NCGA Conference
Computer Graphics 86
May 1986, Anaheim CA

CMU-ITC-080

Helping Developers See the User's Point of View: A Graphic Designer's Perspective

Daniel J Boyarski
Interface Design Consultant
Information Technology Center
Carnegie Mellon University
Pittsburgh PA 15213

ABSTRACT

Members of a development team, often from diverse backgrounds, bring a different perspective on how to design and produce a user interface for a computer system. However, these experts are coming to agree that a good interface is crucial to the success of the system, because to the user, the interface is the system. This paper discusses my experiences as a member of a development team and provides recommendations on how experts from diverse backgrounds can work together effectively to develop a usable and elegant computer system.

INTRODUCTION

More and more developers, designers, and marketing experts have come to accept the fact that user-interface design will be one of the major factors affecting computer purchase and usage in the future. Put simply, to the user, the interface is the system.

Having said this, why is there still the tendency to regard interface design as icing on the cake, when, in fact, it is one of the main ingredients? Why is it that development teams rarely consult with interface designers when a development project begins? Interface designers are brought into the picture when the system has gone through its major design phase and "window dressing" or "icing" is now required to make the system look presentable and friendly. One of the reasons, I believe, is that the work of an interface designer is not understood very well and, therefore, is not fully appreciated.

As a graphic designer, and as a member of an interface design team, I will discuss my role as part of a development team, show examples of our work, and provide recommendations on how experts from diverse backgrounds can work together effectively to develop a usable and elegant computer system.

What is a Graphic Designer?

A graphic designer is a visual communicator of information that is transmitted from a sender to a receiver. A graphic designer gives form to this information using type (letters) and images. The images may be photographs, illustrations, diagrams, or type itself. The information may be a birth announcement, a concert poster, a plane schedule, a book on Australia, or instructions on how to use a computer. The majority of what a graphic designer designs ends up being printed -- that's where the term "graphic" comes from. However, there are other formats for visual communication: road signs, exhibitions, television and film graphics, computer screens, and the like.

What is common to all the work of a graphic designer is the organization, the giving of graphic form, and the transmission of information

from a sender (the client) to a receiver (the audience). To accomplish this, a graphic designer has to wear several hats: that of psychologist, sociologist, artist, linguist, and businessperson. As a problem solver, a designer helps the client understand and define the problem at hand. Only with this definition of the problem -- its function, its audience, its constraints -- can a designer proceed with the process of designing. This is not unlike the process followed by architects, writers, and engineers, all problem solvers.

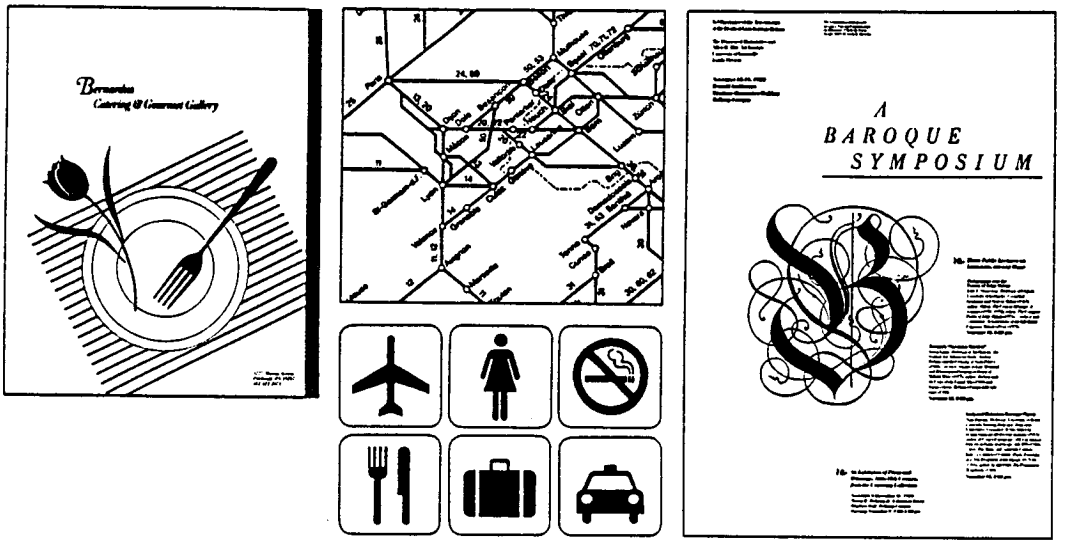

Figure 1. Examples of graphic design work. Clockwise, from far left: menu cover, train route map, poster, and sign symbols for the U.S. Department of Transportation.

