Ease of Use: Design concepts section Back to web version

Design concepts

Computer products too often seem designed for the computer engineers who create them. But IBM is working to change this practice. We believe that all users have the right to an enjoyable experience when using a computer. We also believe that sharing the knowledge we have acquired from our own practical experience can help others create hardware and software that is easy for everyone to use. Here in "Design Concepts" we discuss the users' bill of rights that we subscribe to, the principles that drive successful user interface design, and using models to design for Ease of Use.

What is user experience design? The foundation Human-Computer Interaction provides for User Experience Design.

What is a user interface? A definition and explanation of the term *user interface*.
Interacting with computers - UI evolution

Design basics Principles that form the foundation of good design.

User rights The bill of rights for computer users, and changes that must take place in the computer industry.

The three models Models used in designing for Ease of Use.
Introduction - User - Designer - Implementor

What is user experience design?

User Experience Design fully encompasses traditional Human-Computer Interaction (HCI) design and extends it by addressing all aspects of a product or service as perceived by users. HCI design addresses the *interaction* between a human and a computer. In addition, User Experience Design addresses the user's initial awareness, discovery, ordering, fulfillment, installation, service, support, upgrades, and end-of-life activities. HCI design constitutes a major portion of the activities performed by a User Experience Design team, so the following paragraphs provide an overview of HCI design followed by references to additional material about User Experience Design.

Human-Computer Interaction, or HCI, is the study, planning, and design of what happens when you and a computer work together. As its name implies, HCI consists of three parts: you the user, the computer itself, and the ways you work together.

The Human Perspective

HCI design teams must consider these factors in regard to users: what users expect and need, what physical abilities and limitations they may have, how their perceptual systems work, and what they find attractive and enjoyable when they use computers.

When humans interact with computers, they bring to the encounter a lifetime of experience. Even the first time we touch a computer, expectations learned in other areas of life can affect how we think a computer should work. For instance, because of our experience with other machines, we expect computers to provide immediate feedback when we press the on button. Elevators, automobiles, and other machines provide immediate auditory and visual cues that machine is responding, and so we expect the same from computers. Without this feedback, we wonder if the computer is functioning

properly.

Since users have various preferences, work environments, and physical capabilities, designers must also provide alternative ways for different users to communicate with their computers. Information can be exchanged by voice, keyboard, mouse, or other means.

Understanding how people's sensory systems (sight, hearing, touch) relay information is essential to designing a good product. For example, display layouts should accommodate the fact that people can be distracted by the smallest movement in the outer (peripheral) part of their visual fields, so only urgent conditions should be indicated by moving or blinking visuals.

And of course people like designs that hold their attention. Designers must decide how to make products attractive without distracting users from their tasks.

The Computer's Persona

In the natural world, most actions have obvious consequences. When you pick up your clothes from the cleaners', you see the clothes on their hangers, hear the rustling sound of their plastic sheaths, and feel their weight as you carry them. All these experiences serve as feedback confirming that you successfully completed your errand.

A computer carries on its business in a much less obvious way. The information a computer contains and the operations it performs are represented inside the computer in a form that we can't directly observe - binary digits encoded as two levels of electrical charge. What a computer displays or presents does not arise naturally from what it is doing inside. Any feedback the user might need must be explicitly planned out and programmed.

To make matters worse, computers don't even "think" as we do. They can remember amazingly large sets of instructions, but they have to be told every little thing in simple terms of "if this happens, do that" or "as long as this keeps happening, do that." And things we humans do almost automatically, such as jumping to conclusions or neglecting a trivial matter to take care of something more important, require even more extensive instructions to the computer.

Interaction

So, given all these differences between humans and computers, how are we supposed to get along with them and get our work done? In other words, how can we interact with them effectively?

In order to come up with a product that's easy for people to use, software designers apply what they know about humans and computers, and consult with potential users of their products throughout the design process. When they know what their users want and need the product to do, they collaborate with programmers. Programmers know how to write instructions in languages that computers can understand. They also know what computers are capable of doing. The designers and programmers look for a reasonable balance between what can be programmed (written as computer instructions) within the necessary schedule and budget, and what would be ideal for the users. They have users try out any changes to make sure that the product is still easy, efficient, and pleasant to use.

As you see, designers and programmers play important roles in HCI, but it's the computer users who have the final say about the quality of the interactions.

User Experience Design

While User Experience Design includes the human-computer interface, it is about designing the *total user experience*, which consists of all aspects of a product or service as perceived by users. Additional information about User Experience Design is available on this Web site. The User Engineering section describes an overall process for doing User Experience Design, and specific design guidelines are

available in the sections on out-of-box experience (OOBE) and Web guidelines.

This site is undergoing an evolution to become an authoritative and comprehensive source for information about designing the total user experience - through the rigorous and precise methods of User Engineering. Please visit frequently, watch for highlights of recent additions, and provide Feedback so this evolution can be guided to fulfill your needs.

What is a user interface?

User Interface (or *UI*) is one of those jargon-y terms that you hear from computer salespeople and other techno-geeks, but that you may have never heard defined. It's not a hard concept to understand, though. It's simply the parts of a computer and its software that you (the computer *user*) see, hear, touch, or talk to. It is the set of all the things that allow you and your computer to communicate with each other. For example, if you are reading this on a computer screen, then you're looking at part of a user interface right now. The screen is showing you these words, communicating a message to you.

