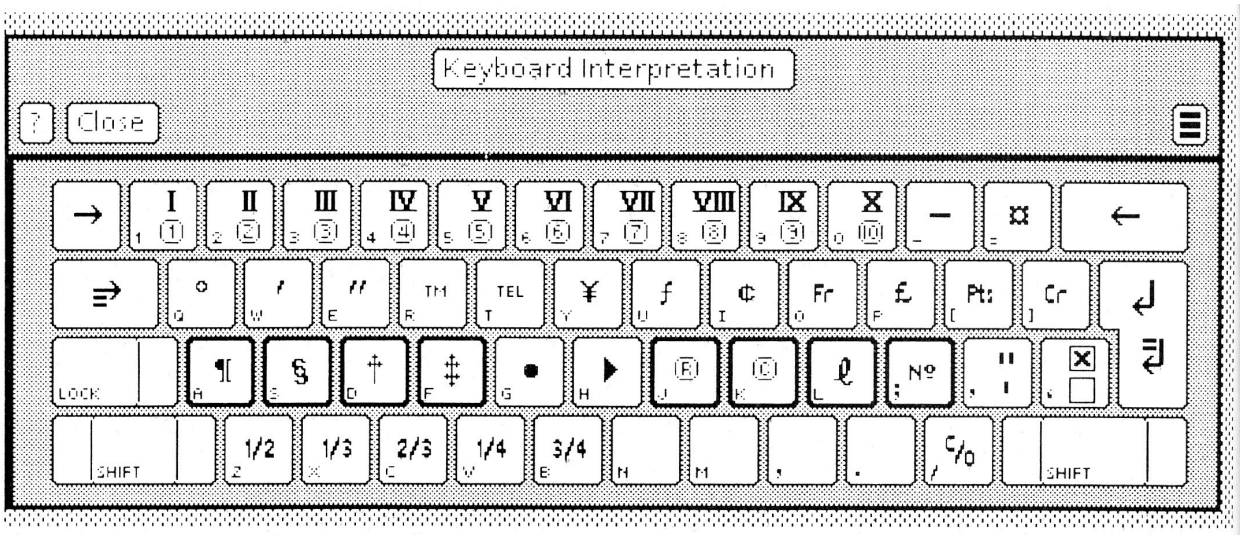


Figure 6: The keyboard-interpretation window serves as the source of characters that may be entered from the keyboard. The character set shown here contains a variety of office symbols.

These paradigms *change the very way you think*. They lead to new habits and models of behavior that are more powerful and productive. They can lead to a *human-machine synergism*.
Star obtains additional consistency by using the class and subclass notions of Simula (see reference 4) and Smalltalk (see reference 11). The clearest example of this is classifying icons at a higher level into *data icons* and *function icons*. Data icons represent objects on which actions are performed. Currently, the three types (i.e., subclasses) of data icons are documents, folders, and record files. Function icons represent objects that perform actions. Function icons are of many types, with more being added as the system evolves: file drawers, in- and out-baskets, printers, floppy-disk drives, calculators, terminal emulators, etc.

In general, anything that can be done to one data icon can be done to all, regardless of its type, size, or location. All data icons can be moved, copied, deleted, filed, mailed, printed, opened, closed, and a variety of other operations applied. Most function icons will accept any data icon; for example, you can move any data icon to an out-basket. This use of the class concept in the user-interface design reduces the artificial distinctions that occur in some systems.

• Simplicity

Simplicity is another principle with which no one can disagree. Obviously, a simple system is better than a complicated one if they have the same capabilities. Unfortunately, the world is never as simple as that. Typically, a trade-off exists between easy novice use and efficient expert use. The two goals are not always compatible. In Star, we have tried to follow Alan Kay’s maxim: “simple things should be simple; complex things should be possible.” To do this, it was sometimes necessary to make common things simple at the expense of uncommon things being harder. Simplicity, like consistency, is not a clear-cut principle.

One way to make a system appear simple is to make it uniform and consistent, as we discussed earlier. Adhering to those principles leads to a *simple user’s model*. Simple models are easier to understand and work with than intricate ones.