The Information Technology Center: Background

In the fall of 1982, Carnegie Mellon University (CMU) and IBM signed a contract whereby CMU would become the first university to completely computerize its campus with a distributed, as opposed to a time-sharing, computer system. IBM was to develop the hardware. CMU established the Information Technology Center (ITC) to develop the software. Though the primary goal was to develop course applications and to use the computer in the classroom, the total system is to be used by everyone on campus: students, faculty, and staff. That meant that the audience for this system would be far from homogeneous, and would run the gamut from completely computer naive to completely computer literate. How could the ITC meet the needs of such a diverse audience?

The Interface Design Team

That was precisely our concern when three of us -- a writer, a graphic designer, and a graduate student in Rhetoric -- began work at the ITC, in mid-1984, on user interface and documentation design. As non-computer experts working in an environment dominated by developers with Ph.D.'s in Computer Science, how could we ensure that the final system would be usable by our constituents, non-experts, who comprise a significant proportion of the campus community? Our team of three was actually part of a larger interface design group at the ITC, which was made up primarily of developers. The work of this group covered the design and implementation of a base editor, upon which course applications could be built, a text editor, a mail and bulletin board system, and Tutor85, an integrated authoring/programming environment.

We stated our goal this way: "The interface for Andrew (the name given to the system) should be elegant, friendly, and transparent. The user, from novice to expert, should be able to work on Andrew with a minimum of instruction, apprehension, and wasted time." We still subscribe to that principle and it continues to guide us as we work on interface design issues.

Process

Since we were brought into the ITC a year and a half into the development of the system, a large part of our work there has been to evaluate the current system and to make suggestions for improving it. We have worked on menus, scrollbars, cursors, icons, status indicators, fonts for display and output, defaults/preferences, the general screen appearance, and documentation. My particular emphasis has been on the visual aspects of the interface, given my background as a visual communicator. For example, the design of a book is very similar to the design of a display screen. There are many levels of information that need to be organized, prioritized, and given clear graphic form. Issues of information heirarchy, legibility, formatting of text and graphics, sequencing, and, yes, even aesthetics, are common to both design projects.

My two colleagues brought their expertise in writing, cognitive psychology, and testing procedures to these same issues. With a similar humanistic background in problem-solving skills, the three of us developed a process of working, very similar to that used by designers and writers. Since communication between user and computer is ultimately what we are dealing with, this process is very appropriate. It looks something like this:

definition of the problem:
    what is it?
    what will it do?
    who is it for?
build, design, or write, according to the definition
review this initial work
refine
test on users (the intended audience)
review test results
refine
(retest, re-review, refine, if necessary)
release

This outline is just that, an outline. The linearity of it at times gives way to referring back to previous steps, for the sake of a clearer problem definition. This definition drives the work being produced, and, in turn, all work produced is evaluated on how it solves the problem. No more, no less.

A Case Study: Cursors

When we started working at the ITC, one basic cursor was in use (figure 2). This same arrow represented the mouse cursor, the text editor cursor, the scrollbar cursor, and the menu cursor. In Andrew, the cursor is the image (object) on the screen that moves when the mouse is moved. I shall limit this discussion to this particular cursor; other cursors now exist, but would complicate the issue.

With testing, it was shown that, especially in the text editor, users had difficulty pointing to a specific spot between characters (letters), to correct a misspelling, for example. The reason was that the arrow cursor had a basic horizontal axis (figure 3), not helpful in making what is basically a vertical move (figure 4).