Like any good communication channel, a user interface is a two-way street. You don't want to just see or hear whatever the computer puts in front of you, you also want to tell it what you'd like to do. For instance, to get to this web page, you "asked" your computer to show you pages on a certain subject. You may have used a mouse to point and click on a button or word, or maybe you spoke instructions to the computer. However you express it, everything you tell the computer is *input*; what it conveys to you is *output*. The ways you can receive output and give input depend entirely on the user interface.

The best user interfaces are the ones you don't have to pay much attention to. They make sense to you and do what you expect them to. When an interface is easy to use, you can spend your time doing your work instead of looking everywhere for the right button or key to press. It's almost *transparent*--you can see right through the interface to your own work.

Interacting with computers A discussion of input and output devices used to communicate with users, and controls used to set preferences and make choices.

UI evolution The evolution of the user interface, from the command-line to objects that borrow from the real world.

Interacting with computers

"Oh, just do what I want, you confounded machine!"

Computers are not like you and me, so to communicate with them, we need go-betweens. At a basic level, we need something to relay information from the computer to us, and from us to the computer. We call these translation aids *input devices*, *output devices*, and *controls*.

Input devices are the hardware components you use to "talk" to a computer. You use them to place requests, send messages (to the computer or to other people), move around in virtual worlds, or even shoot at "enemies" in some computer games. A few examples of commonly used input devices are:

- Computer keyboard
- Joystick

- Microphone
- Mouse
- Pen (some with, some without, a pad)
- Touch-sensitive screen
- Trackball
- TrackPoint

Each input device is optimized for certain uses or users, and less efficient or less convenient for others. For example, a microphone used with voice-recognition software can be very helpful for people who can't use a keyboard (whether due to work environment or disability). But voice input is difficult in very noisy places, impractical in situations where quiet is required, and possibly more error-prone than some other techniques. Joysticks and trackballs are well-suited for navigational control, as in video games or exploration of 3D environments, where smooth movement is more important than fine target acquisition. Pen input can be very convenient for a package carrier recording delivery signatures, but learning to use pen gestures as commands requires some study and practice. No one input device is best for every situation.

Output devices are the various hardware elements a computer uses to communicate with the user. Some examples of output devices are:

- Head-mounted display
- Headphones
- Pen-sensitive screen (serves as both an input and output device)
- Printer
- Speakers
- Touch-sensitive screen (serves as both an input and output device)
- Video display terminal (monitor)

Like input devices, these devices are specialized for particular uses. The head-mounted displays and headphones, for example, are used for *immersive virtual reality* applications, such as arcade games and flight simulators. They present 3-dimensional images to the user's eyes and stereo sound to the ears, while excluding the sights and sounds of the user's physical environment. The illusion of being in another world can be very compelling. But using such a device for extended periods of time, as when performing office tasks, could cause problems. The user can suffer nausea from the disparity between the motion perceived by the inner ear and that perceived by the eyes; eyestrain from extended near focus on the display; neckstrain from the weight of the helmet; and discomfort from heat buildup in the helmet and headphones. The user is also socially isolated from peers (a mixed blessing). Refinements to the technology will ease some of these problems, while others are intrinsic to each medium.

Controls are the software elements, usually shown on a display, that you use to set preferences and make choices. Like such hardware controls as knobs and dials, they can be used to control many different things. Some familiar controls include:

- Menus
- Pushbuttons
- Radio buttons
- Sliders

Some software controls are used for both output and input; they show your choices or the current setting, and allow you to operate the control.

These elements are all necessary for interaction between people and computers, but they are not enough to guarantee a "meeting of the minds." We also need an *interface* that is easy to understand and to use.

UI evolution

Human-computer interfaces have come a long way since the early days when computer users typed line after line of computer jargon using green text on black screens. This was clearly an interface that only a technologist could love - the rest of us simply had to put up with it if we wanted or had to use a computer. It was clear to some people that significant improvements would be needed in order to achieve the increase in productivity that was being promised through the use of computers.

This awareness led to advances that made the interface more visual using techniques such as menus to ease the burden on users' memories. But the human-computer interface was still comparatively in the dark ages - each application had its own unique interface and there was little similarity across applications. The computer could still only be used effectively by highly-skilled specialists and enthusiasts who were willing to invest a significant amount of time in learning, and who were willing to put up with the computer's quirky demands and behaviors.

By the time the modern graphical user interface, or GUI, became available, computer technology was becoming more accessible to businesses and individuals. The GUI interface used computer graphics, little pictures called *icons,* and the *mouse* to make using a computer easier. Many applications started to look similar because standard *controls,* such as menus, buttons, and check boxes were provided by the computer manufacturer. It was easier for application developers to use the standard controls than it was for them to develop their own - so users benefited as well. But even though the basic interface mechanisms were becoming more consistent and easier to use, applications and the overall user environment were becoming more and more complex. A word processor could be used not only for writing documents, but could also include spreadsheet data, charts, and drawings. Users were no longer limited to running one application at a time - they could run several in separate *windows* that overlapped on the display.

