User Interface Principles

FEBRUARY 11, 2004

Contents

List of Tables

List of Figures

Chapter 1

Principles

AUTHOR: C.B. HOVEY

CREATED: 03 NOVEMBER 2003

REVISED: 06 NOVEMBER 2003

1.1 Introduction

The purpose of this document is to examine the principles of human-computer interaction. This **Principles** document, in conjunction with a **Style Guide**, a catalog of **Use Cases**, and **Specifications**, constitute a four-prong approach to guide developers in the creation of MSC.Simulation Office products, built on the MSC.Simulation Office Framework (also known as "MSCx"). Figure 1.1 shows these four components in the context of a MSC.Simulation product.

1.1.1 Why are Principles Important?

The study and application of principles are important because they benefit all parties with an interest in the software application:

- **To the developer**—Principles help developers design user interfaces with the user in mind, less from a "user's functional needs" point-of-view and more from a "user's

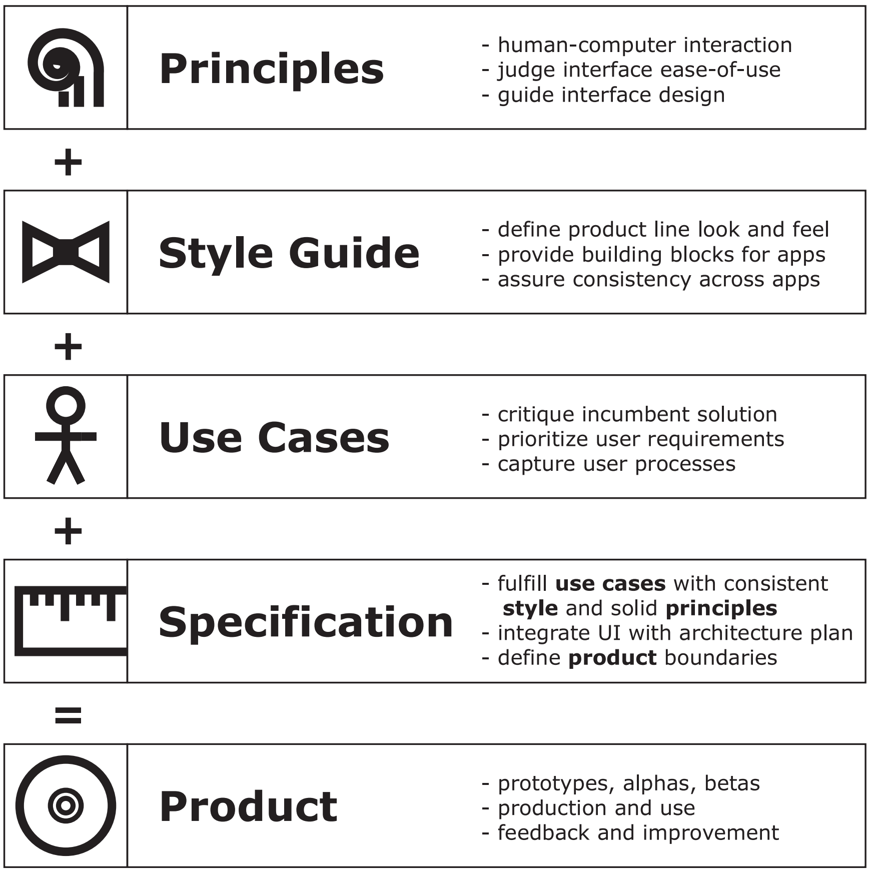

Principles
- human-computer interaction
- judge interface ease-of-use
- guide interface design
+
Style Guide
- define product line look and feel
- provide building blocks for apps
- assure consistency across apps
+
Use Cases
- critique incumbent solution
- prioritize user requirements
- capture user processes
+
Specification
- fulfill **use cases** with consistent **style** and solid **principles**
- integrate UI with architecture plan
- define **product** boundaries
=
Product
- prototypes, alphas, betas
- production and use
- feedback and improvement

FIGURE 1.1: Illustration of a product built for MSC.Simulation Office, composed of **Principles**, **Style Guide**, **Use Cases**, and **Specification**.

cognitive ability" point-of-view. These principles, founded in the study of human-computer interaction, treat the user first as a human being and second as a trained professional using a software application. The emphasis on the human element stems from the notion that if we wish to produce well designed software, we must first understand for whom the software is being designed. Human characteristics such as memory, concentration, and habits all play into the human-computer interaction. Observing these characteristics leads to an understanding of the cause-effect relationship between good design and satisfied users (or, as is more often demonstrable, bad design and unsatisfied users). Ultimately, principles form a basis that enables developers to evaluate and improve their designs, leading naturally to a high quality user interface.

- **To the user**—On the most basic level, principles are the catalyst that convert a first-cut design into a quality design that is easy-to-use. When users perceive that a software application is difficult to use, developer ignorance of one or more of the underlying user interface principles is often the cause. Designing user interfaces from a principles-first point-of-view often translates into a user's perception that the application is user-friendly. Principles, however, do not play an exclusive role at guiding user interface design. On level with principles, feedback from active users who frequently depend on the software should be factored into a second design iteration. Often developers devote most of their effort to *creating* the software, and little time to *using* the software, particularly in "real life" situations. While principles provide a solid framework from which to embark on design, user feedback provides the input used to refine it.

- **To the business**—Finally, the business venture is interested in making a profit from the design and production of the software application. If the application is designed well, it will be easy-to-use. If the application is easy-to-use, user acceptance will be high and the user install base will grow. Concomitant with this growth will be increased revenues (and hopefully) increased profits. Though the link between a quality user interface and profitability might seem logical enough, I think it is interesting to cite some empirical data from [Nielsen, 2002] to buttress this claim:

  – "When a certain rotary dial telephone was first tested, users were found to dial fairly slowly. A human factors expert spent one hour to come up with a simple graphical interface element which speeded up users' dialing behavior by about 0.15 seconds per digit, for a total annual savings of about $1,000,000 in reduced demands on the central switches." [Nielsen, 2002]

  – "An Australian insurance company has annual savings of A$536,023 from redesigning its application forms to make customer errors less likely... The cost of the usability project was less than A$100,000. The old forms were so difficult to fill in that they contained an average of 7.8 errors per form, making it necessary for company staff to spend more than one hour per form repairing the errors." [Nielsen, 2002]

– "A major computer company saved $41,700 the first day the system was in use by making sign-on attempts faster for a security application. This increased usability was achieved through iterative design at a cost of only $20,700..." [Nielsen, 2002]

While the returns on investment (ROI) in the aforementioned studies are compelling, I think it is important to emphasize that committing to the development of a quality user interface consumes resources, both in time and money. Nielsen cites a study published in the *Communications of the ACM* that pegs the cost to include human factors in to software development as $128,330 per year (in 1988 dollars). Also included in [Nielsen, 2002],

> A study of software engineering costs estimates showed that 63% of large software projects significantly overran their estimates... When asked to explain their inaccurate cost estimates, software managers cited 24 different reasons and, interestingly, the four reasons that were rated as having the highest responsibility were all related to usability engineering: frequent requests for changes by users, overlooked tasks, users' lack of understanding of their own requirements, and insufficient user-analyst communication and understanding.

I am reminded of that old business adage shown in Fig. 1.2. If I had to choose, and the

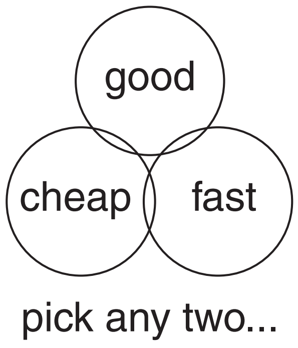


FIGURE 1.2: The business predicament: GOOD, CHEAP, FAST: PICK ANY TWO.

context was my lifeblood software that would be in the market for many years to come, I would always try to pick GOOD as half of my selection.

My final thought on this subject is simple: Quality user interface design is achievable, but it doesn't come for free. Design must be made a priority, on par with code architecture and implementation, quality and regression testing, and responding to customers' requests for new and enhanced features.

1.1.2 What Are the Open-Ended Questions?

Although this document attempts to put a framework around the elements that contribute to both good and bad design, it cannot speak as a final authority; design is too subjective of a field for such black-and-white clarity. Moreover, no design text, regardless of how thorough, can supplant human opinion and the decision making process. Therefore, one of the recurring open-ended questions will be

- *Is this the best design?*

While it may be difficult to answer that question definitively, we may apply design principles to assist in the creation of what comfortably feels like one of the best designs. For example, the **Efficiency** and **Vocabulary** sections (Sections 1.8 and 1.25, respectively) offer concrete examples of how competing design alternatives can be measured against each other to produce a theoretical efficiency of each design. Such a quantitative evaluation can either "tip the scales" in favor of one of the design alternatives; or, expose bottlenecks and thus opportunities to make a particular design more efficient. As another example, the sections on **Accessibility** and **Neutrality** (Sections 1.4 and 1.18, respectively) provide concrete examples of how to make the software interface appealing and sensitive to all people across the globe.

