# The Anti-Mac Interface

*by Don Gentner and Jakob Nielsen*

**Summary:**
We reverse all of the core design principles behind the Macintosh human interface guidelines to arrive at the characteristics of the Internet desktop.

This paper was **written in 1996**, and was originally published as:
Gentner, D., and Nielsen, J.: The Anti-Mac interface, *Communications of the ACM* **39**, 8 (August 1996), 70-82.

By exploring alternative interfaces that transcend the principles behind conventional graphical interfaces, a human-computer interface emerges that is based on language, a richer representation of objects, expert users, and shared control.

At recent user interface conferences, several speakers have lamented that the human interface is stuck. We seem to have settled on the WIMP (windows, icons, menus, pointer) model, and there is very little real innovation in interface design anymore.

Physicists and mathematicians often stretch their imaginations by considering what the world would be like if some of their basic assumptions and principles were violated (for example, see [1]). This has led to new concepts such as non-Euclidean geometry, positrons, antimatter, and antigravity. At the least, violating basic assumptions is a useful mental exercise, but a surprising number of the resulting concepts have provided useful descriptions of the real world.

In this article, we explore the types of interfaces that could result if we violate each of the Macintosh human interface design principles. We focus on the Macintosh interface because it is a prime example of the current interface paradigm, and Apple Computer has published an explicit list of Macintosh human interface design principles [2]. These principles have not significantly changed since the introduction of the Macintosh; style guides for other popular graphical interfaces, such as Motif, OPEN LOOK, and Windows [16, 18, 22], list a very similar set of principles as the basis for their interfaces.

We should state at the outset that we are devoted fans of the Macintosh human interface and frequent users of Macintosh computers. Our purpose is not to argue that the Macintosh human interface guidelines are bad principles, but rather to explore alternative approaches to computer interfaces. The Anti-Mac interface is not intended to be hostile to the Macintosh, only different. In fact, human interface designers at Apple and elsewhere have already incorporated some of the Anti-Mac features into the Macintosh desktop and applications. The Macintosh was designed to be "the computer for the rest of us" and succeeded well enough that it became, as Alan Kay once said, "the first personal computer good enough to be criticized." This article should be taken in the same spirit.

The Macintosh was designed under a number of constraints, including:

- It needed to sell to "naive users," that is, users without any previous computer experience.
- It was targeted at a narrow range of applications (mostly office work, though entertainment and multimedia applications have been added later in ways that sometimes break slightly with the standard interface).
- It controlled relatively weak computational resources (originally a non-networked computer with 128KB RAM, a 400KB storage device, and a dot-matrix printer).
- It was supported by highly impoverished communication channels between the user and the computer (initially a small black-and-white screen with poor audio output, no audio input, and no other sensors than the keyboard and a one-button mouse).
- It was a standalone machine that at most was connected to a printer.

These constraints have all been relaxed somewhat during the 12 years since the introduction of the Macintosh, but we will explore what might happen if they were to be eliminated completely.

# The Macintosh Human Interface Design Principles

According to the Macintosh guidelines [2], the design of human interfaces for Macintosh system software and applications is based on a number of fundamental principles of human-computer interaction. These principles have led to excellent graphical interfaces, but we wonder: How do these principles limit the computer-human interface? What types of interfaces would result from violating these principles?
We will address these two questions for each of the Macintosh human interface design principles. (The Macintosh design principles and their corresponding Anti-Mac principles are summarized in Table 1.)

| **Mac** | **Anti-Mac** |
|---|---|
| Metaphors | Reality |
| Direct Manipulation | Delegation |
| See and Point | Describe and Command |
| Consistency | Diversity |
| WYSIWYG | Represent Meaning |
| User Control | Shared Control |
| Feedback and Dialog | System Handles Details |
| Forgiveness | Model User Actions |
| Aesthetic Integrity | Graphic Variety |
| Modelessness | Richer Cues |

**Table 1.** *The Mac and Anti-Mac design principles.*

## Metaphors

The first Macintosh principle states that the interface should be based on metaphors

with the familiar noncomputer world around us. In the Macintosh interface, computer files are represented as documents in paper folders that are placed on a desktop. Files are deleted by dragging them to the trash can. Many recent interfaces have tried to overcome the limitations of the desktop metaphor by extending it to some other room or building metaphor (e.g., Bob or Magic Cap, Figure 1) or to a village metaphor (e.g., eWorld). These 3D designs try to emulate virtual reality on a flat screen but often seem to introduce a level of clunky indirectness in achieving common user goals. They are navigationally cumbersome, asking users to go to the "other end of town" to pick up their email from the Post Office, and interactionally cumbersome, overloading users with additional windows and other interface elements necessitated by the metaphor but not by the user's task.