In the late 1980's, the HCI group at IBM recognized that users would be overwhelmed by these new capabilities, and that the computer itself was doing little to help them manage several things at once. This recognition led to the development of the object-oriented user interface, or OOUI, a style of interface that is becoming popular today in such systems as Microsoft's Windows 95 and IBM's OS/2 Warp. An OOUI allows users to focus on the information they need to do their work, and it hides many of the traditional aspects of using a computer that users don't need or want to worry about.

Where is this evolution leading? To be fully embraced by the general population and become a bonafide *consumer* product, the evolution must take the computer through some further steps to make it even more simple and natural to use. One major factor will be the presentation of information and computer capabilties that resemble what users see and experience in the real-world. Users will interact with telephones, fax machines, and writing tablets on the computer display that look and behave very much like their real-world counterparts, while at the same time providing additional capabilities that aren't possible in the real-world. Users will visit *places* presented using 3D graphics and *virtual reality* techniques. They will visit libraries and historical sites, and chat with friends and colleagues throughout the world, all from the comfort of their office, living room, or hotel room. Instead of using cumbersome and unfamiliar computer-oriented devices, users will interact with computers using natural human-oriented techniques, such as writing and speaking. And the computer will exhibit characteristics of *personality* that will make it seem more friendly and pleasant to work with.

This evolutionary advancement is being driven by the fast-growing technology of the personal computer, and increasing demands from users that the computer match their way of thinking, rather than the other way around. Computers are becoming more a part of everyday life. As a result, we recognize that the user interface is one of the most critical elements of consumer acceptance. Please take a few moments and visit some of our other web pages to learn more about IBM's vision and our approach to making your experience using a computer both productive and enjoyable.

Design basics

The design principles presented here combine traditional wisdom with extensions to address the evolution of future interfaces. Existing design principles are based on our own experiences in user interface design, on the design experiences of others, and on insights from linguistics and psychology. We have extended these design principles to address evolving interfaces that will provide a more friendly appearance and behavior in the future. The increasing use of 3-D and real-world representations as well as the blossoming popularity of the Internet and the World Wide Web have strongly influenced these progressions.

The most recent influence on these principles has come from our design experience in creating an object-oriented user interface (OOUI). IBM pioneered OOUI architecture and design. Popular operating systems such as Windows 95, IBM OS/2 Warp, and CDE for Unix provide varying degrees of object-orientation for users.

In order to effectively apply these design principles, you need to understand users' tasks and requirements. Understanding and applying principles will be meaningless if users are unhappy with the final product.

Our goal for user interface design is to have the interface positively support users' endeavors and never intrude adversely. The interface should be transparent to the task the user is trying to accomplish and be efficient, satisfying, and fun to use.

Design Principles

Simplicity: Don't compromise usability for function

Keep the interface simple and straightforward. Users benefit from function that is easily accessible and usable. A poorly organized interface cluttered with many advanced functions distracts users from accomplishing their everyday tasks. A well-organized interface that supports the user's tasks fades into the background and allows the user to work efficiently.

Basic functions should be immediately apparent, while advanced functions may be less obvious to new users. Function should be included only if a task analysis shows it is needed. Therefore, keep the number of objects and actions to a minimum while still allowing users to accomplish their tasks.

Support: Place the user in control and provide proactive assistance

To give users control over the system, enable them to accomplish tasks using any sequence of steps that they would naturally use. Don't limit them by artificially restricting their choices to your notion of the "correct" sequence.

The system should also allow users to establish and maintain a working context, or frame of reference. The current state of the system and the actions that users can perform should be obvious. Users should be able to leave their systems for a moment or a day and find the systems in the same familiar state when they return. This contextual framework contributes to their feeling of stability.

Most users perform a variety of tasks, being expert at some and novice at others. In addition to providing assistance when requested, the system should recognize and anticipate the user's goals, and offer assistance to make the task easier. Ideally, assistance should provide users with knowledge that will allow them to accomplish their tasks quickly. Intelligent assistance is like the training wheels on a bicycle - at some point, most users will want to take them off and go forward on their own. The assistance should allow them to become independent at some point when they choose to be so.

Familiarity: Build on users' prior knowledge

Allow users to build on prior knowledge, especially knowledge they have gained from experience in the real world. A small amount of knowledge, used consistently throughout an interface, can empower the user to accomplish a large number of tasks. Concepts and techniques can be learned once and then applied in a variety of situations. Users should not have to learn new things to perform familiar tasks. The use of concepts and techniques that users already understand from their real world experiences allows them to get started quickly and make progress immediately.

The metaphors used in today's user interfaces tend to be inadequate when compared to the real world. Through the use of visuals and interaction techniques that more closely resemble users' real world experiences, there should be little need to continue reliance on such metaphors.

In the past, designers tended to invoke a principle of consistency when no single design alternative appeared to be the best answer. By choosing to be consistent with something the user already understands, an interface can be made easier to learn, more productive, and even fun to use.

Avoid the tendency to employ consistency without understanding your users, their tasks, and their shared experiences. When choosing a dimension within which to be consistent, seek to understand what the user expects and be consistent with those expectations. Providing a familiar experience is the ultimate use of consistency in which a truly intuitive interface will result.