A second, and perhaps more tractable, open-ended question remains:

- *While it is all well and good to create a Principles document, how will anyone ensure that developers of an application built for MSC.Simulation Office take heed of it?*

To date, the MSC.Simulation Office Principles Subcommittee has been unable to answer this question with certainty. Suggested answers have included

- Have reading and applying the Principles document be part of the job description. *Counterpoint: Isn't this just a fairly soft type of enforcement?*

- Have a review committee oversee, evaluate, and if required, redesign all user interface proposals for MSC.Simulation Office. *Counterpoint: While this might improve quality, are the quality gains worth the bureaucratic slowdowns involved with a committee review process?*

- Develop a MSC.Simulation Office user interface widget library composed strictly of abstract base classes and interfaces from which all vertical application GUIs built for MSC.Simulation Office must derive. *Counterpoint: If the base classes are capable of being useful to the application, then they are probably useful enough to be misused as well.*

1.2 Overview

What follows is a discussion of various user interface principles, organized by a name roughly agreed to by the user interface design community [Raskin, 2000, Nielsen, 2002].

Most of the high priority topics have been covered in detail, while lower priority topics will be completed at a later time. The topics are arranged in alphabetical order for no reason other than ease of finding a particular principle from the list.

1.3 To-Dos

The following items remain as work to complete on the Principles document:

- Revisit the **Modes** section and include feedback from Klinger, Cooper, and Goodson.

- Finish medium priority topics.

- How to bridge the gap between Principles and Style Guide, and the principles and actual developer exercising the principles during implementation.

- Finish low priority items.

1.4 Accessibility

AUTHOR: K. VINCENZ

CREATED: 17 OCTOBER 2003

REVISED: 22 OCTOBER 2003

1.4.1 Building Accessibility into Software for the Physically Disabled

As a nation, we have already removed barriers for the disabled from everyday life, such as making ramped curbs, setting aside parking spots, and installing elevators in buildings. We also must be sure to build accessibility into our software. In the U.S. alone, there are an estimated 30,000,000 people who cannot use a computer because of its inaccessible design. At MSC, we want to remove barriers, not build them.

1.4.2 Types of Disabilities

Disabilities fall into these categories:

- **Visual Impairments**—Dim or hazy vision, extreme far- or near-sightedness, color blindness, tunnel vision, and complete blindness.

  More on Color Blindness— The most common form of color blindness is red-green (the inability to distinguish red and green). The second most common form is blue-yellow, and it is almost always associated with a red-green deficit. The most severe form of color blindness is *achromatopsia*, the inability to see any color.

- **Movement Impairments**—Difficulty moving the mouse.

- **Hearing Impairments**—Ranges from being able to hear some sounds but not distinguish spoken words, to deafness.

- **Cognitive and Language Impairments**—Ranges from dyslexia to difficulties remembering things, solving problems, or comprehending and using spoken or written language.

- **Seizure disorder**—Certain light or sound patterns can cause epileptic seizures in some susceptible users.

1.4.3 Why is Accessibility So Important?

Besides ensuring that everyone can use your software, there are two Federal government acts that have increased the focus on making software accessible, which are listed below. Though these acts apply directly to Federal agencies and public institutions, they also apply to any company from which these agencies and institutions procure services.

- Section 504 of the 1973 Rehabilitation Act and the 1990 Americans with Disabilities Act state that "No otherwise qualified individual with a disability shall, solely by reason of his/her disability, be excluded from the participation in, be denied the benefits of, or be subjected to discrimination under any program or activity of a public entity."

- Section 508 (Information Technology and People with Disabilities) amendment to the Workforce Investment Act of 1998 "requires that Federal agencies' electronic and information technology is accessible to people with disabilities, including employees and members of the public."

  Section 508 is a set of functional and technical requirements applicable to electronic and information technology, such as software applications, Websites, video and multimedia, and documentation and support products. For the software requirements, see Section 1.4.5 at the end of this section.

  The IRS provides a series of e-learning modules to educate software engineers on Section 508. See the e-learning modules (which are free). The IRS summarizes what you should do to ensure accessibility in your software the following two ways:

  > As a developer, you have two methods to ensure your software conforms to Section 508. First, you can make the software compatible with assistive technology. Second, you can make the software product completely accessible without the aid of other assistive technology. You should

make sure that whichever method you choose, your product does not interfere with adaptive equipment. For example, do not make your software product use a certain size font, which will not allow a piece of adaptive equipment such as a screen magnifier to efficiently increase the font size.

For more information on Section 508, see www.access-board.gov.

1.4.4 Guidelines for Building Accessibility into Your Applications

The guidelines below were gathered from the following sites:

- Microsoft MSDN Library (http://msdn.microsoft.com/library/default.asp)
- GNOME Accessibility Project (http://developer.gnome.org/projects/gap/)
- Section 508 US Government Site (http://www.usdoj.gov/crt/508/508home.html)
- Web Site Accessibility Class (http://www.lgta.org/accessibility/)

1.4.4.1 KEYBOARD NAVIGATION Note: For a very good overview of keyboard interface design, see http://msdn.microsoft.com/library/default.asp and then select **Accessibility** > **Technical Articles** > **Guidelines for Keyboard User Interface Design**. Alternatively, assuming Microsoft hasn't rearranged its website (which it does from time to time), go directly to the webpage. Design a keyboard user interface that helps users with visual and movement impairments.

- Provide access to all application features through the keyboard. This also satisfies many power users.
- Because typing is difficult or even painful for some users, minimize the number of keystrokes required for any given task.
- Use a logical keyboard navigation order.
- Don't assign awkward reaches to frequently performed keyboard operations.

- Don't require repetitive use of simultaneous key presses. Some users are only able to press and hold one key at a time.
- Ensure that objects that can be resized or moved by drag and drop can also be resized or moved with the keyboard. For precision sizing and placement, provide a dialog into which users can type coordinates or snapping of objects to a user-definable grid.

1.4.4.2 KEYBOARD FOCUS There should never be any confusion as to which control on the desktop has focus at any given time. This is particularly helpful for assistive technologies (ATs).

- Start focus at the most commonly used control.
- Show current input focus clearly at all times.

1.4.4.3 MOUSE INTERACTION Not everybody can use a mouse with equal dexterity, and some users may have difficulty seeing or following the mouse pointer.

- Don't depend on input from mouse button 2 or button 3, which is physically more difficult for the disabled to click.
- Don't warp the mouse pointer, or restrict mouse movement to part of the screen. This can interfere with ATs, and is usually confusing even for users who don't rely on ATs.

1.4.4.4 GRAPHICAL ELEMENTS Provide options to customize the presentation of all the important graphical elements in your application. This will make it easier for people with visual or cognitive impairments to use.

- Don't hard code graphic attributes such as line, border, or shadow thickness. These elements should be read from the window manager. If this is not possible, provide options within your application to change them.
- Allow multi-color graphical elements (e.g., toolbar icons) to be shown in monochrome only, if possible. These monochrome images should be shown in the system fore-

ground and background colors, which users can chose themselves for maximum legibility.

- Make interactive interface elements easily identifiable. For example, do not make the user hover the mouse over an object to determine whether it is clickable or not. Leave sufficient space between objects and clearly delineate object boundaries.
- Provide an option to hide graphics that don't convey essential information. Graphical images can be distracting to users with some cognitive disorders.

1.4.4.5 FONTS AND TEXT Even to a user with normal vision, textual output provides the majority of the information and feedback in most applications. Choose and position text carefully on the screen, and let users choose font and size so users with vision impaiments can also use application effectively.

- Don't hard code font styles and sizes. If you have to, never hard code any font sizes smaller than 10 points.
- Label objects with names that make sense when taken out of context. Users relying on screen readers or similar ATs will not necessarily be able to immediately understand the relationship between a control and those surrounding it.
- Position labels consistently throughout your application.
- When you use static text to label a control, end the label with a colon. For example, Username: to label a text field into which the user should type their username. This helps identify it as a control's label rather than an independent item of text.
- When you use static text to label a control, ensure that the label immediately precedes that control in the Tab order. This will ensure that the mnemonic (underlined character) you assign to the label will move focus to or activate the correct control when pressed.
- Use San Serif fonts for screen display. Studies show that Verdana and MS Sans Serif are the most readable, though it varies depending on the point size.