Thus started a series of arrow cursor options (figure 5) that I designed and released, and were tested on the ITC staff and selected novice users. Their comments were helpful in determining what did and didn't work. I looked for objective comments like, "...this really helps me correct typos easily; I can point directly to a spot between letters," and I tended to place less importance on subjective comments like, "...what is that ugly bent harpoon for?" I should point out that this series of trial cursors was not greeted happily by everyone at the ITC; some developers were disturbed that I was tampering with a "familiar friend."

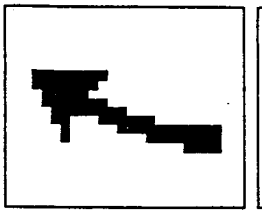

Figure 2.

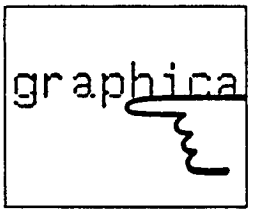

Figure 3.

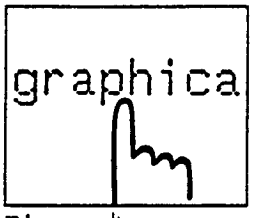

Figure 4.

With much trial and error, the shape of the cursor that is currently implemented (figure 6) is based on a clear understanding of the function of that particular cursor and on the test results. It is also the result of a happy compromise between the functional needs of that cursor and the intuitive needs of the users, who favored a "flowing" arrow instead of a static one, since this cursor represents the motions made with the mouse. Form follows function.

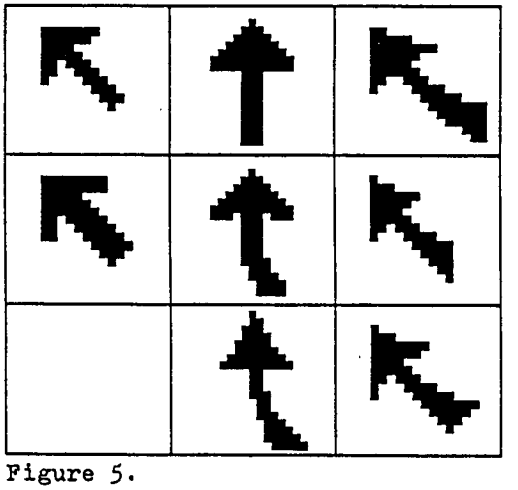

Figure 5.

Through this process, it became clear that a specific function or action requires a specific cursor. So, the action of pointing to a spot between characters in a text requires a specific cursor to aid in that action (figure 7). On the other hand, the action of selecting a menu item is a different action, and, therefore, requires yet another cursor to aid in that action (figure 8). Different actions require different cursors. Different modes sometimes require different cursors, as a visual signal that the user is, in fact, in a new mode.

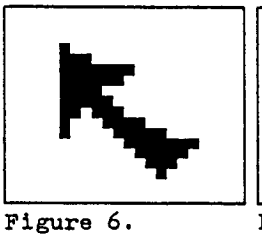

Figure 6.

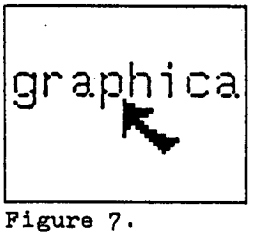

Figure 7.

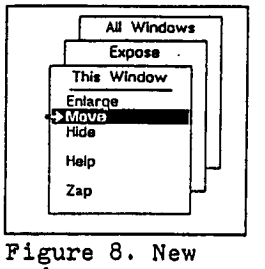

Figure 8. New mode, new cursor.

This case study covers only one cursor in a broad and sophisticated system. Our goal is to standardize all cursors, so that

1. they properly serve their function,
2. they give the user a clear signal of being in a different mode, and
3. they work as a consistent set of cursors.

These issues of semantics (the meaning), syntactics (the form),

and pragmatics (the fitting into a family or set of symbols) are always considered by graphic designers in the design of communication symbols, as illustrated, in figure 1, by the symbols for the U.S. Department of Transportation. These same issues should drive the design of cursors, icons, and symbols for any computer system.