Another way to achieve simplicity is to minimize the redundancy in a system. Having two or more ways to do something increases the complexity without increasing the capabilities. The ideal system would have a minimum of powerful commands that obtained all the desired functionality and that did not overlap. That was the motivation for Star’s “generic” commands. But again the world is not so simple. General mechanisms are often inconvenient for high-frequency actions. For example, the SHOW PROPERTIES command is Star’s general mechanism for changing properties, but if is too much of an interruption during typing. Therefore, we added keys to optimize the changing of certain character properties: BOLD, ITALICS, UNDERLINE, SUPERSCRIPT, SUBSCRIPT, LARGER/SMALLER (font), CENTER (paragraph). These significantly speed up typing, but they don’t add any new functionality. In this case, we felt the trade-off was worth it because typing is a frequent activity. “Minimum redundancy” is a good but not absolute guideline.

In general, it is better to introduce new *general* mechanisms by which “experts” can obtain accelerators rather than add a lot of special one-purpose-only features. Star’s mechanisms are discussed below under “User Tailorability.”

Another way to have the system as a whole appear simple is to make each of its parts simple. In particular, the system should avoid overloading the semantics of the parts. Each part should be kept conceptually clean. Sometimes, this may involve a major redesign of the user interface. An example from Star is the mouse, which has been used on the Alto for eight years. Before that, it was used on the NLS system at Stanford Research Institute (see reference 5). All of those mice have three buttons on top. Star has only two. Why did we depart from “tradition”? We observed that the dozens of Alto programs all had different semantics for the mouse buttons. Some used them one way, some another. There was no consistency between systems. Sometimes, there was not even consistency *within* a system. For example, Bravo uses the mouse buttons for selecting text, scrolling windows, and creating and deleting windows, depending on where the cursor is when you push a mouse button. Each of the three buttons has its own meaning in each of the different regions. It is difficult to remember which button does what where.

Thus, we decided to simplify the mouse for Star. Since it is apparently quite a temptation to overload the semantics of the buttons, we eliminated temptation by eliminating buttons. Well then, why didn’t we use a one-button mouse? Here the plot thickens. We did consider and

prototype a one-button mouse interface. One button is sufficient (with a little cleverness) to provide all the functionality needed in a mouse. But when we tested the interface on naive users, as we did with a variety of features, we found that they had a lot of trouble making selections with it.

In fact, we prototyped and tested six different semantics for the mouse buttons: one one-button, four two-button, and a three-button design. We were chagrined to find that while some were better than others, *none of them* was completely easy to use, even though, a priori, it seemed like all of them would work! We then took the most successful features of two of the two-button designs and prototyped and tested them as a seventh design. To our relief, it not only tested better than any of the other six, everyone found it simple and trouble-free to use.

This story has a couple of morals:

> The intuition of designers is error-prone, no matter how good or bad they are.

> The critical parts of a system should be tested on representative users, preferably of the “lowest common denominator” type.

> What is simplest along any one dimension (e.g., number of buttons) is not necessarily conceptually simplest for users; in particular, minimizing the number of keystrokes may not make a system easier to use.

• Modeless Interaction

Larry Tesler defines a *mode* as follows:

A mode of an interactive computer system is a state of the user interface that lasts for a period of time, is not associated with any particular object, and has no role other than to place an interpretation on operator input. (See reference 18.)

Many computer systems use modes because there are too few keys on the keyboard to represent all the available commands. Therefore, the interpretation of the keys depends on the mode or state the system is in. Modes can and do cause trouble by making habitual actions cause unexpected results. If you do not notice what mode the system is in, you may find yourself invoking a sequence of commands quite different from what you had intended.

Our favorite story about modes, probably apocryphal, involves Bravo. In Bravo, the main typing keys are normally interpreted as commands. The “i” key invokes the Insert command, which puts the system in “insert mode.” In insert mode, Bravo interprets keystrokes as letters. The story goes that a person intended to type the word “edit” into his document, but he forgot to enter insert mode first. Bravo interpreted “edit” as the following commands:

| | |
|---|---|
| E(verything) | select everything in the document |
| D(elete) | delete it |
| I(nsert) | enter insert mode |
| t | type a “t” |

The entire contents of the document were replaced by the letter “t.” This makes the point, perhaps too strongly, that modes should be introduced into a user interface with caution, if at all.