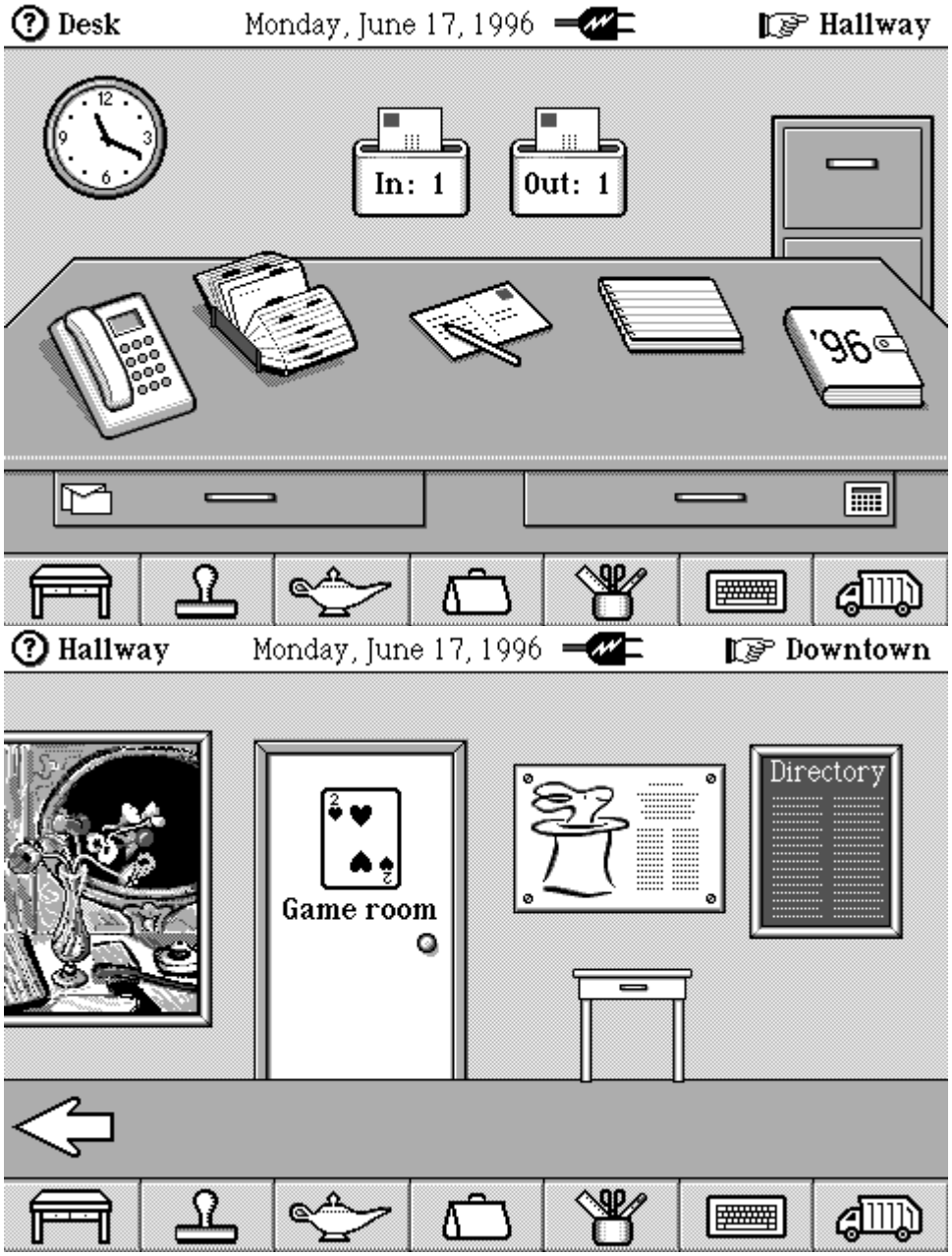


**Figure 1.** *The Magic Cap interface is based on literal metaphors for desktops, buildings, and villages.*

Although the use of metaphor may ease learning for the computer novice, it can also cripple the interface with irrelevant limitations and blind the designer to new paradigms more appropriate for a computer-based application. The designers of the Phelps farm tractor in 1901 based their interface on a metaphor with the interface for the familiar horse: farmers used reins to control the tractor. The tractor was steered

by pulling on the appropriate rein, both reins were loosened to go forward and pulled back to stop, and pulling back harder on the reins caused the tractor to back up [5]. It's clear in hindsight that this was a dead end, and automobiles have developed their own user interfaces without metaphors based on earlier technologies. Nonetheless, people today are designing information-retrieval interfaces based on metaphors with books, even though young folks spend more time flipping television channels and playing video games than they do turning the pages of books.

The three classic problems with metaphors [9] are:

- The target domain has features not in the source domain (e.g., telling the user that "a word processor is like a typewriter" would not lead the user to look for the replace command).
- The source domain has features not in the target domain (a typewriter has the ability to mark up any form you receive in the mail, but a user trying to do that on current computer systems will normally fail).
- Some features exist in both domains but work very differently (the treatment of white space representing space characters, tabs, and line feeds is very different on typewriters and in word processors). Therefore, users may have trouble seeing beyond the metaphor to use the system in the ways it was intended. It is possible to design interfaces to map very closely to metaphors (e.g., Bob Mack's NOSE-no-surprise editor-really did act like a typewriter in all respects), but those interfaces will often be very low-powered and suited mainly for walk-up-and-use situations.

As an example of the limitations of the desktop metaphor, consider the trash can on the Macintosh desktop. It is a fine metaphor for the wastebasket in an office. In both cases we dispose of objects by putting them in the trash can and we're able to retrieve documents we had thrown out until the trash is emptied. However, the limits of the single-trash-can metaphor has led to a system that fails to meet the user's needs and causes confusion by masking the realities of the implementation. In the underlying implementation, there are separate trash containers for each volume, such as a hard disk or a floppy disk, but to avoid the confusion of multiple trash cans in the interface, the contents of the trash containers for all mounted volumes are combined and shown as a single desktop trash can. Because there is only one trash can in the metaphor, if the user empties the trash to create room on a floppy disk, the trash contents on both the floppy and the hard disk are deleted, even though there was no need to delete files from the hard disk's trash can. The desktop metaphor has enforced a limitation on the interface that does not serve the user's real needs. An alternative approach would be to simply cross out files and have them disappear (perhaps to a temporary limbo before being permanently discarded). This approach avoids the problems caused by the trash can metaphor, even though it does not have an analogy in the real world.