Another Case Study: Console

As Andrew grew in size and complexity, there also grew a need to consolidate various status indicators, error message indicators, as well as clock and date indicators (figure 9). One of the developers took it upon himself to design a console window that grouped together those relevant indicators. As he worked on this project, he realized that different developers (users) wanted to design and/or customize their own consoles. What resulted was a range of consoles that varied in size, complexity, and looks. This was fine for the individual developer and his specific needs, but would have been confusing to users unfamiliar with console. This was the opinion of the smaller user interface team.

So, we sat down with the developer and, together, over a period of a few weeks, designed a basic console that displayed indicators for system load, clock, day and date, printer delivery, mail delivery, system trouble, file server activity, and error and status messages. These were all organized into a fairly compact window, using icons and words. Principles of "form follows function" and "keep it simple, keep it consistent" were constantly stressed by us. With much discussion, trial and error, give and take, we arrived at the current version (figure 10).

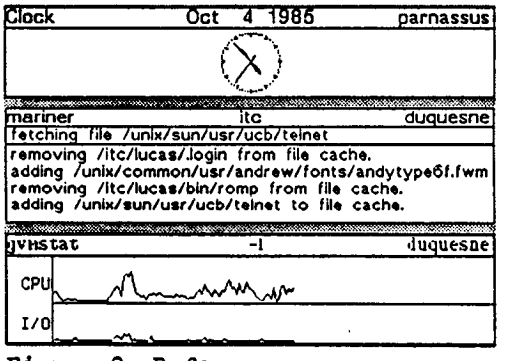

Figure 9. Before.

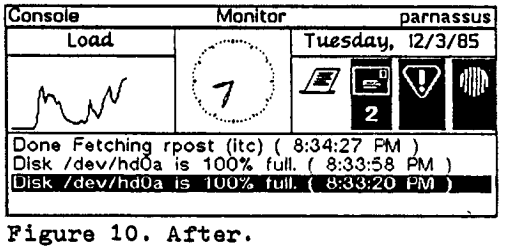

Figure 10. After.

With this console as a model, other developers proceeded with their custom-made consoles. There resulted a family of consoles, all variations on the same theme. Andrew users are now given a choice of consoles to select from, depending upon their needs.

What made this small project a success was the team-work of the user interface designers and the developer. Each one had something to offer, each one had input, and, in the end, the team designed the product, the basic console.

Observations

I believe, as do my two colleagues, that the improvements we have been able to suggest are largely a result of our humanistic problem solving skills. While everyone at CMU is a problem solver, including the ITC system developers, the humanist brings a different set of problem solving skills, a different perspective, and a different focus to a problem and can thus arrive at workable solutions a developer may never think of. Briefly, let me discuss our objectivity and our focus on the audience.

Objectivity

Because we are not actually writing the software, we have less at stake in the current design, and we often don't know how much or how little work would be involved in implementing our recommendations. As a result, we are free to make recommendations based on what we believe is really necessary for the interface, not on constraints like how difficult it would be to implement them.

Focus on the Audience

Training in our individual humanistic disciplines has given us heuristics for attending to and solving human problems rather than machine problems. We maintain this focus on the audience by regularly training and testing users, often computer novices, which helps us from becoming too immersed in the ITC world. Users remind us that simplicity, memorability, and ease of use are very important principles which often get sacrificed to speed, complexity, and multiplicity. Also, in many ways, we are members of the intended audience, and as a faculty member (myself), student, and staff member (the writer), we interact daily with a large segment of the campus community and have a good sense of their needs and expertise.

On the other hand, the majority of the developers do not interact very much with the campus community. They spend eight to ten (often more) hours each day, writing code. Their focus on heavy programming, as a perceived need by Andrew users, does not represent the majority of users at CMU.

More Observations

Our comments and suggestions have been greeted with a range of reactions, from applause to disgust, within the ITC. Progress, I believe, has been made, both in improving the interface for Andrew, and in our interactions with the developers. Early in our tenure at the ITC, we were looked upon with question; we weren't one "of them." Over time, however, we have worked side-by-side with some developers, reaching agreement on issues, and providing the system with the start of an elegant and friendly interface. We have proven, over the course of a year and a half, that we do have an important contribution to make to the development of Andrew. We, too, have realized that working with the developers, instead of springing our ideas on them, generally produces better results. Of course, this works both ways.