Obviousness: Make objects and their controls visible and intuitive

Where you can, use real-world representations in the interface. Real-world representations and natural interactions (direct action) give the interface a familiar look and feel and can make it more intuitive to learn and use. Icons and windows were early attempts to draw on user experiences outside the computing domain. As we move toward real-world representations, reliance on such computer artifacts should decline. In an object-oriented interface the objects and concepts presented to users parallel familiar things from the real world; for example:

- Trash can - when we throw things away we usually use some type of trash receptacle or "trash can". An object on the desktop displayed as a trash can communicates to users that it is a place for discarding things. It should look like the real object rather than like an abstract container, and the user should be able to show its contents in a meaningful way.
- Telephone - the actions we take with telephones are so familiar to most of us that they require little thought. A telephone object on the desktop indicates to users that it will allow them to perform phone-related tasks, and users will expect it to behave like the real thing.

The controls of the system should be clearly visible and their functions identifiable. Visual representations provide cues and reminders that help users understand roles, remember relationships, and recognize what the computer is doing. For example, the numbered buttons on the telephone object indicate that they can be used to key in a telephone number.

Allow users to interact directly with objects and minimize the use of indirect techniques. Identifying an object and doing something with it (like picking up the handset of a phone to answer it) usually are not separate actions in the real world. Likewise, with direct action techniques, explicit selection is not necessary because selection is implicit in the actions users take with objects. Real-world 3D interfaces are especially conducive to direct interaction.

Encouragement: Make actions predictable and reversible

A user's actions should cause the results the user expects. In order to meet those expectations, the designer must understand the user's tasks, goals, and mental model. Use terms and images that match users' task experience, and that help users understand the objects and their roles and relationships in accomplishing tasks.

Users should feel confident in exploring, knowing they can try an action, view the result, and undo the action if the result is unacceptable. Users feel more comfortable with interfaces in which their actions do not cause irreversible consequences.

Even seemingly trivial user actions, such as deselection or moving objects, should be reversible. For example, a user who spends several minutes deliberating and selecting individual files to be archived from a group will be very upset if all the files are accidentally deselected and the deselection cannot be undone.

Avoid bundling actions together, because the user may not anticipate the side effect. For example, if a user chooses to cancel a request to send a note, only the send request should be cancelled. Do not bundle another action, such as deletion of the note, with the cancel request. Rather than implementing composite actions, make actions independent and provide ways to allow users to combine them when they wish.

Satisfaction: Create a feeling of progress and achievement

Allow the user to make uninterrupted progress and enjoy a sense of accomplishment. Reflect the results of actions immediately; any delay intrudes on users' tasks and erodes confidence in the system. Immediate feedback allows users to assess whether the results were what they expected and to take alternative action immediately. For example, when a user chooses a new font, the font of all applicable text, or of sample text, should change immediately. The user can then decide if the effect is what was desired and, if not, can change it before switching attention to something else.

Offer a preview of the results of an action when it would be inconvenient for a user to apply the action and then reverse it. For example, if a user wants to bold, underscore, and use helvetica font in certain places throughout a document, provide a sample part of that document with those changes applied, allowing the user to decide if that is the right action to take. This saves the user a lot of time by not having to reverse the action that's been applied to an entire document and enhances the user's confidence in the system.

Avoid situations where users may be working with information that is not up-to-date. Information should be updated immediately or refreshed as soon as possible so that users are not making incorrect decisions or assumptions. If, for some reason, the results of a refresh cannot be displayed immediately, the situation should be communicated to users. This becomes especially important in networked environments where it is more difficult to maintain state between networked systems dynamically. For example, most Web browsers display a completion percentage in the information area so that users know the progress of the graphics loading process.

Availability: Make all objects available at all times

Users should be able to use all of their objects in any sequence and at any time. Avoid the use of modes, those states of the interface in which normally available actions are no longer available, or in which an action causes different results than it normally does.

Modes restrict the user's ability to interact with the system. For example, one of the most common uses of modes in menu-driven systems is the modal dialog box (such as "Print" and "Save as") used to request command parameters. Modal dialogs tend to lock users out of their system; to continue, users must complete - or cancel - the modal dialog. If users need to refer to something in an underlying window to complete the dialog, they must cancel the dialog, access the information they need and re-invoke the dialog.

Safety: Keep the user out of trouble

Users should be protected from making errors. The burden of keeping the user out of trouble rests on the designer. The interface should provide visual cues, reminders, lists of choices, and other aids, either automatically or on request. Humans are much better at recognition than recall. Contextual and hover help, as well as agents, can provide supplemental assistance. Simply stated, eliminate the opportunity for user error and confusion.

Users should never have to rely on their own memory for something the system already knows, such as previous settings, file names, and other interface details. If the information is in the system in any form, the system should provide it.

Two-way communication may be necessary at times to allow users to clarify or confirm requests, or to remedy a problem. In the past, many interfaces have treated communication with users as primarily one-way, computer-to-user. The communication should be interactive - as rich in presentation and interaction capabilities as the rest of the interface. It should present relevant information, provide access to related information and help, and allow users to make task-specific decisions to continue. For instance, spell check, as designed in some systems, highlights potentially misspelled words as users work, allowing them to either select a new word or continue to work until they reach a point where they can go back and validate the potentially misspelled words.