Metaphors not only constrain and mislead users, they can also limit designers' ability to invent more powerful interface mechanisms. For example, we are as guilty as anybody for using the tired book metaphor: When we recently designed the user interface to 300MB worth of Sun's online documentation, we used a book metaphor to unify the icons, the vocabulary, and the hierarchical structure of the user's navigation. Our excuse was that the interface will display information originally written as separate printed manuals and that it would have been confusing to use a free-form hyperspace model to represent this text. There is no doubt, however, that the book metaphor prevented us from introducing desirable features like the ability

to reorder the chapters according to their relevance scores after a search.

The desktop metaphor assumes we save training time by taking advantage of the time that users have already invested in learning to operate the traditional office with its paper documents and filing cabinets. But the next generation of users will make their learning investments with computers, and it is counterproductive to give them interfaces based on awkward imitations of obsolete technologies. Instead, we need to develop new interface paradigms based on the structure of computer systems and the tasks users really have to perform, rather than paradigms that enshrine outmoded technology. The way to advance the interface is not to develop ever-more-faithful imitations of the desktop, but instead to escape the limitations of the desktop especially as computers themselves become ubiquitous [21] and are used away from the desk.

## Direct Manipulation

Using direct manipulation, users interact directly with objects in the interface [17]. The archetypal example is to move a file from one directory to another by opening the original folder and using the mouse pointer to drag the file icon to the destination folder. This procedure works well for simple actions with a small number of objects, but as the number of actions or objects increases, direct manipulation quickly becomes repetitive drudgery. The dark side of a direct manipulation interface is that you have to directly manipulate everything. Instead of an executive who gives high-level instructions, the user is reduced to an assembly line worker who must carry out the same task over and over.

Direct manipulation also means that users must always operate at the atomic level. They cannot group a related series of basic actions into one high-level action or use conditionals. Suppose we have a group of images and want to convert all of the PICT files into icons (see Figure 2).


**Figure 2.** *We wish to convert all the PICT files*

> *into icons. This task involves several steps and would be very difficult to accomplish with direct manipulation. A simple scripting language provides a natural specification of the task.*

If the conversion requires several steps, this will be a very tedious process with direct manipulation, but a simple scripting language provides a natural means for specifying this task. Direct manipulation also limits the precision of our actions to the precision achieved with eye-hand-mouse coordination. Language and mathematics can be more precise ("Place the bottom of the triangle level with the middle of the circle") and more dynamic ("Maintain the height of this histogram bar at 37% of that of the bar to its left"). Finally, direct manipulation requires the user to be involved in every action, but sometimes the user may not know what to do. For example, as applications become more complex and involve many files spread throughout the computer, installation and removal of applications exceeds the understanding of most users. Rather than directly draging files to the proper places, most users would prefer pressing a single button on an installation program and have the computer move the files to the appropriate places. Indeed, install and uninstall programs have become an essential adjunct of complex software packages.

## See-and-Point

The see-and-point principle states that users interact with the computer by pointing at the objects they can see on the screen. It's as if we have thrown away a million years of evolution, lost our facility with expressive language, and been reduced to pointing at objects in the immediate environment. Mouse buttons and modifier keys give us a vocabulary equivalent to a few different grunts. We have lost all the power of language, and can no longer talk about objects that are not immediately visible (all files more than one week old), objects that don't exist yet (future messages from my boss), or unknown objects (any guides to restaurants in Boston).

If we want to order food in a country where we don't know the language at all, we're forced to go into the kitchen and use a see-and-point interface. With a little understanding of the language, we can point at menus to select our dinner from the dining room. But language allows us to discuss exactly what we would like to eat with the waiter or chef. Similarly, computer interfaces must evolve to let us utilize more of the power of language. Adding language to the interface allows us to use a rich vocabulary and gives us basic linguistic structures such as conditionals. Language lets us refer to objects that are not immediately visible. For example, we could say something like "Notify me if there is a new message from Emily." Note we are not advocating an interface is based solely on language. Neither does the interface have to understand full natural language. Real expressive power comes from the combination of language, examples, and pointing.

## Consistency

Consistency is one of those principles that sounds like a great idea when you first come across it, but it is very difficult to apply the principle in any real situation where there is a wide array of conflicting things with which you can be consistent [8].

The basic advantage of consistency is the hope that learning will be reduced if objects with a similar function always look and behave the same. People will

recognize applications more easily if they all have similar icons. Yet in the real world, people have no difficulty switching between ballpoint pens and fibertip pens even though they look somewhat different and have different controls. They are similar enough that they are both recognized as pens, whereas their varying appearances provide pleasure and clues to their slightly different functionality. It is the rich and fine-grained representation of objects in the real world that allows pens or books to have a wide variety of appearances and still be easily recognizable. As representations of objects in the computer interface become richer and more fine-grained, the need for complete consistency will drop.

Note that consistency is not symmetrical. Whereas we argue that objects with similar functions need not have consistent appearances or controls, it is still important objects with similar appearances have similar behavior and functions. Except as a joke, a pen that looks like a tennis shoe will not be very useful.