1.4.4.6 Capitalization

- For greatest readability, use standard capitalization rules—for example, common nouns are usually all lowercase and proper nouns are always capitalized.
- Avoid the use of capitalized letters because:
  - When too many words are capitalized, they lose their importance and no longer attract attention.
  - Readability studies have shown that text is more easily read when it doesn't contain excessive use of initial caps or all uppercase letters (*tile* is easier to read than *TILE*).
  - Never use all uppercase letters for emphasis, for example in an error message. This is considered yelling, and is rude.
- Using lowercase letters in no way diminishes the stature or credibility of an object or a concept. After all, even the title "president of the United States" is lowercased when it doesn't immediately precede the office holder's name.

1.4.4.7 Color and Contrast Users with visual impairments require a high level of contrast between the background and text colors.

- Don't hard code application colors. Some users need to use particular combinations of colors and levels of contrast to be able to read the screen comfortably.
- Do not use colors by themselves to convey meaning or give directions to users. For example, do not use plain red buttons to imply stopping or green buttons to imply continuing. Use at least one other method, such as shape, position, or textual description. In the example shown in Fig. 1.3, arrows help convey the stock price value as do the colors:
- Maximize the color contrast between the text and the background (and do not use background patterns).

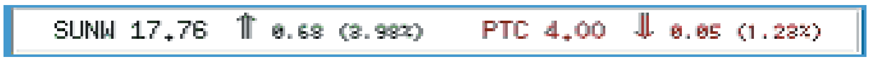

SUNW 17.76 ⇑ 0.68 (3.98%) PTC 4.00 ⇓ 0.05 (1.23%)

FIGURE 1.3: Arrows, in addition to color, help convey the stock price. Were only colors used (without arrows), the user interface would convey less information to users with red-green color blindness than to users without such blindness.

– Do not use [red, green, brown, gray, purple] next to, on top of, or changing to [red, green, brown, gray, purple] .

  The combinations shown in Fig. 1.4 would be difficult for a red-green color blind user:

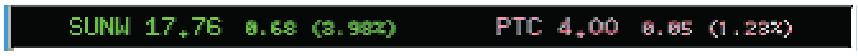

SUNW 17.76 0.68 (3.98%) PTC 4.00 0.05 (1.23%)

FIGURE 1.4: Illustration of a color combination that would be difficult to read by a user having red-green color blindness.

– Try viewing your page with a monochrome monitor setting. If the elements of the page are clear there, a color-deficient person will probably be able to use it.

A good reference on color contrast is http://www.lighthouse.org/color_contrast.htm.

1.4.4.8 MAGNIFICATION

Many users, even those not visually impaired, benefit from magnification of text and graphics. Without magnification, a visually impaired user may not be able to access and use the program at all.

- Provide the ability for the user to magnify the work area.
- Provide options in the application to scale the work area. Users need to have an option to magnify the work area 150% to 400% or more.

1.4.4.9 AUDIO

- Do not convey information through audio only. As with color, make sure that the user is able to have any audible information conveyed in other ways, such as using visual equivalents (text captions or symbols).

1.4.4.10 Animation

- Don't use flashing or blinking elements having a frequency greater than 2 Hz and lower than 55 Hz.
- Don't flash or blink large areas of the screen. Small areas are less likely to trigger seizures in those susceptible to them.
- Make all animations optional. The animated information should be available in at least one non-animated format, at the user's request.

1.4.4.11 Documentation Not only does your software application need to be accessible, but so does the information helping the user use the application (guides, demos, and help files).

- Provide documentation in an accessible format.
  - HTML is an excellent format for ATs and there are many simple guidelines available for ensuring the HTML code you provide is accessible. There are also free testers available to rate your HTML's accessibility, such as .
  - Tagged PDFs created with Adobe Acrobat 6, allow users to synthesize the document as text or read it using a screen reader. In addition, in Adobe Acrobat 6, you can change the font of the text and the bookmarks on the left. For more on Adobe Acrobat 6.0 accessibility features, see: http://www.adobe.com/products/acrobat/access_faq.html.
- Document all the application's accessibility features. Keyboard navigation and shortcuts are particularly important to document. Include an accessibility section in the documentation.

Note: The Vertical & Modelers documentation group is working on accessibility standards for its documentation, which it will publish in its Style Guide.

1.4.5 Section 508 Software Provisions

1. Section 508 Software Provisions

(a) When software is designed to run on a system that has a keyboard, product functions shall be executable from a keyboard where the function itself or the result of performing a function can be discerned textually.

(b) Applications shall not disrupt or disable activated features of other products that are identified as accessibility features, where those features are developed and documented according to industry standards. Applications also shall not disrupt or disable activated features of any operating system that are identified as accessibility features where the application programming interface for those accessibility features has been documented by the manufacturer of the operating system and is available to the product developer.

(c) A well-defined on-screen indication of the current focus shall be provided that moves among interactive interface elements as the input focus changes. The focus shall be programmatically exposed so that assistive technology can track focus and focus changes.

(d) Sufficient information about a user interface element including the identity, operation and state of the element shall be available to assistive technology. When an image represents a program element, the information conveyed by the image must also be available in text.

(e) When bitmap images are used to identify controls, status indicators, or other programmatic elements, the meaning assigned to those images shall be consistent throughout an application's performance.

(f) Textual information shall be provided through operating system functions for displaying text. The minimum information that shall be made available is text content, text input caret location, and text attributes.

(g) Applications shall not override user selected contrast and color selections and other individual display attributes.

(h) When animation is displayed, the information shall be displayable in at least one non-animated presentation mode at the option of the user.

(i) Color coding shall not be used as the only means of conveying information, indicating an action, prompting a response, or distinguishing a visual element.

(j) When a product permits a user to adjust color and contrast settings, a variety of color selections capable of producing a range of contrast levels shall be provided.

(k) Software shall not use flashing or blinking text, objects, or other elements having a flash or blink frequency greater than 2 Hz and lower than 55 Hz.

(l) When electronic forms are used, the form shall allow people using assistive technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.

1.5 Affordance

AUTHOR: S. GOODSON

CREATED: 23 SEPTEMBER 2003

REVISED: 06 OCTOBER 2003

Note: Sections containing only an outline of bulleted items (in place of written text with discussion and figures) signifies that the section is currently under development.

Definition "The perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used." [Norman, 1989, Norman, 2002a, Norman, 2002b].

For example, a door handle provides both

- a convenient place to grab and pull (a physical affordance), and
- an indication that pulling is a useful action to perform (a perceived or cognitive affordance).

In order to effectively use a program (or any other tool), a user must have some way of determining which actions are useful, meaningful actions to perform, and of predicting the consequences of those actions. In designing screen based applications, we have little control over physical affordances; it is perceived affordances that matter. The 3-D appearance of a button shouts "push me," and scroll-bars look like they might slide, but in general a perceived affordance is the result of a shared convention. The user must rely on either experience or training to know that a scroll-bar controls the position in the document, and that a menu button provides a list of additional choices.

Violating these conventions by putting a "pull-style" handle on a door that must be pushed, or by using a menu bar button to perform some other action, only serves to frustrate and confuse the user.

Controls should

1. be easily perceived and interpreted

2. follow standard conventions

For additional reading on Affordance, see

- [Norman, 1989]
- [Norman, 2002a]
- [Norman, 2002b]
- [Cooper, 2003]

1.6 Branding

AUTHOR: K. VINCENZ

CREATED: 04 NOVEMBER 2003

REVISED: 10 NOVEMBER 2003

Branding associates a set of images or words with a product, service, or company to create a representation or perception of the company and its products in the mind of customers. Branding enables a company to own a word (Windows, Nike, Google)—a brand. The brand becomes an intangible asset of the company that contributes millions or even billions of dollars to a company's stock price and market valuation (estimates in the Financial Times (1999) valued the Coca-Cola brand at $83 billion; McDonald's at $26 billion). Branding increases a company's revenues, lowers cost of acquiring new customers, increases customer and employee retention, and increases profitability. As we define and create the SimOffice suite of products, we must create a powerful brand in the minds of our customers, and provide them with a compelling and consistent experience using those products, that enforces the brand's value proposition.

1.6.1 Defining and Building a Brand

Building a strong brand is a complex task. There are two ways to build a brand:

- **Directly** through customer experience (such as, customers actually using the product or going to a restaurant). To create a brand directly, you must create a compelling and consistent customer experience that satisfies customers and keeps them coming back.
- **Indirectly** through advertising (such as, TV commercials, magazine ads, and billboards) and sponsoring events that are fun or interesting to customers so customers associate the experience of the event with the brand. Indirect branding requires repeated exposure (must be received at least 10 times to be effective).

To create a brand, a company must develop a good understanding of who their customers are and what they value. Then, the company must:

- Determine how the products, services, and company should be organized to deliver this value.

- Develop the capabilities and resources (staff, culture, vendor relationships, and so on) to support and deliver the value of the brand.

Michael Young, head of MSC Marketing and Customer Satisfaction, defines the value proposition behind the SimOffice brand as

> [We want to] use Simulation Office to change the game on our competitors by establishing it as the broadest, most powerful, best integrated, smartest-licensed (MSC.MasterKey) suite of tools out there, backed up by our support and our services (which are focused on improving our customers' product development processes).