Commands in Star take the form of noun-verb. You specify the object of interest (the noun) and then invoke a command to manipulate it (the verb). Specifying an object is called “making a selection.” Star provides powerful selection mechanisms that reduce the number and complexity of commands in the system. Typically, you will exercise more dexterity and judgment in making a selection than in invoking a command. The object (noun) is almost always specified before the action (verb) to be performed. This helps make the command interface modeless; you can change your mind as to which object to affect simply by making a new selection before invoking the command. No “accept” function is needed to terminate or confirm commands since invoking the command is the last step. Inserting text does not even require a command; you simply make a selection and begin typing. The text is placed after the end of the selection.

The noun-verb command form does not by itself imply that a command interface is modeless. Bravo also uses the noun-verb form; yet, it is a highly modal editor (although the latest version of Bravo has drastically reduced its modalness). The difference is that Bravo tries to make one mechanism (the main typing keys) serve more than one function (entering letters and invoking commands). This inevitably leads to confusion. Star avoids the problem by having special keys on the keyboard devoted solely to invoking functions. The main typing keys only enter characters. (This is another example of the simplicity principle: avoid overloading mechanisms with meanings.)

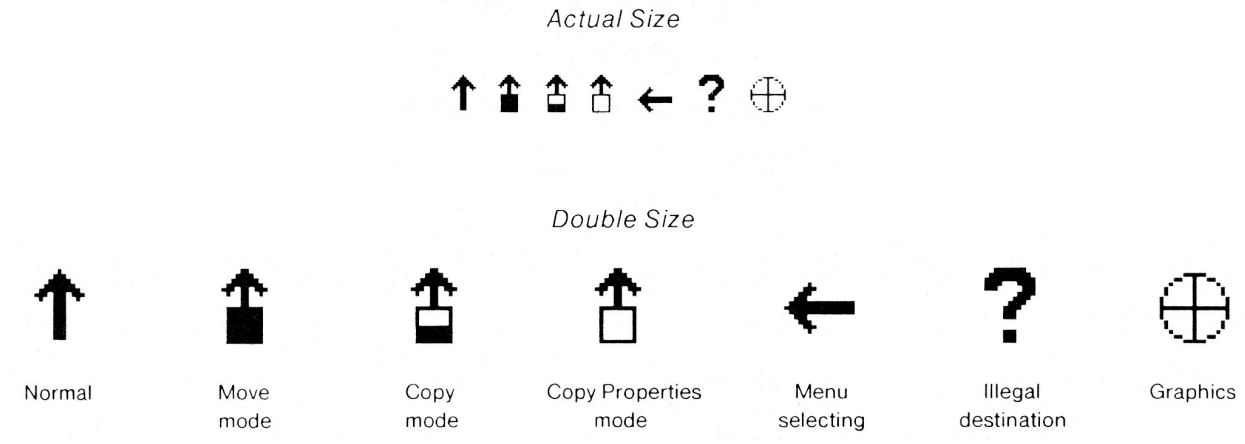


Figure 7: Some of the cursor shapes used by the Star to indicate the state of the system. The cursor is a 16- by 16-bit map that can be changed under program control.

Modes are not necessarily bad. Some modes can be helpful by simplifying the specification of extended commands. For example, Star uses a “field fill-in order specification mode.” In this

mode, you can specify the order in which the NEXT key will step through the fields in the document. Invoking the SET FILL-IN ORDER command puts the system in the mode. Each field you now select is added to the fill-in order. You terminate the mode by pushing the STOP key. Star also utilizes temporary modes as part of the MOVE, COPY, and COPY PROPERTIES commands. For example, to move an object, you select it, push the MOVE key that puts the system in “move mode,” and then select the destination. These modes work for two reasons. First, *they are visible*. Star posts a message in the Message Area at the top of the screen indicating that a mode is in effect. The message remains there for the duration of the mode. Star also changes the shape of the cursor as an additional indication. You can always tell the state of the system by inspection (see figure 7). Second, *the allowable actions are constrained during modes*. The only action that is allowed – except for actions directly related to the mode – is scrolling to another part of the document. This constraint makes it even more apparent that the system is in an unusual state.