## WYSIWYG

What You See Is What You Get (WYSIWYG) means your document, as it appears on the screen, accurately reflects what it will look like when it is printed. This is certainly a big improvement over earlier computer-based text formatting systems such as troff, in which there was no obvious relation between the document's appearance on the screen and what was produced by the printer, and it took many trials to get a new document to print properly.

The problem with WYSIWYG is that it is usually equivalent to WYSIATI (What You See Is All There Is). A document has a rich semantic structure that is often poorly captured by its appearance on a screen or printed page. For example, a word may be printed in italic font for emphasis, as part of a book title, or as part of a quotation, but the specific meaning is lost if it is represented only by the fact that the characters are italicized. A WYSIWYG document shows only the final printed representation; it does not capture the user's intentions.

Other text representations, such as Standard Generalized Markup Language (SGML), preserve the semantic meaning inherent in the text and have rules for converting the text and semantics into their appearance on the printed page. For example, a string of text may be labeled as a book title in SGML. In one situation that text might be printed in an italic font; in another case it might be printed in a bold font, and in a third case it might be accessed by a bibliographic retrieval program.

WYSIWYG assumes there is only one useful representation of the information: that of the final printed report. Although we are not arguing against a print preview function, even when the goal is to produce a printed document, it may be useful to have a different representation when preparing the document. For example, we may want to see formatting symbols or margin outlines, or it may be useful to see index terms assembled in the margin while we are composing.

In some sense, WYSIWYG is equivalent to the horselike tractor we discussed earlier in connection with metaphors. WYSIWYG assumes people want paper-style reports: Information might be produced (and even consumed) on screen, but it should still be structured the same way it was on paper. These days, however, who has time to read all the documents we get? In reality, people rarely read information from beginning to end the way they would have read a report in the good old days where even busy managers got at most a single report per day. Instead, electronic

information should be modularized and presented in ways that encourage people to read as little as possible by popping information of interest to the specific user to the top, while enabling each user to link to backup information as needed. Object-oriented authoring and reading would allow the same piece of information to be presented at multiple levels and possibly from different perspectives.

We should also mention that not all users can see. For blind users, information that is encoded with SGML-like rich intentional attributes can be represented in alternative formats (e.g., sound) in a much more usable fashion than is possible with simple screen readers that are limited to reading aloud from the pixels on the screen without knowing what they mean.

## User Control

It has become almost a religious crusade among WIMP advocates that the user should be in control—the user, not the computer, should initiate and control actions. The negative side of user control is that the user has to be in control. There are many situations where we do not want to be in control: flying an airliner, for example, or watching our toast in the morning to see that it doesn't burn. Many activities in life are either so difficult or so boring and repetitive that we would like to delegate them to other people or to machines. Similarly, on the computer, there are many activities that we either do not want to control or do not know how to control. This is prime territory for agents and daemons, computer processes that tirelessly stand guard, take care of routine tasks, or have the knowledge that we lack to handle complex tasks.

Most computer programmers gave up complete control some time ago when they stopped writing in machine language and let assemblers, compilers, and interpreters worry about all the little details. And these days, few users still specify the exact routing for their email messages. We've learned that it's not worth the trouble and that computers can often do a better job than we can. Computers and people differ widely in their expertise. For example, people are notoriously bad at vigilance tasks, so it makes sense to let the computer take control of periodically saving our word processor documents, backing up our hard drives, and reminding us of our meetings.

Even if it were desirable, full control is becoming impossible with networked computers. User control assumes you are the only actor in the system, but these days, millions of people are on the Internet and can change the system behind your back. Indeed, it is one of the benefits of the Internet that new resources appear without any work on your part. It would be possible to conceptualize the Internet as a groupware application with 40 million users, but in reality, it is better thought of as a system without any centralized control in which things just happen without other users' asking you what you would like.

## Feedback and Dialog

This principle states that the computer interface should provide the user with clear and immediate feedback on any actions initiated by the user. It is closely connected with the previous principle of user control: If the user is required to be in control of all the details of an action, then the user's needs detailed feedback. But if a sequence of activities can be delegated to an agent or encapsulated in a script, then there is no longer a need for detailed and continuous feedback. The user doesn't have to be

bothered unless the system encounters a problem that it cannot handle.

A supervisor interacts very intensely with a new employee and solicits detailed progress reports, but as the employee gains experience and the supervisor gains confidence, the need for this detailed feedback and dialog decreases. The supervisor can assign complex tasks and be confident that "no news is good news." Rather than always providing the user with feedback on activities, the computer should be more flexible in the amount of feedback it provides. Initially, the computer could provide detailed feedback to familiarize the user with its operations and instill confidence; then the feedback could be scaled back over time and restricted to unusual circumstances or times when the user requests more feedback.

## Forgiveness

The forgiveness principle states that user actions should generally be reversible and that users should be warned if they try to do something that will cause irreversible data loss. But even forgiveness can have a negative side. Consider as an example the common case where a user wants to copy a file to a floppy that does not have enough free space. The Macintosh gives an error message explaining there is not enough room and we must throw away 125K. But when we throw away some files and again attempt to copy the file, we get another error message stating there is not enough room unless we empty the trash, and asking if we want to empty the trash now. In this case forgiveness becomes a nuisance. The underlying problem here is that the computer has a very meager understanding of the interaction history. A stateless interface is doomed to present the inappropriate message because it cannot relate the user's action to its own prior recommendation to delete files. The computer needs to build a deeper model of our intentions and history.