Adopt the following design perspective: users know what they want to accomplish, but sometimes they find it difficult to express their desires using the objects and actions provided, and the system is unable to recognize their request. Two-way communication may be used to help users reach their goals.

Versatility: Support alternate interaction techniques

Allow users to choose the method of interaction that is most appropriate to their situation. Interfaces that are flexible in this way are able to accommodate a wide range of user skills, physical abilities, interactions, and usage environments.

Each interaction device is optimized for certain uses or users and may be more convenient in one situation than another. For example, a microphone used with voice-recognition software can be helpful for fast entry of text or in a hands-free environment. Pen input is helpful for people who sketch, and mouse input works well for precisely indicating a selection. Alternative output formats, such as computer-generated voice output for foreign language instruction, are useful for some purposes. No single method is best for every situation.

Users should be allowed to switch between methods to accomplish a single interaction. For example, allow the user to swipe-select using the mouse, then to adjust the selection using the keyboard. At the same time, users should not be *required* to alternate between input devices to accomplish what they perceive as a single step or a series of related steps in a task. For example, it would be tedious to require the use of a mouse for scrolling while editing text from the keyboard. Users should be able to complete an entire useful sequence through the same input device.

Providing a range of interaction techniques recognizes that users are individuals with different abilities and situations. The differences include disabilities, preferences, and work environments.

Personalization: Allow users to customize

The interface should be tailorable to individual users' needs and desires. No two users are exactly alike. Users have varying backgrounds, interests, motivations, levels of experience, and physical abilities. Customization can help make an interface feel comfortable and familiar.

Personalizing a computer interface can also lead to higher productivity and user satisfaction. For example, allowing users to change default values can save them time and hassle when accessing frequently used functions.

In an environment where multiple users are using a shared machine, allow the users to create their own system personality and make it easy to reset the system. In an environment where one user may be using many computers, make personalization information portable so the user can carry that "personality" from one system to another.

Affinity: Bring objects to life through good visual design

The goal of visual design in the user interface is to surface to the user in a cohesive manner all aspects of the design principles. Visual design should support the user model and

communicate the function of that model without ambiguities. Visual design should not be the "icing on the cake" but an integral part of the design process. The final result should be an intuitive and familiar representation that is second nature to users.

The following are visual design principles that promote clarity and visual simplicity in the interface:

- **Subtractive design** - reduce clutter by eliminating any visual element that doesn't contribute directly to visual communication.
- **Visual hierarchy** - by understanding the importance of users' tasks, establish a hierarchy of these tasks visually. An important object can be given extra visual prominence. Relative position and contrast in color and size can be used.
- **Affordance** - when users can easily determine the action that should be taken with an object, that object displays good affordance. Objects with good affordance usually mimic real world objects.
- **Visual scheme** - design a visual scheme that maps to the user model and lets the user customize the interface. Do not eliminate extra space in your image just to save space. Use white space to provide visual "breathing room."

User rights

The customer is always right

There was a time when the very people who designed hardware and software were the only ones to use them. For such highly skilled technicians, the highly complicated systems they developed were not a problem; in fact, they were preferred. But, times have changed. Today, technical specialists are greatly outnumbered by a new and growing category of computer user - the novice. These users may be highly trained professionals, but relatively new to computing.

It's a simple fact. And, as IBM usability expert Dr. Clare-Marie Karat points out, it's a shift in customer base that requires a shift of focus: "The computer industry must change its perspective and design products and systems for the intended user of the product - with all of the user's skills, abilities, and faults in mind. The user is, after all, the customer."

To assist this change in perspective, Karat has proposed a new set of 10 industry guidelines "to transform the culture in which information technology systems are designed, developed and manufactured," and to ensure all future products are precisely what the customer expects. Her theory: in this new computer age, the customer is not only right, the customer has rights.

User Rights

1. Perspective: The user is always right. If there is a problem with the use of the system, the system is the problem, not the user.
2. Installation: The user has the right to easily install and uninstall software and hardware systems without negative consequences.
3. Compliance: The user has the right to a system that performs exactly as promised.
4. Instruction: The user has the right to easy-to-use instructions (user guides, online or contextual help, error messages) for understanding and utilizing a system to achieve desired goals and recover efficiently and gracefully from problem situations.
5. Control: The user has the right to be in control of the system and to be able to get the system to respond to a request for attention.
6. Feedback: The user has the right to a system that provides clear, understandable, and accurate information regarding the task it is performing and the progress toward completion.
7. Dependencies: The user has the right to be clearly informed about all systems requirements for

successfully using software or hardware.
8. Scope: The user has the right to know the limits of the system's capabilities.
9. Assistance: The user has the right to communicate with the technology provider and receive a thoughtful and helpful response when raising concerns.
10. Usability: The user should be the master of software and hardware technology, not vice-versa. Products should be natural and intuitive to use.

"Technologists need to realize that the bulk of customers they are now building product for are users who expect the technology to work well and have practical applicability," Karat continues. Consequently, she states, "there is a critical need for the industry to focus on ease of use in order to address the needs of new customers and to continue growth in the technology industry."