1.6.2 Why Ensure Branding

Some of the benefits derived from branding include:

- **Price premium**—Because of the associations that exist with brands (quality, status), customers are willing to pay a higher price for the brand than for others.

- **Satisfaction, loyalty, and trust**—If customers have high levels of satisfaction with the brand, they become loyal to that brand, which again helps prevent price sensitivity. In addition, if customers come to trust a brand, because they have had consistent experiences with it, customers will be more likely to automatically select that brand.

- **Accept sister products**—If customers had good experiences with products in a brand, they are more likely to purchase new products introduced into that brand. For example, if a customer has purchased a Hewlett-Packard (HP) calculator, and feels the brand stands for quality, the customers will more likely purchase an HP printer over another, possibly less expensive, printer.

- **Faster decision-making**—If the brand is trustworthy, it reduces anxiety and doubt in the mind of the customer. It makes their decision making easier and safer. Customers tend to economize on information, only gathering enough information to help them make a decision. Branding helps customers make those decisions.

1.6.3 Guarding the Brand

Unless a brand is managed well, it can:

- **Become diluted**—Over the years, as companies add more products into the same brand and at different cost points, their brand can loose its power to invoke images. What does it mean to buy a Ford, when Ford has a car in every entry level and has many different types of vehicles?

  To avoid this, many companies create new brands for the products they enter in new market niches. Rolex created the less-expensive Tudor brand while Honda created the upscale Acura brand.

  We must ensure that the SimOffice brand doesn't become diluted as we enter new products into new segments of the virtual prototyping market.

- **Become uncool**—There are many reasons for this including:
  - Cultural change (Aunt Jemima seems racist now)
  - Challenger brands move into space (Starbucks over Maxwell House Coffee)
  - Perceived as outmoded, old-fashioned, or no longer hip (Levi's, Campbell's)

- **Have negative associations**–The most well known example of this is Tylenol, after its packaging was tampered with in the 1980's. McNeil-PPC, the parent company, was able to resurrect the brand using advertising and public relations campaigns. Some of MSC's brands also have negative associations and even associations with the companies from which the products originated. MSC products are often perceived as expensive, hard to use, or second to other products.

According to Johan Kramar, cofounder and creative director of KesslsKramer advertising agency:

> Companies have to keep reinventing and revitalizing the messages behind their brands—not killing them but reincarnating them.

We have this opportunity with SimOffice, to integrate our products into one suite, improving all of their interfaces, while still clearly differentiating between them and keeping the positive assocations of their current branding.

1.6.4 Changing Nature of Brands

From an interview from Fast Company magazine, Fast Pack, 1999: *JOHN HAGEL, chief strategy officer at entrepreneurial operating company Twelve Entrepreneuring in San Francisco. Author of Net Worth: Shaping Markets When Customers Make the Rules (Harvard Business School Press, 1999).*

> We're going to see a new kind of branding emerge, a much more customer-centric branding where the promise is, "I know you as an individual customer better than anyone else, and you can trust me to assemble the right products or services to meet your individual needs."

Scott Bedbury: Helped build both Nike and Starbucks into brand leaders before launching Brandstream, his own branding consultancy. Author of A New Brand World: 8 Principles for Achieving Brand Leadership in the 21st Century (Viking, March 2002), written with Stephen Fenichell.

> How can you differentiate your brand? To me, the answer is that being different is ultimately about values–your values, your brand's values, and your company's values. Your values are what set you apart. Starbucks is built on Howard Schultz's values. Howard promised himself that he would create a company that would never leave its employees behind. And when he took over Starbucks in 1987, he set out to provide stock ownership and health and medical benefits for all of his half-time employees.
>
> Today, when you look at the quick-serve restaurant industry, Starbucks is different from almost all the others – and it's not just because a place like Burger King fills bellies, whereas a good coffee house fills the soul. It's also because

companies like Taco Bell have 250% to 300% turnover among their hourly employees and 100% to 150% turnover among their managers. The turnover rates at Starbucks are a fraction of that.

1.6.5 Global Branding

According to O'Shaughnessy, writing in *The Marketing Power of Emotion*, customers have loyalty to their national local brand. RJR Nabisco has been more successful selling a new brand of cigarette called Peter 1 in Russia, than they have been selling Camel or Winston. In addition, in India, Coca-Cola's best selling brand is Thums Up, not Coke. MSC must be sensitive to the cultural dimensions and market requirements of global marketing on the SimOffice brand.

1.6.6 Recommendations on Naming

Shakespeare claims that a "rose by any name would smell as sweet," but we all know the power of names. Brand names can be symbols of status, prestige, reliability, and quality. Names like Infiniti suggest durability; Acura, precision and accuracy; Raliart, fun and rally sports. For durable goods, a good name can allow a company to charge 20% more for their product.

- The name of a product should not change.
- The name should be distinctive, memorable, and easy to pronounce.
- A name speaks for the company, breaking down misconceptions.

1.6.7 Enforcing Branding in the User Interface

The average MSC customer will usually not see an MSC representative on a daily or even monthly basis, probably not until the customer attends a user conference or, if the customer is in management, decides to purchase additional products. Instead, customers will interact everyday with their MSC user interface. That interface will provide customers with their emotional experience with MSC. It is the major tool through which we enforce the MSC brand.

According to Nate Fortin, writing at the Cooper Web site, one company that has integrated branding throughout its user interface is Apple:

> To their credit, a few organizations have already begun to leverage the branding opportunities offered by software. Take Apple Computer for example. Apple has successfully built brand equity within many of its software user interfaces. Apple's QuickTime Player leverages identity elements found throughout the Apple communications platform, including the operating system, the corporate Web site, hardware products, and printed collateral. Each one of these applications serves as an individual voice in harmony with the larger concert, communicating a consistent message of quality and innovation regarding Apple products (see Fig. 1.5).

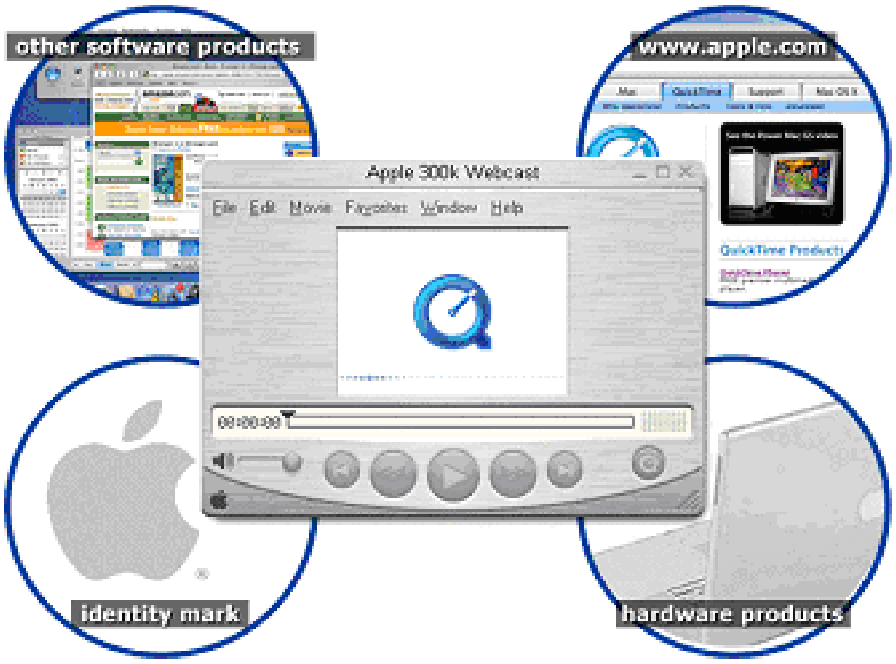


FIGURE 1.5: Apple's Quicktime Player is an example of software that builds brand equity.

1.6.8 Start Early

Early in the development process of SimOffice, MSC must define the branding images and words for SimOffice to ensure that they are integrated throughout the user interface, and not just pasted on top, such as in the opening splash screen. In addition, by defining the branding early, it ensures that developers will not need to make extensive changes late in the development cycle, such as searching for strings with an old product name to change it to a new name, at a time when the name may be in several thousand files.

1.6.9 Elements of Branding in the User Interface

Here are a few aspects of branding that we must be sure to do correctly throughout the user interface:

- The MSC logo and tag lines are used consistently, and placed correctly, with correct trademarking.
- Product naming, MSC, periods (.), and slashes (/) are used correctly and consistently.
- Correct color palette is used consistently.