## Perceived Stability

Perceived stability means that elements in the computer interface should not be changed without the user's involvement. For example, icons on the desktop and windows should reappear as they were when the user last closed them. But it is a sure sign of a lack of confidence if a person's last words before leaving the room are, "Don't move until I get back." In a computer interface, the principle of perceived stability implies the computer will most likely make things worse on its own and the user will be unable to function in a changing environment. We're still children in the computer age, and children like stability. They want to hear the same bedtime story or watch the same video again and again. But as we grow more capable and are better able to cope with a changing world, we become more comfortable with changes and even seek novelty for its own sake.

One of the most compelling aspects of computer games and some computer-based learning environments is the lack of stability that derives from dividing control between the user and the computer or even among users on networked computers [7]. This should not be surprising, because the world we live and act and play in is an arena of mixed control with initiatives from individuals, their associates, and the larger environment.

There are many things computers and other people can do for us if we decide they don't have to maintain perceived stability. Today, electronic messages and news articles appear unbeckoned on our computer desktops. Applications have existed for some time that can automatically scan and sort incoming mail [12]. Programs such

as Magnet, a Macintosh application that searches for files meeting a specified criterion and moves them into the user's folders, act as simple agents to build an improved environment for the computer user.

The World-Wide Web is a prime illustration of the advantages of shared control in user interfaces. The Web changes constantly, and every time users connect to a home page, they might be presented with a completely new interface. For example, AT&T changes its home page daily [3], and in designing Sun's home page we decided we needed to change it drastically every month to keep the users' interest [14]: Stability can be boring! Denying the principle of perceived stability can often make the interface simpler and more intuitive. We can be easily overwhelmed by all the capabilities of a large application, but if the application discreetly rearranges the interface from time to time to offer us only the features of current interest to us, we can get the feeling that everything we need is easy to find and readily at hand. >

## Aesthetic Integrity

The principle of aesthetic integrity states that the graphic design of the interface should be simple, clean, and consistent. Screens should be visually pleasant and easy to understand. Part of the need for aesthetic integrity derives from the limited expressiveness of current computers. If computers could communicate with a richer language, it would not be so important that everything have a "single look." With networking, a person's computer world extends beyond the bounds of his or her desktop machine. Just as a city designed by a single architect with a consistent visual appearance would be difficult to navigate and somewhat boring to visit, a variety of visual designs would make our computer world more interesting, more memorable, and more comprehensible.

As an extreme example, the old user interface to the Internet had great aesthetic integrity, with one "ls" listing in ftp looking pretty much like any other, and with all needed information (modification date, size, and file type as indicated by the extension) shown in a severely minimalist manner. Even so, the wild and woolly extremes of Web home-page design have taken over. Users seem to prefer more illustrative indications of location and content in cyberspace. Note we do not argue for complete anarchy: With Darrell Sano we recently designed SunWeb (Sun's internal Web pages) to have a unified, though flexible, look [15]. We did want SunWeb to be recognizably different from, say, Silicon Surf (SGI's Web pages), but we did not want the Help buttons to appear in 20 different places.

As our computer world expands (especially over the network) to encompass millions of objects, these objects should not all look the same. Totally uniform interfaces will be drab and boring and will increase the risk of users' getting lost in hyperspace. Richer visual designs will feel more exciting. From a functionality perspective, richness will increase usability by making it easier for users to deal with a multiplicity of objects and to derive an understanding of location and navigation in cyberspace.

## Modelessness

Modelessness means the computer interface should not have distinct modes that restrict the user's actions depending on the mode he or she is in. Users should be able to perform any task at any time. Although modelessness seems to be an object of veneration among some Macintosh interface designers, even the section on modelessness in the Macintosh Human Interface Guidelines [2] is primarily devoted

to explaining how to use modes successfully. The basic problem presented by modelessness is that the user cannot cope with everything at once. Users need the interface to narrow their attention and choices so they can find the information and actions they need at any particular time. Real life is highly moded [11]: What you can do in the swimming pool is different from what you can do in the kitchen, and people can easily distinguish between the two because of the richness of the experience and the ease with which we can move between environments.

# The Anti-Mac Interface

Although this article starts as an exploration of alternatives to the individual Macintosh human interface principles, it has not quite ended up that way. Just as the Macintosh design principles are interrelated and give a resulting coherence to the Macintosh interface, violation of those principles also points to a coherent interface design that we call the Anti-Mac interface.

The basic principles of the Anti-Mac interface are:

- The central role of language
- A richer internal representation of objects
- A more expressive interface
- Expert users
- Shared control

## The Central Role of Language

Over the past million years, humans have evolved language as our major communication mode. Language lets us refer to things not immediately present, reason about potential actions, and use conditionals and other concepts not available with a see-and-point interface. Another important property of language missing in graphical interfaces is the ability to encapsulate complex groups of objects or actions and refer to them with a single name. An interface that can better exploit human language will be both more natural and more powerful. Finally, natural languages can cope with ambiguity and fuzzy categories. Adding the ability to deal with imprecise language to the computer interface will increase the computer's flexibility and its fit with people's normal expressive modes. As Susan Brennan has shown [4], natural language, in addition to being natural for people, has several advantages as a medium for human-computer interaction, including the role of negotiated understanding, use of shared context, and easy integration with pointing and other input/output channels.