Still, not only do "individual companies need to focus on ease-of-use," Karat believes "there will also need to be cooperation across the industry to achieve these goals, as systems today integrate components from several companies." Simply put, making IT easy for the customer is a team effort. IBM is working with the major companies in the industry to remove inhibitors in order to make life easier for tomorrow's users.

Making IT easy is the obvious choice. Building ease of use into products "builds quality into products," Karat says, "and benefits both the customers and the industry."

The three models

Models facilitate understanding users, analyzing complex systems, and describing effective designs. The use of three models contributes to the design of easy-to-use computer systems: the user's conceptual model, the designer's model, and the programmer's model. Here we provide an understanding of the three models, including how they are used and the relationships between them.

Introduction How the use of models facilitates designing for ease of use.

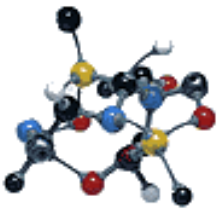
User Understanding the user's perspective - the user's conceptual model.

Designer Specifying what we want the user experience to be - the designer's model.

Implementor Conveying a complete and unambiguous design to the implementors - the implementor's model.

Introduction

How the use of models facilitates designing for ease of use



Models are used in many different fields. Researchers in physics, chemistry, and molecular biology use models to explore relationships between atomic and molecular components of systems. Economists and city planners use models to analyze and predict the performance of complex economic and social systems. Teachers use models as an aid in explaining complex systems in a variety of fields. Whether the model is a plastic replica of an airplane, an exploded parts diagram in a book, or an elaborate computer simulation model, the purpose of the model is to convey an understanding of the components that make up an object or a system and the relationships between components.

Models can be useful in designing and analyzing the user interface of a computer system. Relationships between elements in the interface, the programming system that implements the elements, and the users of the interface can be described and analyzed by using models. Furthermore, a model of the interface can be implemented as a prototype to support iterative testing with users.

We use three models that are relevant to the design and implementation of a user interface. Each model provides a different perspective on the interface, beginning with the end user's perspective, and including the designer's perspective, and the implementing programmer's perspective.

In user interface design we use models to describe an interface in terms of objects, properties, behaviors, and relationships between objects. A model provides a framework for analysis, understanding, and decision making.

A model does not need to address every aspect and feature of a system. A level of detail adequate to understand relationships of interest, explain observations, and make design tradeoffs is sufficient. In some cases it may be desirable to use several different models with various levels of detail for the same system. One model might be adequate for a salesperson to explain the system to a prospective customer. Another model of the system might be needed to help develop specifications for subcontracted components.

A model must be accurate at whatever level of detail is chosen. Models must be under constant scrutiny and should be changed to reflect varying requirements and explain observed behaviors.

We have found it useful to consider three models during user interface design:

- The *user's conceptual model* represents what the user thinks is happening and why
- The user interface *designer's model* describes what the user is intended to experience
- The *programmer's model* describes implementation details

To illustrate the relationship between these three models, consider an analogy between the role of a user interface designer and an architect who is designing a house. These roles are similar in many respects because both of them require an understanding of all three models.

The user interface designer's role is to create a designer's model, or blueprint, of the user interface, just as an architect creates a blueprint of a house. To do this, the designer must:

- Understand the user's conceptual model. Just as an architect must understand a client's needs and expectations to design a house that pleases the client, the user interface designer needs to understand users, their tasks, and their expectations.
- Use accepted user interface design principles. Architects use basic principles that apply to housing design. A good architect knows the environment in which the house will be built with regard to temperature, weather, humidity, and other factors, and successful designs that have been used in that environment. Accordingly, the user interface designer needs to have a knowledge of accepted and proven principles in the field of user interface design.
- Understand the capabilities and limitations of the programming environment, and the skills of the programmers who will be implementing the interface. Just as an architect must know the strengths and weaknesses of building materials and the skills of the tradespeople who will build the house, user interface designers must understand the capabilities and restrictions of operating systems, file systems, window managers, programming toolkits, and other components used to implement the interface.

User

Understanding the user's perspective - the user's conceptual model

The *user's conceptual model* of a system is a mental image that each user subconsciously forms as he or she interacts with the system. People create mental models by putting together sets of perceived rules and patterns in a way that explains a situation. A typical person cannot draw or describe his or her mental models and in many situations the person is not even aware that these mental models exist.

A mental model does not necessarily reflect a situation and its components accurately. Still, a mental model can help people predict what will happen next in a given situation, and it serves as a framework for analysis, understanding, and decision-making.

The user's conceptual model is based on each user's expectations and understanding of what a system provides in terms of functions and objects, how the system responds when the user interacts with it, and the goals the user wants to accomplish during that interaction. These expectations, understandings, and goals are influenced by the user's experiences, including interaction with other systems, such as typewriters, calculators, and video games.

Because each user's conceptual model is influenced by different experiences, no two conceptual models are likely to be the same. Each user looks at a user interface from a slightly different perspective.