  Note on Color from Nate Fortin: Another factor that you must consider when using color to brand the user interface is how it affects behavior. This is often another area that requires some rethinking when trying to apply guidelines from a corporate brand manual. Again, most of the brand documentation that exists today is not designed with technology-based products in mind. One of the clearest examples of this can be found with companies that have specified a bright red color in their primary brand pallet (which by my last count are quite a few). While this can be quite effective when applied to a box of cereal, a lawn-tractor, or a product brochure in which the name of the game is to stand out from the competition, it can present significant challenges in the context of the user interface.
- Correct use of icons and typography.
- Consistency across all products and plug-ins in the SimOffice suite, enforcing the branding message that SimOffice is an integrated suite of tools, and ensuring that customers can easily move from one product to the other. As new products are introduced into the suite, they must also be consistent with past products.

- Clear definition of the different products in the suite, preserving the positive value of existing brands and maintaining their identity. Customers should be able to clearly differentiate when they are using a particular MSC product (MSC.Nastran, MSC.Patran, or MSC.ADAMS), even as they use models or data from one product in another. This demonstrates that our products are individual pieces of an integrated whole.

- Well-written, accurate, and goal-oriented documentation (help, guides, embedded user assistance) that is consistent across all products, integrated together, and offers the customer a positive experience.

- Messages that teach and inform and are polite.

- Training that is examples-based, goal-oriented, consistent across all products, integrated together, and offers the customer a positive experience.

Again, as Nate Fortin noted:

> If your software behaves rudely, doesn't satisfy users' goals, and otherwise offers a poor user experience, it's rather difficult to transform that into a positive brand experience. Attempting to do so is what Alan Cooper refers to as "putting lipstick on a pig." Sure, you might fool some people long enough to make a sale, but in the long term you can cause tremendous damage to your brand.

1.6.10 Section Bibliography

1. *The Marketing Power of Emotion*, John and Nicholas Jackson O'Shaughnessy, Oxford University Press, 2003.

2. *The 22 Immutable Laws of Branding*, Al and Laura Ries, HarperCollins, September 2002.

3. Branding and User Interface, Part 1: Brand Basics, Nate Fortin, http://www.cooper.com/content/insights/newsletters/2003_04/Branding_Part01.asp.

4. Branding and the User Interface, Part 2: Tips on New Media Branding: Behavior and Color, Nate Fortin, http://www.cooper.com/content/insights/newsletters/2003_07/Branding_Part_2.asp.

5. Fast Pack 1999, Fast Company, February 1999, http://www.fastcompany.com/magazine/22/fastpack99.html.

6. Nine Ways to Fix a Broken Brand, Scott Bedbury, Fast Company, February 2002.

7. Determining How Design Affects Branding, User Interface Engineering, February 2002, http://www.uie.com/Articles/design_and_branding.htm.

8. Branding and Usability, User Interface Engineering, February 1999, http://www.uie.com/branding.htm.

1.7 Consistency

AUTHOR: K. VINCENZ

CREATED: 26 SEPTEMBER 2003

REVISED: 30 SEPTEMBER 2003

McDonald's built an empire on the idea of consistency. Go into any McDonald's and you were assured of the same quality and same type of food. You know what you are getting regardless of whether you are in Chicago, London, or Georgia.

This same idea applies to software design. If you start a Windows program, you know that the Exit is under the File menu and that Copy, Cut, Paste are under the Edit menu. You also know that if you double-click a word, it is highlighted. This type of consistency assures users in the same way that McDonald's does. I know what I am going to get, where to find, and I can get my job done more quickly and with more confidence because I do not have to relearn the basics of an interface.

A common reason for ignoring consistency is the idea that you have created an product that is so great and solves so complicated a problem that anyone would want to use it regardless of its inconsistencies. However, unless you've created a product that foresees the price of stocks or picks lottery numbers correctly, you had better be sure you're your product is consistent, and thereby easy to use, so your users will use it.

Another reason is that consistency stifles creativity and that you have found a better way to do things, better than what the Apple or Windows style guide says. Well, despite many redesigns, the order of keys on the keyboard has stayed the same; the keypad on the phone doesn't change. People get used to something and want to see it continue. They have more important things to do than relearn common tasks. Therefore, ensuring consistency in your interface matters to users.

The following is a summation of of the current literature available on providing consistency in user interface design. Some of the text is used verbatim from those sources. For a listing of sources, such as [Apple, 2002] [Berkun, 1999], [Nielsen, 2002], [Spolsky, 2001], and [Tognazzini, 2003] see the Bibliography.

1.7.1 What is Consistency?

- Consistency in an interface allows users to transfer knowledge and skills from one product to another. By doing so, consistency leads to ease of learning and ease of use.
- Consistency matches the user model, since the user's model is likely to reflect the way that he/she sees other programs behaving.
- Consistency is predictable, and people like predictable things. They feel comfortable when they can rely on different parts of a product to do exactly what they think it will do.
- Consistency applies to both within the individual application and across complete computer systems and even across product families.
- Consistency improves the user's productivity because it leads to higher throughput and fewer errors since the user can predict what the system will do and can reply on a few rules to govern the use of the product. The smaller number of errors and shorter learning time leads to improved user satisfaction and fewer frustrations.

1.7.2 Advantages to Consistency

All of the consistency advantages were taken from the book, *Coordinating User Interfaces for Consistency,* by Jakob Nielson [Nielsen, 2002].

- Ease of learning implied by consistency leads to lower training costs.
- Consistent set of software leads to reduced user support because users will not request help as frequently when all software is consistent. The support that has to be offered will be less expensive because support personnel will not have to learn so many interfaces themselves.
- The architecture for consistent interfaces forms a coherent basis on which the company can expand its suite of products. The products evolve in a controlled manner.

- Reduces development costs both with respect to designers (who don't have to design every aspect of the interface) and with respect to programmers (who may reuse code that implements standard aspects of the interface.

- In the same way, consistency will lead to reduced maintenance costs, first because the interface which is implemented in the first release is built on a solid foundation, and second because all systems following a given standard will evolve together, again leading to reuse of any design and code changes.

- Because of the ease of use and learning, consistent software is expected to lead to increased software use and sales.

- The interface style will lead to better product definition in the marketplace (the notion of product families) and to the development of market segmentation such that users who have already bought products are likely to buy additional products from the same vendor to ensure consistency with the interfaces they already know.

- Consistent software has the potential for leading to more aesthetic user interfaces because the different aspects of the interface comply with an underlying norm and because (presumably) a significant human factors effort has been invested in the design of the interface architecture. Designers may then build on this foundation and could be expected to produce more creative designs when they are allowed to focus on designing those aspects of a product that are unique to that product rather than having to reinvent every interaction technique needed for a complete interface.

- Consistent software simply fulfills a market demand since compliance with user interface conventions will in itself lead to better reviews of software in the trade press. In many cases, consistency of the user interface is one of the checklist items covered in magazine reviews of software packages.

1.7.3 Dangers of Consistency

- Even though a finished standard will save development costs for the individual product, arriving at a good standard is a costly process in itself, and the time and money to develop the standard must be invested up front before it can be used, thus possibly leading to delay for the products which are to use the standard (from *Coordinating User Interfaces for Consistency*, see [Nielsen, 2002]).

- Having a standard also implies an overhead in the software lifecycle to ensure conformance and some need for evaluating whether new user interfaces are in fact consistent. It may even be necessary to have some kind of enforcement in place. It is difficult to assess conformance with a standard in an area as fuzzy as user interfaces, so one may have to conduct experiments to get empirical verification of consistency (from *Coordinating User Interfaces for Consistency*, see [Nielsen, 2002]).

- A defined common user interface risks being a lowest common denominator to the extent that one wants to incorporate a large installed base of perhaps somewhat primitive interfaces. In any case, the common user interface might be stifling for innovation in new products and introduce resistance to change even where change is needed (from *Coordinating User Interfaces for Consistency*, see [Nielsen, 2002]).

- The very idea of consistency also implies reduced flexibility in the design of individual products so that they may not be able to be as tailored to application-specific requirements or contexts. In addition, a standard may not just prevent enhancements but it could even enforce bad design if it includes poor rules (from *Coordinating User Interfaces for Consistency*, see [Nielsen, 2002]).

- From an organizational perspective, having a corporate standard lessens motivation among developers if they feel that they do not share ownership of the user interface. And the very fact that one has a formalized program for consistency could mean that consistency can distract from other design issues, perhaps to the extent that developers feel that they do not need to take other human factors considerations in account as long as they follow the standard. By the same token, too much focus on consistency could promote foolish consistency in cases where good design would deviate from common user interface (from *Coordinating User Interfaces for Consistency*, see [Nielsen, 2002]).