It has, however, been a slow process. A great deal of time is often spent in meetings, reviewing and revising a specific program function or some minor detail of the interface. This is due, far too often, to unclear problem definitions, outright resistance, or a lack of communication among the members of the larger interface group.

This brings me to my observations as a member of a large development team:

1. Different backgrounds support and promote different languages. I've heard Computer Science, UNIX, hacker-slang, design, psychology, and some English spoken at meetings and in the halls. Meetings in which these languages surface produce chaos and little progress.

2. Different priorities are the result of different backgrounds. Generally, our smaller interface design team stressed the needs of the user and keeping the interface simple, consistent, and predictable. The developers generally stressed speed, complexity (in more options and customization), and their own needs as programmers.

3. Different problem-solving strategies are also the result of

different backgrounds. Some tend to favor a top-down approach, while others favor the opposite. Top-down starts with defining goals, audience, and individual tasks. Bottom-up, the opposite, has members working on their own, experimenting, "messing around," then sharing the results; the "hey, look what I've come up with" approach. Then, there follows an attempt to pull all these results together into some coherent whole.

Both approaches have merit. History illustrates that both have produced wonderful discoveries as well as solid solutions. Where would we be if Alexander Fleming hadn't experimented with agar plates and accidentally discovered penicillin? On the other hand, the Macintosh interface was designed top-down, with a clear set of guidelines and goals defined and agreed to by the developers from the beginning.[1]

Diversity in the members of a team can be a positive factor and can contribute to a challenging and fruitful working relationship. On the other hand, such diversity can also produce a lack of understanding and appreciation for each other's work, a lack of self-worth as a team member, and, in the long run, an absence of motivation for the project. The challenge here is to work <u>together</u> as partners, to learn from one another, and to build on each other's strengths. It takes a serious commitment to the project on everyone's part, a subduing of individual egos, and a good leader/manager for a project of this, or any, scale to be successful.

Recommendations

1. Construct a development team made up of experts from a variety of disciplines. This includes experts in hardware, software, human factors, professional writing, graphic design, testing, management, and marketing. This diversity is necessary, due to the complexity of such a project.

2. Put this team together at the <u>beginning</u> of the project.

3. Select a leader who will guide the development of the project with objectivity, wisdom, strength, and nurturing. This person should be a <u>manager</u> of resources: human, technical, and financial.

4. Clearly define the project's goals, based on an understanding of the project and its audience. User surveys, when possible, can be an indispensable aid.

5. Clearly define each member's tasks within the larger picture of the project's goals, and conduct <u>regular</u> progress reports. This gives each member a specific place in the team and fosters a mutual respect among everyone. Members should adhere to their assigned tasks, instead of drawing up their own private agendas, ones that may have little to do with the set goals of the project.

6. Draw up a calendar that everyone subscribes to, and <u>stick to it</u>.

7. Promote regular communication among members. Well-organized and planned meetings are one way to achieve this. Electronic bulletin boards and newsletters are other ways.

8. Test, test, test. Keep the user in mind. After all, this is whom you are working to please.

9. Learn from existing models. There is usually a great deal of written material about systems on the market or in development, and reviewing them can help in a team's setting of project goals.

By no means is this an exhaustive list. It is one I put together, based on my experiences and on discussions with other user interface designers.

Lastly

"User-interface design is still an art, not a science. Many times... 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. (Because user-interface design is new), 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." (discussing the Xerox Star user interface)[2]

There is some very fine interface design work being produced today. However, given the growing number of computer systems being developed and manufactured, it is not enough. It is imperative that development teams take advantage of the expertise of individuals from various relevant backgrounds, including interface designers. Such a mixture of skills can only contribute to computer systems that are humane, elegant, easy to use, and, sometimes, even fun!

FOOTNOTES

1. Horn, Bruce; Macintosh software developer. Private lecture.

2. Smith, David Canfield, Eric Harslem, Charles Irby, Ralph Kimball, and Bill Verplank. "Designing the Star User Interface." Byte, April 1982.