The problem for the interface designer is to design an interface that users find predictable and intuitive when each user is approaching the interface from a different perspective. To come as close as possible to matching users' conceptual models, designers should find out as much as they can about users' skills, motivations, the tasks they perform, and their expectations. This process involves:

- using resources such as task analysis, surveys, customer visits, and user requirements lists
- incorporating information that users provide into the user interface design
- conducting usability tests

This is an iterative process that may require many cycles. As the design progresses, users may identify aspects of the interface that are difficult to learn, that are counter-productive, or aspects they simply do not like.

Through interaction with the user interface, users' conceptual models may be expanded, which in turn may cause them to realize new requirements that they had not thought of before. As users provide this level of information to the designer, the picture of their conceptual models will become clearer.

Conceptual models of an object-oriented user interface consist of the objects, properties, behaviors, and relationships of those objects, that are involved in the user's interaction with the system.

When users first interact with a new interface, they are likely to attempt to understand its operation in terms of roles and relationships they already understand. In other words, we each carry with us a current conceptual model. Where existing models lead to correct expectations, the model is reinforced and the user will feel the interface is intuitive. When results are not as expected, the user may rationalize by inventing new roles and relationships in their model, in order to explain observed behavior.

If the user-supplied extensions are accurate, they will be reinforced through interaction with similar aspects in different parts of the system. Otherwise, they are likely to cause confusion. Sometimes users develop superstitions about the interface. These superstitions result from incorrect rationalizations about how and why the interface appears to behave as it does. Superstitions are likely to cause unexpected results in response to the user's actions and further contradict the user's intuition. This can lead to a breakdown in understanding and trust by the user. The use of metaphors and consistency by designers can help user's correct and extend their conceptual models.

A new interface should resemble something familiar to help users get started, then allow them to explore new concepts. It is often said that a characteristic of a good user interface is that it is *intuitive.* Perhaps when used in this sense intuition can best be characterized as a good match between the user's conceptual model and the designer's model.

By using metaphors, designers can take advantage of users' experience and allow a user to rely on intuition while expanding the user's conceptual model to take advantage of new capabilities provided by the interface. Interfaces that use metaphors and allow users to safely explore the computerized environment are popular for this reason.

For example, a computerized car dealer application might provide a *worksheet object* to be used by a salesperson in the task of selling a car. The computerized worksheet would contain the same information and would be used in the same way as a paper worksheet. Like the paper worksheet, the worksheet object would allow the salesperson to enter the car's price and stock number, the customer's name and address, and information about the proposed terms of the sale.

However, the computer-based worksheet could also expand the salesperson's conceptual model by providing capabilities that go beyond those of a paper worksheet. Instead of typing information into the worksheet one field at a time, the salesperson might simply "drag and drop" a car object onto the worksheet. The fields in the worksheet that are relevant for the car being sold would be automatically filled in by the associated fields from the car object. Monthly payments and finance charges could be calculated automatically. Instead of having to hand a paper worksheet to the sales manager for approval, the salesperson could drag and drop the worksheet into a specific mail outbasket to have it automatically sent to the sales manager through the dealer's computer network.

This worksheet object would not only meet the salesperson's expectations, it would go beyond them. It is an object that the salesperson expects to use during the task of selling a car, it has behaviors and characteristics that the salesperson is accustomed to, and it provides additional value through the use of a computer.

In this example, the worksheet object acts as a metaphor for an object that already exists in the salesperson's conceptual model of a car dealership and the task of selling cars. It is an object with which the salesperson is already comfortable, yet it provides additional capabilities that make the salesperson's job easier than using a paper counterpart.

Users' conceptual models constantly evolve as they interact with an interface. Just as users influence the design of a product, the interface design influences and modifies users' concepts of the system. Designers can help users develop an accurate conceptual model by using well defined distinctions between objects and by being consistent across all aspects of the interface.

For example, given an object-oriented car dealer application, the salesperson would open and work with familiar objects, instead of starting and running computer programs, opening files, and so forth. This object-oriented approach has fewer concepts for the salesperson to deal with and matches the salesperson's real world better than one in which a task is accomplished by starting applications and opening files.

Naturally, the conceptual model of a salesperson who is already familiar with using a graphical computer interface requires little modification. This salesperson would already know how to use icons, windows, menu bars, and push buttons.

In any case, the distinctions between objects must be clear and useful, and the interface must be consistent. Otherwise, the users' conceptual models will be modified in ways other than those intended by the interface designer.