- It is just important to be visually inconsistent when things must act differently as it is to be visually consistent when things act the same.
- Avoid uniformity. Make objects consistent with their behavior. Make objects that act differently look different.
- The most important consistency is consistency with user expectations.
- In rare cases, consistency can become a self-perpetuating monster: It has to be used for a purpose. A foolish consistency is one that serves no benefit for the end user. Making things look and work the same is pointless if the user can no longer accomplish their tasks. Rank making things useful above making them consistent. An example is interfaces for video games. Imagine your company was developing two video games, a driving game and a Pac-Man game. The best UI for the driving game would be a steering wheel, but the Pac-Man game would work better with a joystick and some buttons. Trying to design one UI to use for both of these games would be a disaster. At best you'd reach a middle ground that wasn't good at anything. Consistency applied to certain user tasks can make the user experience worse, not better. Consistency does not guarantee usability. It generally helps a user interface, but there are no guarantees in interface design. In this video game example you would have to choose between the user's cost of learning two different specialized UIs against learning one UI they could reapply but wasn't well suited to any of the tasks they wanted to do (from *Avoiding Foolish Consistency* [Berkun, 1999]).

1.7.4 Examples of Consistency

The most general examples of consistency are Microsoft Office products. How you open, save, and print a file are all the same. A user knows to expect to his/her goal of saving a document to be accomplished by looking under the File menu. If he/she wants Help, they can look under the Help menu. By making these products consistent, Microsoft ensured that you would find it easier to use their spreadsheet program than another company's spreadsheet program, because you had already learned their word processor. And no one wants to learn more than they need to get a job done.

1.7.5 Examples of Inconsistency

Inconsistencies within user interfaces are so prevalent that it's like shooting fish out of the water.

- We all have had the embarrassing experience of walking up to a door with a horizontal bar and pushing only to find out that the door should be pulled. But that is inconsistent. A horizontal bar on a door means you push while a handle means you pull.
- The US telephone pad is inconsistent with calculators and European telephone pads. On a US phone pad, 1 is at the upper left corner. On a calculator or European telephone, the 1 is in the lower left corner. Image fumbling late at night in a hotel in Denmark, trying to use the phone, only to dial the wrong number.
- We've all been frustrated when an error message asks you a Yes or No question, while another error message will ask you a OK or Cancel. Or, the OK button isn't the usual place so you keep selecting the incorrect button.
- In MSC.ADAMS, we have many terms for the same concept. MSC.ADAMS uses simulation, analysis, and event all meaning to run ADAMS/Solver and solve the equations of motion. Another examples are test data, actual data, physical data, given data, which mean just one type of data. You should pick the most descriptive set of terms and use it throughout.
- In Adobe Reader, when you select Find (Ctrl-F), the dialog box appears, and once you enter the text you want to search for, the dialog box disappears. To search again, you have to use Ctrl-G. If you select Ctrl-F again because you can't remember Ctrl-G, you have to fill in the dialog box again. In Microsoft Office products, however, the Find (also Ctrl-F) dialog box stays on the screen so you don't have to use Find Again. Since Ctrl-G is inconsistent, users fail to use or remember it.

1.7.6 Consistency Check List

Ask yourself the following questions when thinking about consistency in your product.

- Is your product consistent within itself and with other MSC products?
- With earlier versions of your product?
- With the standards of platform on which it will operate? For example, does your application use the reserved and recommended keyboard equivalents?
- Its use of metaphors?
- With people's expectations? Matching everyone's expectations is the most difficult kind of consistency to achieve because an audience with a wide range of expertise probably uses your product. You can address this problem by carefully weighing the consistency issues in the context of your target audience and their needs.
- Are the same types of objects in the same area?
- Are the same techniques used for the same types of actions?
- Are you using a real-world metaphor?

1.7.7 Measuring and Ensuring Consistency

The most important aspect of consistency is to define what it means in MSC products and find a way to measure and enforce that consistency.

1.8 Efficiency

AUTHOR: C.B. HOVEY

CREATED: 21 OCTOBER 2003

REVISED: 30 OCTOBER 2003

1.8.1 Definition

The concept of **efficiency** in the context of human-computer interaction can be defined as

$$\text{efficiency} = \frac{\text{task completion}}{\text{time}}. \tag{1.1}$$

For a given unit of time, the more tasks that can be completed with an application, the more efficient the application is said to be. Conversely, for any single task, the more time required to complete that task, then the less efficient the application is.

This definition of efficiency is similar to other measures of efficiency in different contexts. For example, in investment science, the *return on investment* (ROI) can be thought of as a measure of the investment's efficiency. Similarly, with automobiles, the *miles per gallon* (mpg) reflects a particular vehicle's efficiency. Finally, with manufacturing processes, the so-called *throughput*, in terms of units produced per day, is another measure of efficiency.

1.8.2 Theoretical Efficiency

There are two models of human-computer interaction that are examined here: the GOMS model and Fitts' Law. Both of these model are well grounded in experiemental research. The models are useful not only for obtaining an **objective** evaluation of a particular user interface design, but also for "tipping the scales" when two design alternatives seem equally good. The models should be used *in conjunction with* and not in place of designer experience and end-user feedback.

1.8.2.1 THE GOMS KEYTROKE-LEVEL MODEL The GOMS (goals, objects, methods, and selection rules) Keystroke-Level Model explicated in Raskin [Raskin, 2000] offers a practical way to quantify the theoretical efficiency of a user interface. The model, based on extensive experimentation with users performing computer tasks in a laboratory, associates theoretical timings, shown in Table 1.1, with a paricular human-computer interaction.

Abbreviation	**Action**	**Time (seconds)**
K	keying	0.2
P	pointing	1.1
H	homing	0.4
M	mental preparation	1.35
R	computer response	variable

TABLE 1.1: GOMS theoretical timings for keying, pointing, homing, mental preparation, and computer response.

Let's examine these actions in detail.

- K **(keying)**

  This is the time required to press a key on the keyboard or click a mouse button.

- P **(pointing)**

  This is the time required to point to a particular position (some (x, y) coordinate) on the view (also known as the monitor or the display). A concept known as *targeting*, discussed later, is related to yet remains distinct from *pointing*.

- H **(homing)**

  This is the time required to move the user's hand from the keyboard to the mouse (or, more generally, the GID); or, from the mouse to the keyboard. The time spent homing can often bloat the task time, making the user interface less efficient. Applications often offer modal or quasi-modal hotkeys as alternatives to mouse-driven points-and-clicks to make the interface more efficient. Hotkeys are keys that allow manipulation of the user interface without the mouse.

  – **modal hotkeys** Consider, for example, Microsoft Word running on Windows machines. A single press/release of the ALT key changes the application from a text-input mode to a control-selection mode. After the ALT press/release, the controls contained in the view—in particular the menus—have text that newly contain a single underlined letter in the words. A press of the ALT key, for example, causes the "File" menu to appear as "F̲ile". A subsequent keystroke of the "F" key *will not* put the letter "F" in the current document, but instead will open the "File" menu as if the user pointed and clicked on it.

– **quasi-mode hotkeys** Again, use Microsoft Word as an example. Pressing (without releasing—thus the quasi-modal key action) the CTRL key and then the "P" key causes the "Print" dialog box to appear. *Note: For details and definitions of modes and quasi-modes, consult Section 1.16*

- M **(mental preparation)**
  This is the time the user spends thinking about what to do next. It assumes the user is familiar with the interface; that the user is not learning the interface for the first time.

- R **(computer response)**
  This is the time the user spends waiting for the computer to respond to user input. A long process on the behalf of the computer, such as searching for a particular file on a large hard drive, may require a long user wait time. Research has shown that values of R greater than 10 seconds generally tend to annoy users; values of R exceeding 30 seconds impart user suspicion that the application may have crashed. During long computer response times, the application should always provide the user some type of feedback e.g., a progress bar, status strings, or an animation.

With these human-computer interaction steps at hand, we are now ready to list the steps required to calculate, given a particular task, the theoretical efficiency of a user interface:

- **Step 1:** Identify the task.

- **Step 2:** Identify the user interface used to accomplish the task (for example, a particular dialog box you are designing).

- **Step 3:** In the context of a candidate user interface design, break down the task into discrete human-computer user interaction steps, associating each step with a K, P, H, M, or R.
- **Step 4:** Concatenate all the discrete steps for the task.
- **Step 5:** Apply the heuristic rules (listed below) to the concatenated listed composed of Ks, Ps, Hs, Ms, and Rs.
- **Step 6:** Sum the timing for each discrete step to arrive at a total task time.
- **Step 7:** Examine the discrete steps and identify bottlenecks.
- **Step 8:** Redesign the user interface; trying to increase user efficiency.
- **Step 9:** Iterate. Go to Step 3, compare the theoretical timing of the new design to the previous design(s).

See the examples throughout Section 1.25 for illustrations of these steps applied to various user interface designs and scenarios. The aforementioned heuristic rules are extracted from [Raskin, 2000], *Heuristics for Placing Mental Operators*.