• User Tailorability

No matter how general or powerful a system is, it will never satisfy all its potential users. People always want ways to speed up often-performed operations. Yet, everyone is different. The only solution is to design the system with provisions for user extensibility built in. The following mechanisms are provided by Star:

You can tailor the appearance of your system in a variety of ways. The simplest is to choose the icons you want on your Desktop, thus tailoring your working environment. At a more sophisticated level, a work station can be purchased with or without certain functions. For example, not everyone may want the equation facility. Xerox calls this “product factoring.”

You can set up blank documents with text, paragraph, and page layout defaults. For example, you might set up one document with the normal text font being 10-point Classic and another with it being 12-point Modern italic. The documents need not be blank; they may contain fixed text and graphics, and fields for variable fill-in. A typical form might be a business-letter form with address, addressee, salutation, and body fields, each field with its own default text style. Or it might be an accounting form with lines and tables. Or it might be a mail form with To, From, and Subject fields, and a heading tailored to each individual. Whatever the form or document, you can put it on your Desktop and make new instances of it by selecting it and invoking COPY. Thus, each form can act like a “pad of paper” from which new sheets can be “torn off.”

Interesting documents to set up are “transfer sheets,” documents containing a variety of graphics symbols tailored to different applications. For example, you might have a transfer sheet containing buildings in different sizes and shapes, or one devoted to furniture, animals, geometric shapes, flowchart symbols, circuit components, logos, or a hundred other possibilities. Each sheet would make it easier to create a certain type of illustration. Graphics experts could even construct the symbols on the sheets, so that users could create high-quality illustrations without needing as much skill.

You can tailor your filing system by changing the sort order in file drawers and folders. You can also control the filing hierarchy by putting folders inside folders inside folders, to any desired level.

You can tailor your record files by defining any number of “views” or them. Each view consists of a filter, a sort order, and a formatting document. A filter is a set of predicates that produces a subset of the record file. A formatting document is any document that contains fields whose names correspond to those in the record file. Records are always displayed through some formatting document; they have no inherent external representation. Thus, you can set up your own individual subset(s) and appearance(s) for a record file, even if the record file is shared by several users.

You can define “meta operations” by writing programs in the *CUS*tomer *P*rogramming language CUSP. For example, you can further tailor your forms by assigning computation rules expressed in CUSP to fields. Eventually, you will be able to define your own commands by placing CUSP “buttons” into documents.

You can define abbreviations for commonly used terms by means of the abbreviation definition/expansion facility. For example, you might define “sdd” as an abbreviation for “Xerox Systems Development Department.” The expansion can be an entire paragraph, or even multiple paragraphs. This is handy if you create documents out of predefined “boilerplate” paragraphs, as the legal profession does. The expansion can even be an illustration or mathematical formula.

Every user has a unique name used for identification to the system, usually the user’s full name. However, you can define one or more *aliases* by which you are willing to be known, such as your last name only, a shortened form of your name, or a nickname. This lets you personalize your identification to the rest of the network.

Summary

In the 1980s, the most important factors affecting how prevalent computer usage becomes will be reduced cost, increased functionality, improved availability and servicing, and, perhaps most important of all, progress in user-interface design. The first three alone are necessary, but not sufficient for widespread use. Reduced cost will allow people to buy computers, but improved user interfaces will allow people to use computers. In this article, we have presented some principles and techniques that we hope will lead to better user interfaces.

User-interface design is still an art, not a science. Many times during the Star design we were amazed at the depth and subtlety of user-interface issues, even such supposedly straightforward issues as consistency and simplicity. Often there is no one “right” answer. Much of the time there is no scientific evidence to support one alternative over another, just intuition. Almost always

there are trade-offs. Perhaps by the end of the decade, user-interface design will be a more rigorous process. We hope that we have contributed to that progress.

by Dr. David Canfield Smith, Charles Irby, Ralph Kimball, and Bill Verplank; Xerox Corporation, 3333 Coyote Hill Rd., Palo Alto, CA 94304
and Eric Harslem; Xerox Corporation, El Segundo, CA 90245

About the Authors

These five Xerox employees have worked on the Star user interface project for the past five years. Their academic backgrounds are in computer science and psychology.