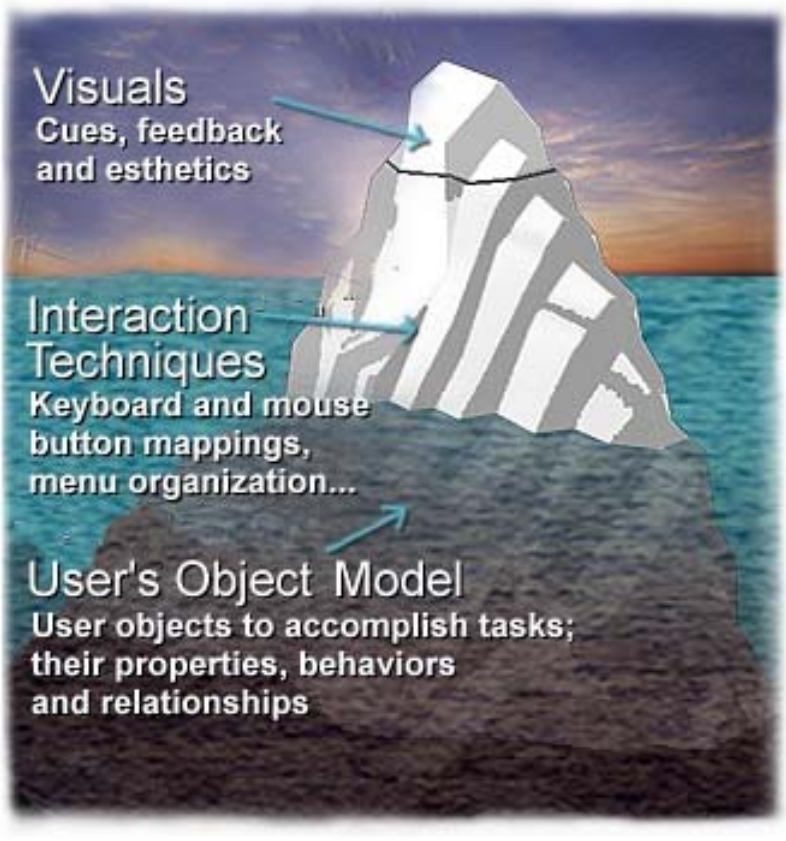
Designer

Specifying what we want the user experience to be - the designer's model

The interface components and relationships intended to be seen by users and intended to become part of each user's conceptual model are described in the *designer's model*. This model represents the

designer's intent in terms of components users will see and how they will use the components to accomplish their tasks.

The designer's model identifies objects, how those objects are represented to users, and how users interact with those objects. User oriented objects are defined in terms of properties, behaviors, and relationships with other objects. Differences in properties and behaviors are the basis for class distinctions, such as the distinctions between folders and documents. Relationships between objects affect how they are used in accomplishing users' tasks. For example, users can use folders to contain and organize memos, reports, charts, tables, and many other classes of objects. Users can discard an object by dragging and dropping the object's icon on a wastebasket icon, and users can print an object by dropping the object's icon on a printer icon. These actions are logical in that they maintain real-world relationships between objects.



Traditionally, the "user interface" of a product has been considered to be the "look and feel" aspects of the product. More recently, the emphasis has been shifting to the objects needed by the user to accomplish their tasks. This is the basis of an object-oriented user interface, or OOUI. A more complete picture of the designer's model is represented in the Designer's Model Iceberg, in which the look and feel aspects of the interface constitute the "tip of the iceberg", and the most important aspects are encapsulated in the object model. An object model describes objects, properties, behaviors, and relationships intended to be seen and used by users of the interface. The object model is the main component of the designer's model in an object-oriented user interface. The "look and feel" aspects play a supporting role.

By relying on a few basic classes and relationships, with well-defined distinctions based on user task needs, the designer's model should be easy for users to learn and understand. Users should quickly develop conceptual models that closely match the designer's model.

The Workplace OOUI Model is an example of a designer's model. This model defines objects that are common to many types of applications. Product designers add objects that are needed by specific products. This is typically done by extending existing objects (creating subclasses) or by defining entirely new types of objects (creating new classes). Definition of the designer's model is crucial to developing products that are easy to learn and understand. Its definition should comprise the first series of steps during product design.

If the designer's model closely matches a user's conceptual model, the user should learn quickly and apply knowledge correctly in new situations. In other words, the user will feel the interface is intuitive. Designers can help users to develop a closely matching conceptual model by creating a clear and concise designer's model. A designer's model is clear and concise when it has made a minimum number of distinctions among objects, the distinctions are clear and useful to users, and they are consistently conveyed throughout the interface.

For the designer's model to be consistent with the user's conceptual model, the designer must know the users, their tasks, and their expectations. If designers do not understand their users, the interface will not behave as users will expect it to. If the system does not behave as users expect it to, their conceptual models will be different from the designer's model and misunderstandings will occur. Users can lose confidence in the reliability of their conceptual model, and thus in the system itself, when these misunderstandings occur. If users form an incorrect conclusion or a superstition to explain an inconsistency, they may try to apply it elsewhere in the system. This can lead to further misunderstandings and distrust of the system.

A misunderstanding can be caused by inconsistency in an object's behavior resulting from a particular action. For example, if a user learns that *double-clicking* the mouse button on an object opens a window, and elsewhere in the interface the same action discards an object instead, the user will quickly learn to distrust the system.

In summary, the designer's model is the model of objects, properties, behaviors, and relationships that the designer intends the user to understand. The designer's goal is that each user's conceptual model exactly matches the designer's model. Users who perceive the interface at this level have a precise understanding of the interface and can take full advantage of the capabilities intended by the design.

Implementor

Conveying a complete and unambiguous design to the implementors - the implementor's model

The *implementor's model* describes the system internals used to implement the designer's model. The implementor's model includes details relevant only to programmers and others who develop the product.

For example, the designer's model might include a directory object that consists of people's names, addresses, office numbers, and so forth. The implementor's model of the directory object might consist of records in a file, with one record for each directory entry; or, it could be a complex organization of multiple records from multiple files.

Users should be shielded from the complexity of the implementer’s model, and can be because the designer’s model should not convey the technical implementation details.