We are not proposing, however, what AI researchers would call a "natural language interface." It seems a computer that can hold a normal conversation with the user will remain in the realm of science fiction for some time yet, and we are interested in computer-human interfaces for the near future. Instead, we have in mind something more like the interfaces of text-based adventure games, with their understanding of synonyms, relatively simple syntax, and tolerance for error in the input-—a pidgin language for computers. The critical research question is, "How can we capture many of the advantages of natural language input without having to solve the 'AI-Complete' problem of natural language understanding?" Command line interfaces have some of the advantages of language, such as the large number of commands always available to the user and the rich syntactic structures that can be used to

form complex commands. But command line interfaces have two major problems. First, although the user can type anything, the computer can understand only a limited number of commands and there is no easy way for the user to discover which commands will be understood. Second, the command line interface is very rigid and cannot tolerate synonyms, misspellings, or imperfect grammar. We believe that both these deficiencies can be dealt with through a process of negotiation.

Think of the way a new library user might interact with a reference librarian. A librarian who had a command line interface would understand only a limited number of grammatically perfect queries, and the novice user would have to consult an obscure reference manual to learn which queries to write out. A reference librarian with a WIMP interface would have a set of menus on his or her desktop; the user would search the menus and point to the appropriate query. Neither interface seems very helpful. Instead, real reference librarians talk with the user for a while to negotiate the actual query. Similarly, we envision a computer interface that utilizes a thesaurus, spelling correction, displays of what is possible, and knowledge of the user and the task to take part in a negotiation of the user's command. We could imagine, for example, a dialog in which the user makes a free-form request, the computer responds with a list of possible tasks that seem to match the request, and both engage in a dialog to focus on the request the user actually intended.

## A Richer Internal Representation of Objects

Current interfaces have access to very little information about the objects he user deals with. For example, the only information known about a file may be its name, size, and modification date; the type of data it contains; and the application that created it. In order to deal more effectively with these objects, the computer needs a much richer representation of them. For a document, this representation could include its authors, topic matter, keywords, and importance; whether there are other copies; what other documents it is related to; and so forth. The list of attributes is similar to what would be needed by a good secretary who was expected to handle the documents intelligently. It is not necessary for the secretary to fully understand the document's contents, but he or she must have a general sense of what the document is about and of its significance. If a computer interface is to handle documents intelligently, it must have the same sorts of information.

Much of this information does not require natural language understanding. We already have techniques in full-text search systems that could be used to automatically extract this information from the document. As we move to text systems with tags based on meaning rather than a WYSIWYG system, much of this information will be available in the document itself. Further, if we allow the computer more control over the interface, it could monitor what we do with the objects and change their representations accordingly [10]. For example, if two objects were almost always used together, the computer could create a hypertext link between them.

## A More Expressive Interface

The richer internal representation of objects will allow more intelligent interpretation of user commands, but it will also be reflected in a more expressive interface with a richer external representation. The Macintosh interface was originally designed for a 9-inch display that contained less than 200,000 black-and-white pixels. It is remarkable how well it has translated to today's 21-inch displays that contain

2,000,000 color pixels---an increase of a factor of 240 in information capability. This trend will continue until displays approach the size of a desk and the practical resolution of the human eye (an additional factor of 340) and the interface should take advantage of this change to increase the expressiveness of the objects it displays.


**Figure 3.** *The variety and rich visual affordances of a bookcase make it easy to recognize objects of interest.*

Notice in Figure 3 how the books on a bookshelf have a wide variety of appearances and yet are all recognizable as books. This variety adds visual interest and helps us quickly locate a particular book. Yet in our computer interfaces, all documents of a given application usually appear identical. This practice is starting to change—some graphics applications represent documents with thumbnail images, but that is only a beginning. Improved displays and sound can give us a computer world that is as comfortable to navigate as the real world.

In addition to richer output, richer input devices will allow the user more flexibility in manipulating objects, and monitoring devices like active badges [20] will allow the computer to accommodate the user's needs without explicit commands.

## Expert Users

The GUIs of contemporary applications are generally well designed for ease of learning, but there often is a trade-off between ease of learning on one hand, and ease of use, power, and flexibility on the other hand. Although you could imagine a society where language was easy to learn because people communicated by pointing to words and icons on large menus they carried about, humans have instead chosen to invest many years in mastering a rich and complex language.

Today's children will spend a large fraction of their lives communicating with computers. We should think about the trade-offs between ease of learning and power in computer-human interfaces. If there were a compensating return in increased power, it would not be unreasonable to expect a person to spend several years learning to communicate with computers, just as we now expect children to spend 20 years mastering their native language.

It is sometimes claimed that humans have limited cognitive abilities and therefore will not be able to handle increasingly complex computers. Even though the human brain may stay about the same for the foreseeable future, it is demonstrably not true that

a fixed human brain implies fixed abilities to use specific systems. As an example, consider the ability to survive in a jungle without modern weapons. Most readers of this article would probably fare rather poorly if they were dropped in a jungle and told to catch dinner with their bare hands. At the same time, anybody born and raised in the same jungle would probably perform the dinner-catching task superbly. The difference in performance between the two groups is due to the benefits of having grown up in an environment and fully internalized its signals and rules. As the example shows, two people with the same brain capacity can have very different capabilities when it comes to using a given system. We predict that people who have grown up with computers will be much more capable as computer users than the current generation of users. Thus, they will be able to use (and will in fact, demand) expressive interfaces with advanced means of expressing their wants.