References

1. Arnheim, Rudolf. *Visual Thinking*. Berkeley: University of California Press, 1971.
2. Brooks, Frederick. *The Mythical Man-Month*. Reading, MA: Addison-Wesley, 1975.
3. Card, Stuart, William English, and Betty Burr. “Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys, and Text Keys for Text Selection on a CRT.” *Ergonomics,* vol. 21, no. 8, 1978, pp. 601-613.
4. Dahl, Ole-Johan and Kristen Nygaard. “SIMULA – An Algol-Based Simulation Language.” *Communications of the ACM,* vol. 9, no. 9, 1966, pp. 671-678.
5. Engelbart, Douglas and William English. “A Research Center for Augmenting Human Intellect.” *Proceedings of the AFIPS 1968 Fall Joint Computer Conference,* vol. 33, 1968, pp. 395-410.
6. English, William, Douglas Engelbart, and M. L. Berman. “Display-Selection Techniques for Text Manipulation.” *IEEE Transactions on Human Factors in Electronics,* vol. HFE-8, no. 1, 1967, pp. 21-31.
7. Fitts, P. M. “The Information Capacity of the Human Motor System in Controlling Amplitude of Movement.” *Journal of Experimental Psychology,* vol. 47, 1954, pp. 381-391.
8. Ingalls, Daniel. “The Smalltalk Graphics Kernel.” BYTE, August 1981, pp. 168-194.
9. Intel, Digital Equipment, and Xerox Corporations. *The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications*. Version 1.0, 1980.
10. Irby, Charles, Linda Bergsteinsson, Thomas Moran, William Newman, and Larry Tesler. *A Methodology for User Interface Design*. Systems Development Division, Xerox Corporation, January 1977.
11. Kay, Alan and the Learning Research Group. *Personal Dynamic Media*. Xerox Palo Alto Research Center Technical Report SSL-76-1, 1976. (A condensed version is in *IEEE Computer,* March 1977, pp. 31-41.)
12. Lampson, Butler. “Bravo Manual.” *Alto User’s Handbook,* Xerox Palo Alto Research Center, 1976 and 1978. (Much of the design of all the implementation of Bravo was done by Charles Simonyi and the skilled programmers in his “software factory.”)
13. Metcalfe, Robert and David Boggs. “Ethernet: Distributed Packet Switching for Local Computer Networks.” *Communications of the ACM,* vol. 19, no. 7, 1976, pp. 395-404.
14. Miller, George. “The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information.” In *The Psychology of Communication,* by G.

Miller, New York: Basic Books, 1967. (An earlier version appeared in *Psychology Review,* vol. 63, no. 2, 1956, pp. 81-97.
15. Seybold, Jonathan. “Xerox’s ‘Star’.” In *The Seybold Report,* Media, PA: Seybold Publications, vol. 10, no. 16, 1981.
16. Smith, David Canfield. *Pygmalion, A Computer Program to Model and Stimulate Creative Thought*. Basel, Switzerland: Birkhauser Verlag, 1977.
17. Smith, David Canfield, Charles Irby, Ralph Kimball, and Eric Harslem. The Star User Interface: An Overview. Submitted to the AFIPS 1982 National Computer Conference.
18. Tesler, Larry. Private communication; but also see his excellent discussion of modes in “The Smalltalk Environment.” BYTE, August 1981, pp. 90-147.
19. Thacker, C. P., E. M. McCreight, B. W. Lampson, R. F. Sproull, and D. R. Boggs, “Alto: A Personal Computer.” In *Computer Structures: Principles and Examples*, edited by D. Siewiorek, C. G. Bell, and A. Newell, New York: McGraw-Hill, 1982.
20. Wertenbaker, Lael. *The World of Picasso*. New York: Time-Life Books, 1967.
21. Zloof, M. M. “Query-by-Example.” *Proceedings of the AFIPS 1975 National Computer Conference,* vol. 44, 1975, pp. 431-438.