- **Rule 0 Initial insertion of all candidate Ms**
  Insert Ms in front of all Ks (keystrokes). Place Ms in front of all Ps (acts of pointing with the GID) that select commands, but do not place Ms in front of any Ps that point to arguments of those commands.
- **Rule 1 Deletion of anticipated Ms**
  If an operator following an M is fully anticipated in an operator just previous to that M, then delete that M. For example, if you move the GID with the intent of tapping the GID button when you reach the target of your GID move, then you delete, by this rule, the M you inserted as a consequence of rule 0. In this case, PMK becomes PK.
- **Rule 2 Deletion of Ms within cognitive units**
  If a string of MKs belong to cognitive unit, then delete all the Ms but the first. A cognitive unit is a contiguous sequence of typed characters that form a command

name or that is required as an argument to a command. For example, Y, *move*, *Helen of Troy*, or *4564.23* can be examples of cognitive units.

- **Rule 3 Deletion of Ms before consecutive terminators**
  If a K is a redundant delimiter at the end of a cognitive unit, such as the delimiter of a command immedicately following the delimiter of its argument, then delete the M in front of it.

- **Rule 4 Deleteion of Ms tat are terminators of commands**
  If a K is a delimiter that follows a constant string—for example, a command name or any typed entity that is the same every time that you use it—then delete the M in front of it. (Adding the delimiter will have become habitual, and thus the delimiter will have become part of the string and not require a separate M.) But if the K is a delimiter for an argument string or any string that can vary, then keep the M in front of it.

- **Rule 5 Deletion of overlapped Ms**
  Do not count any portion of an M that overlaps an R—a delay, with the user waiting for a response for the computer.

The process of using the GOMS model for every design in the user interface may seem overly time-consuming. However, the GOMS model provides an *objective* measure of efficiency that can be used to compare two competing designs alternatives. Often the merits of two user interface designs are hotly debated not from an objective viewpoint, but rather, from what a particular person *feels* is a better design. In instances such as these, which occur quite frequently with user interface design, the GOMS model is an effective tool that can be used to adjudicate, at least from an efficiency point-of-view, which design is superior.

1.8.2.2 FITT'S LAW

- Fitts' Law - time to target
- Mac vs. PC top menu items

- four corner efficiency and the edges
- buttons too small - controls nested too tightly together

1.8.3 Example: Hotkeys facilitate user efficiency

Some of the discussion above commented on the use of hotkeys to eliminate the need for a user to move back and forth between the keyboard and the mouse. Indeed, this homing activity eats away at user efficiency. Were it possible for the user to use only the mouse, or only the keyboard, this homing activity would be irrelevant. However, with many applications, simply having a keyboard-only or mouse-only interaction for user input is unreasonable. Applications such as CAD and CAE programs make extensive use of a graphics view port. Articulating the view and the items contained therein require painting gestures that are easily accomplished with a mouse, yet nearly impossible to perform with a keyboard. Therefore, it is unlikely that any modern-day application running within a *graphical* user interface could obviate use of the mouse completely. However, the use of hotkeys **as alternatives** implemented in parallel to mouse actions can offer the habituated user sizeable increases in efficiency. Two exmples come readily to my mind:

- The Windows 2000 operating system has a task bar that contains active applications. A common way to switch between applications is to point and click on the task bar items. This way, however, requires the use of the mouse. Alternatively, Windows has a keyboard-only method, consisting of holding down the ALT key while successively tapping the TAB key to switch between applications. The latter requires significantly less time and is therefore more efficient.
- The Lotus Notes 6 email application displays hotkeys that appear when the user holds down the ALT key. These hotkeys appear as small yellow boxes (resembling a tooltip), containing a single number or letter, above the user interface controls, such as menus and buttons. I have found that using these small tooltip-like hotkeys in conjunction with the ALT key allows me to read, compose, and reply to email much faster than when I was using *both* the keyboard and the mouse.

The take-home message as it applies to user efficiency is this: While it may not always be possible to allow the user to have keyboard-only interaction with the application, the application designers should make every effort to include keyboard equivalents to mouse actions, which in turn can increase user efficiency.

1.8.4 Fitts' Law

From this point forward in this section—content to come; outline follows.

- time to target
- Mac vs. PC top menu items
- four corner efficiency and the edges
- buttons too small - controls nested too tightly together

1.8.5 Experimental Efficiency

- Usuabilty Laboratories
  - Experimental Design - Objectives and Metrics
  - Test subjects
  - Critical number of test subjects - maximize benefit to cost ratio at three users - done informally, this is good news to us
  - Thinking aloud - good for flagging big time sinks
  - Some type of automated program? If believe Nielsen's research of only needing three people, then perhaps dimished value in collecting lots of use statistics

1.8.6 Subjective Efficiency

- In-the-field - feedback from users, sales, AEs
- Beta testing

1.9 Familiarity

AUTHOR: C.B. HOVEY

CREATED: 12 NOVEMBER 2003

REVISED: 19 NOVEMBER 2003

1.9.1 Introduction

Though it may seem like a minor point, many user interface researchers prefer to avoid using the word "intuitive" to describe user interfaces; they prefer to use "familiar" instead.

> Many interface requirements specify that the resulting product be intuitive, or natural. However, there is no human faculty of intuition, as the word is ordinarily meant; that is, knowledge acquired without prior exposure to the concept, without having to go through the learning process, and without having to use rational thought.
>
> When uses say that an interface is intuitive, they mean that it operates just like some other software or method with which they are familiar [Raskin, 2000].

1.9.2 Definition

Therefore, the notion of a user interface being intuitive actually means one of two things: familiar because it is consistent, or familiar because it supports habituation.

- Familiar (Intuitive) relating to **Consistency** (cf., Section 1.7)—Applications, for example, built on the Windows framework with Windows-consistent design (e.g., top level menus such as 'File,' 'Exit,' and 'Help') are consistent with each other. When a user is presented with an application she has never seen before, the user will easily and quickly navigate the application commands because the application is consistent with other applications she has used. The user might claim the application is therefore *intuitive*, but what the user is really conveying is that the application is *familiar* because it is *consistent* with other applications.

- Familiar (Intuitive) relating to **Habituation** (cf., Section 1.10)—Applications with a design that is Consistent with the application framework they are build upon (e.g., Windows, Apple, Palm) provide a mechanism for users to become nearly instantaneously productive with a certain subset of application commands—for example, the Windows 'Save,' 'Save As...', 'Print,' and 'Exit' menu items in the 'File' menu. This productivity is attributable to the **Habituation** principle. Users who have become habituated to these commands are as efficient at executing these commands with a well-known application as they are with a never-before-seen application. By way of habituation, the efficiency a user experiences with the new application leads her to describe the interface as *intuitive*. What the user is really conveying is the habits learned in one interface make her immediately effective and efficient in this new interface.

1.9.3 Summary

When a user interface is claimed to be "intuitive," what is really meant is that the interface is "familiar" to the user.

> One of the most laudatory terms used to describe an interface is to say that it is "intuitive." When examined closely, this concept turns out to vanish like the pea in a shell game and be replaced with the more ordinary but more accurate term "familiar." [Raskin, 2000]

This Familiarity stems from the user interface being designed in such a way that it supports both Consistency principle and the Habituation principle.

1.10 Habituation

AUTHOR: C.B. HOVEY

CREATED: 27 AUGUST 2003

REVISED: 08 SEPTEMBER 2003

In a general sense, habituation means for a person "to fall into a habit." People exhibit both good habits (brushing one's teeth before going to bed, getting exercise every day) and bad habits (chewing one's fingernails, continuing to eat even after feeling full). These good and bad habits extend to the human-machine interaction. A user's habits will participate in the human-machine dialog, and influence the user's impression of how easy or difficult a machine is to use.

Designers of user interfaces must embrace the habituation principle if they wish to create quality software. Habituation can work for or against the designer. Habituation can also work for or against the user. In the following, I define habituation, and then demonstrate examples of habituation used to the benefit and detriment of both users and designers.

1.10.1 Definition

Habituation is simply the process of completing a task so often, that completion of the task becomes automatic, requiring little or no conscious thought. A person, through the habituation process, will gradually become more efficient at completing that task. Consider the following concrete examples.

1.10.2 Example: Touch Typing

A person who has learned to touch-type has undergone a process of *discovering* where each letter of the alphabet resides on the keyboard. Next, the person slowly begins to master key placement, *remembering* where each letter of the alphabet is. Gradually, the "hunting and pecking" for letters diminishes with increased *practice* typing the keys. After some time, the user becomes accustomed to how it feels to execute certain keystrokes within that language. For example, the user begins to recognize how patterns such as "-tion" and "-ing" feel to type. As the skill of the typist grows, the granularity of his focus stretches from individual letters and words, to entire phrases and sentences. Throughout the process of *discovery*, *recollection*, and *practice*, the following evolutions occur:

- The user's need to look at the keys on the keyboard diminishes to the point that the user need not look at the keyboard at all. Moreover, the cognitive effort expended to type the text decreases. This describes the **automation** aspect of habituation.
- The number of words a user can type per unit time increases. The user gets faster and faster at completing the task. This is the **efficiency** aspect of habituation.