## Shared Control

But the entire burden should not be on the individual user. Computer-based agents have gotten attention from computer scientists and human interface designers in recent years, but their capabilities have not yet progressed beyond the most simple-minded tasks. Before people will be willing to delegate complex tasks to agents, we need agents skilled both in specific task areas and in communicating with users. The recent rapid rise in use of the Web illustrates the advantage of sharing control of your computer world with other people. By relinquishing control over a portion of the world, you can utilize the products of other people's efforts, knowledge, and creativity.

A computer environment in which both humans and computer-based agents have active roles implies a mixed locus of control [13]. The user's environment will no longer be completely stable, and the user will no longer be totally in control. For general sanity, users still need to determine the balance of control in the different parts of their environment, they need the means to find out what their agents have been up to, and they need mechanisms for navigating in the wider networked world; but the Anti-Mac interface trades some stability for the richer possibilities of a shared world. In the real world, we don't want anyone rearranging the mess on our desks, but we don't mind if someone puts a Post-It note on our computer screen, empties the wastebasket overnight, or refills the newspaper stand with the latest edition.

## Mutually Reinforcing Design Principles

During the relatively short history of computing, several changes have taken place in the ways computers are used and in the people who form the main user community. These changes in usage and users lead to important changes in user interfaces. Highly usable designs reflect a set of principles that are mutually reinforcing and that match the needs of the main users and their tasks.

For example, the original text-based Unix user interface was very appropriate when it was designed. The command-line interface was suited for a user community dominated by programmers who had the expertise, interest, and time to understand how the many different system features could be combined for optimal use through mechanisms like piping and shell scripts. Early Unix users manipulated relatively few data objects (their own code, text files, and memos) that were mostly plain Ascii text. The Unix design principles of modular commands, with many parameters and options, worked well for the chosen users and their tasks. As evidenced by several attempts to build graphical user interfaces on top of Unix, it is not possible to modify

the design by simply changing one or two aspects (e.g., by representing files as icons), since the full set of design attributes were chosen to work together. The successful modifications of Unix have been those that combine several new design principles to achieve a new mix of mutually reinforcing design elements (e.g., combining icons with drag-and-drop and multiple workspaces to structure the user's tasks).

Similarly, the Macintosh was optimized for a certain user community with a certain main type of data: knowledge workers without any computer science background who wanted to manipulate graphic documents, but not overwhelmingly many of them. The various Macintosh user interface principles work well together and help the chosen user category achieve its main goals. Since the principles are mutually reinforcing, it may not be easy to break just a few as changing users and tasks lead to new requirements. For example, as argued earlier, it becomes impossible to make all objects visible in the interface as system usage changes to deal with millions of information objects. But having invisible objects not only breaks the design principle of see-and-point, it makes direct manipulation impossible, risks weakening the chosen metaphors, and breaks perceived stability if some objects are made visible anyway at certain times.

The Anti-Mac principles outlined here are optimized for the category of users and data that we believe will be dominant in the future: people with extensive computer experience who want to manipulate huge numbers of complex information objects while being connected to a network shared by immense numbers of other users and computers. These new user interface principles also reinforce each other, just as the Mac principles did. The richer internal representation of objects naturally leads to a more expressive external representation and to increased possibilities for using language to refer to objects in sophisticated ways. Expert users will be more capable of expressing themselves to the computer using a language-based interface and will feel more comfortable with shared control of the interface because they will be capable of understanding what is happening, and the expressive interface will lead to better explanations.

# Conclusions

First, to avoid misunderstandings, let us reiterate that we are not criticizing the Macintosh interface because we think it is bad, but because we view it as a starting point for considering the differing needs future user interfaces must meet. Table 2 compares some of the characteristics of the original Macintosh and Anti-Mac user interfaces.

| **Mac** | **Anti-Mac** |
|---|---|
| Users are "the rest of us" (have no previous computer experience) | Users are "The Post-Nintendo Generation" (grown up with computers) |
| Office automation "productivity" applications | Work, play, groupware, embedded, and ubiquitous |
| Weak computer (128K RAM, 68000 CPU) | Humongous computer (multi-gigabyte RAM, Cray-on-a-chip RISC processors) |
| Impoverished communication bandwidth (small screen, keyboard/mouse input) | Rich communication (computer can see you, knows where you are, large high-res screen, new I/O devices) |
| Stand-alone system that is stable unless the user decides to make a change | Connected system subjected to constant change |
| Manipulation of icons | Language |
| Weak object-orientation (small number of large objects with very few attributes) | Strong object-orientation (large number of small objects with rich attribute sets) |
| "Finder" (visible file system) is unifying home base, and files are the basic interaction object | Personal information retrieval as unifying principle with atomic information units as basic interaction object |
| Surf your hard-drive | Information comes to you |
| "The Power to Be Your Best" (*ed.* Apple's slogan at the time) | You won't always have to work that hard |

**Table 2.** *A comparison of the Mac and Anti-Mac interfaces.*

Bruce Tognazzini's *Starfire* film [19] was an attempt to show how a high-end workstation might look in the year 2004. The interface visualized in the film has several similarities to our Anti-Mac design: The system understands objects well enough to integrate them seamlessly (it can construct a 3D texture map of a person from a 2D video image); the screen is very large, with fairly expressive interface elements; the computer has some independent initiative (when the heroine searches for an article reprint, the agent automatically decides to search further and display subsequent articles with links to the original article); and the computer monitors the user (it does not attempt speech recognition while she is talking with a colleague).

Despite its Anti-Mac aspects, *Starfire* is still a very recognizable computer with many similarities to current user interfaces and research systems. Indeed, there are some Anti-Mac features in evidence in current commercial products. For example, products like On Location and SearchIt show autonomy in taking the initiative to index the user's file system when needed and allow users to retrieve email objects by content and other rich attributes even though each email message may be part of a large text file when viewed through the standard system interface.

Any realistic person who reads the preceding outline of the Anti-Mac interface will realize it is mostly impractical with the current state of computers and software. Language understanding is still in its infancy; most of the population are having their first encounters with computers; and most users know better than to trust computer-based agents with a free rein on their systems. Today's standard WIMP interfaces are fairly well suited to current hardware and software capabilities. But hardware designers are not slowing down, and the challenge for application and interface designers is to take advantage of the coming computing power to move the computer-human interface to a new plateau. Interface designers need to start work on these issues now so that solutions will be available when the next generation of computers arrives, and also to help guide the directions of hardware and software development.

To realize the full benefits from the Anti-Mac approach, we argue, it will not be sufficient to retrofit its features one at a time to systems that basically follow current user interface architectures. We believe an integrated design that builds a new user interface from the bottom up will be more coherent and will have a better potential for increasing user productivity by at least an order of magnitude. A full Anti-Mac system will likely have to be based on deep object structures in an architecture that supports network-distributed objects and detailed attributes that are sufficiently standardized to be shared among multiple functionality modules. Realistically speaking, such a complete redesign will take some time to appear. Even if the full system were not to appear for several years to come, it is necessary for detailed research to begin now to develop the needed features and to collect usability data on how the Anti-Mac characteristics should be shaped to truly meet users' needs.

## Acknowledgments

We would like to thank Bruce Tognazzini, Ellen Isaacs, Robin Jeffries, Jonathan Grudin, Tom Erickson, and the members of the human interface group at SunSoft for valuable comments on this article.

# References

1. Alfven, H. Worlds-Antiworlds. W. H. Freeman, San Francisco, 1966.
2. Apple Computer. Macintosh Human Interface Guidelines. Addison-Wesley, Reading, Mass., 1992.
3. AT&T. Home page at URL http://www.att.com
4. Brennan, S. E. Conversation as direct manipulation: An iconoclastic view. In The Art of Human-Computer Interface Design, B. Laurel, Ed. Addison-Wesley, Reading, Mass, 1990, pp. 393-404.
5. Clymer, F. Treasury of Early American Automobiles. McGraw-Hill, New York, 1950.
6. Cypher, A. Eager: Programming repetitive tasks by example. In Proceedings of the ACM CHI'91 Conference Human Factors in Computing Systems (New Orleans, April 28-May 2) 1991,. pp. 33-39.
7. Gentner, D. R. Interfaces for learning: motivation and locus of control. In Cognitive Modelling and Interactive Environments in Language Learning, F. L. Engel, D. G. Bouwhuis, T. Boesser, and G. d'Ydewalle, Eds. NATO ASI Series, vol. F 87, Springer-Verlag, Berlin, 1992.
8. Grudin, J. The case against user interface consistency. Commun. ACM 32, 10 (1989), 1164-1173.
9. Halasz, F., and Moran, T. P. Analogy considered harmful. In Proceedings of the ACM Conference on Human Factors in Computer Systems (Gaithersburg, Md., March 15-17) 1982, pp. 383-386.
10. Hill, W.C., Hollan, J.D., Wroblewski, D., and McCandless, T. Edit wear and read wear. In Proceedings of the ACM CHI'92 Conference on Human Factors in Computing Systems., (Monterey, Calif., May 3-7) 1992, pp. 3-9.
11. Johnson, J. Modes in non-computer devices. Int. J. Man-Mach. Stud. 32 (1990), 423-438.
12. Malone, T. W., Grant, K. R., Turbak, R. A., Brobst, S. A., and Cohen, M. D. Intelligent information-sharing systems. Commun. ACM 30, 5 (1987), 484-497.
13. Nielsen, J. Noncommand user interfaces. Commun. ACM 36, 4 (1993), 83-99.
14. Nielsen, J. Interface design for Sun's WWW site. (June 1995).
15. Nielsen, J., and Sano, D. SunWeb: User interface design for Sun Microsystems' internal web. In Proceedings of the Second International WWW Conference '94: Mosaic and the Web. (Chicago, Oct. 17-20).
16. Open Software Foundation.OSF/Motif Style Guide. Prentice-Hall, Englewood Cliffs, N.J., 1993.
17. Shneiderman, B. Direct manipulation: A step beyond programming languages. IEEE Comput. 16, 8 (1983), 57-69.
18. Sun Microsystems, Inc. OPEN LOOK: Graphical User Interface Application Style Guidelines. Addison-Wesley, Reading, Mass, 1990.
19. Tognazzini, B. The *Starfire* video prototype project: A case history. In Proceedings of the ACM CHI'94 Conference on Human Factors in Computing Systems (Boston, April 24-28), pp. 99-105. Film clips available.
20. Want, R., Hopper, A., Falco, V., and Gibbons, J. The active badge location system. ACM Trans. Inf. Syst. 10, 1 (1992), 91-102.
21. Weiser, M. The computer for the 21st century. Sci. Am. 265, 3 (1991), 94-104.
22. The Windows Interface: An Application Design Guide. Microsoft Press, Redmond, Wash., 1992.

# Learn More

Full-day tutorials with **current guidelines for GUI and application design**:

- Application Usability 1: Page-Level Building Blocks for Feature Design

- Application Usability 2: Dialogue and Workflow Design

---