These two aspects are distinct but closely related. It is largely because of the automation that the efficiency follows. The automation stems from the diminished mental involvement required from the user.

1.10.3 Example: Playing The Piano

From a cognitive point of view, playing the piano is equivalent to typing. Just as a typist learns to transcribe words, the pianist learns to transcribe notes. The habituation process is largely similar too: The pianist learns the placement of each key and the manner in which to articulate them with his fingers.

1.10.4 Habituation Disruption

Habituation disruption occurs after habituation has developed and subsequently, something about the human-machine interaction impedes the user's ability to automatically and efficiently perform a task.

The culprit causing habituation disruption might be the user. Perhaps the user has not performed a certain task in such a long time the user has grown unaccustomed to the task. Consider the piano example. In this context, a person who has ceased playing the piano for a long period of time might forget how to play a certain song that she once could play perfectly. The pianist, as they say, has "gotten rusty," a saying stemming from the extended disuse of machinery.

Alternatively, the participant causing habituation disruption might be the machine. This case, less applicable to the piano example, will become more apparent when I discuss software. However, just for the sake of completeness, let's consider how habituation disruption could be caused by the piano and not the pianist. Suppose the keys on a piano differed from piano to piano. Suppose German manufacturers produced pianos in the manner that we are all accustomed to. The C-to-C octave, shown on the left of Fig. 1.6, would contain eight ivory keys (C, D, E, F, G, A, B, C) and five ebony keys (C#, D#, F#, G#, A#) in their traditional arrangement.

Now suppose American manufacturers produced pianos that contained a single array of keys in the same octave, as shown on the right of the same figure. Instead of smaller ebony keys, inset among the ivory keys, there would be a single row of keys, each of uniform size and position. Only the color of the key (ebony vs. ivory) would be maintained. Any person accustomed to German pianos would encounter habituation disruption upon attempting to play an American piano, and vice versa. Absurd as this example

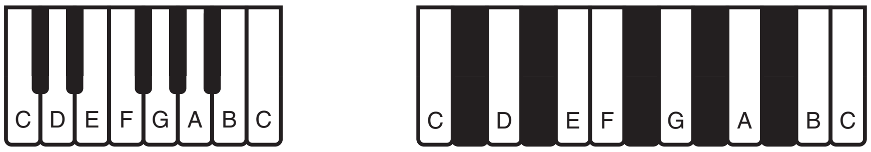


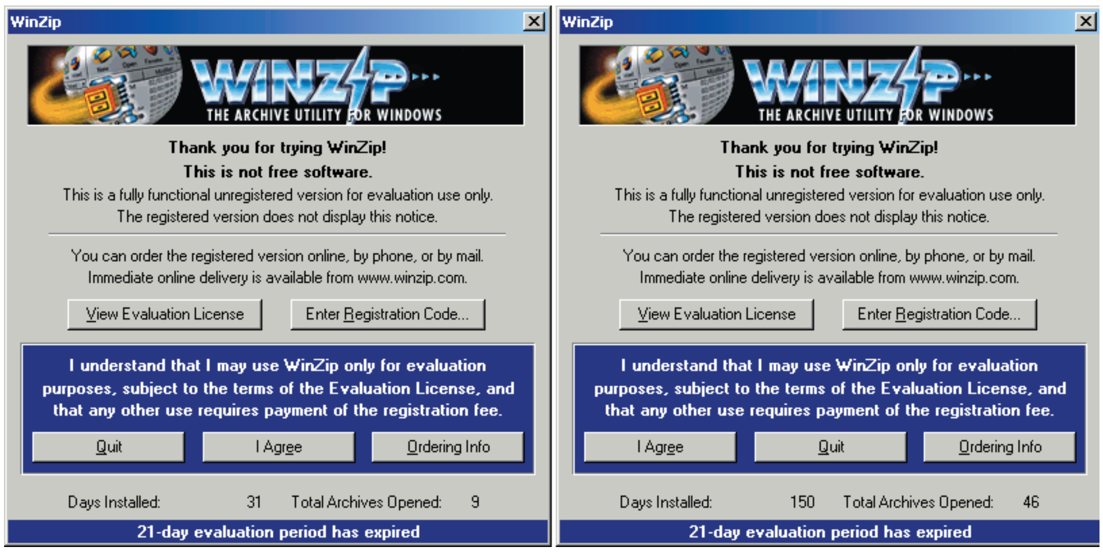
FIGURE 1.6: Rearranging the piano keys (shown on the right) would create habituation disruption for pianists accustomed to the traditional arrangement (shown on the left).

might sound, it is useful to keep this illustration in mind when we start talking about software. Humans have been playing keyboard instruments successfully for centuries. The keyboard has not changed much over that time, and neither have humans. When designing software, designers can learn something from this human-machine interaction: Don't move the keys! We'll see what happens when software applications "move the keys" in subsequent sections, and how such actions cause habituation disruption. In general, habituation disruption is considered an undesirable quality to have in human-machine interactions.

1.10.5 Habituation Preemption

Habituation preemption is different from habituation disruption. Preemption attempts to prevent the user from forming habits; disruption makes the user uncomfortable once habits have been formed. For example, some software requires users to identify themselves with a user name and password. Should the application require the user to change the password, let's say, every month, then the application is utilizing habituation preemption to "protect" the user's password from becoming stale, which might allow a clandestine user to access the account unnoticed for an extended period of time.

Another example of habituation preemption is used by WinZip, an application used to compress and uncompress files. The application allows its use for a 21-day evaluation period, after which, the application is marked as "unlicensed." After the trial period has expired, the program will continue to work perfectly, but with the following hiccup: From time to time, the locations of the "I Agree" and "Quit" buttons are interchanged, as shown Fig. 1.7.



Thank you for trying WinZip!
This is not free software.
This is a fully functional unregistered version for evaluation use only.
The registered version does not display this notice.

You can order the registered version online, by phone, or by mail.
Immediate online delivery is available from www.winzip.com.

I understand that I may use WinZip only for evaluation purposes, subject to the terms of the Evaluation License, and that any other use requires payment of the registration fee.

Days Installed: 31 Total Archives Opened: 9
21-day evaluation period has expired

Thank you for trying WinZip!
This is not free software.
This is a fully functional unregistered version for evaluation use only.
The registered version does not display this notice.

You can order the registered version online, by phone, or by mail.
Immediate online delivery is available from www.winzip.com.

I understand that I may use WinZip only for evaluation purposes, subject to the terms of the Evaluation License, and that any other use requires payment of the registration fee.

Days Installed: 150 Total Archives Opened: 46
21-day evaluation period has expired

FIGURE 1.7: The WinZip uses habituation preemption to annoy the user in the hopes that the user will license the software. Notice how the positions of the "I Agree" and "Quit" buttons are interchanged on the two dialog boxes.

This button swapping behavior is meant to annoy the user and motivate the user to purchase the software. By interchanging the button location, the application designers make use of habituation preemption. From time to time, the user will grow accustomed to pressing the “I Agree” button in a particular location on the dialog box, much like a pianist grows accustomed to the location of the C# key on the piano. However, after a while, the “Quit” button will take the place of the “I Agree” button. The user, intending to quickly get past this pesky dialog with a habitual point and click of the “I Agree” button, will accidentally hit the “Quit” button. Gotcha! Instead of uncompressing the user’s files, the application promptly closes, mostly likely annoying the user.

1.10.6 Still More Examples

Briefly, I’ll enumerate several additional examples of habituation. These examples are purposely *not* linked to software applications. Rather, they are more generally tied to human-machine interactions. This level of abstraction is important to emphasize because I feel often software designers propose solutions based on how user interface widgets (combo boxes, tree controls, etc.) operate rather than on how human beings operate. It’s an easy trap to fall in to, and an effective tool at avoiding the trap is to think beyond the realm of software development, to the realm of cognitive psychology.

- **Light switches in dark rooms**—Have you ever walked into a dark room, just through the door and placed your hand on the exact position of the light switch? You couldn’t see the light switch, you just knew it was there. This is the habituation principle in action. After discovery, recollection, and practice, you became habituated to the location of the light switches and could easily turn on the lights without problems.

  Think of how disruptive it would be to your habits if someone changed the position of your light switches. Perhaps the switches were moved just slightly left or right, up or down. The habituation—and thus the automation and efficiency with which you could navigate into a dark room—would be considerably undermined.

- **Driving on the right side of the road**—Most of the world uses the right-hand lane for driving; only a few countries use the left-hand lane. Let’s say, for the sake of discussion, that you’ve grown up accustomed to and skilled at driving on the right-hand side of the road. Driving, even with traffic all around you, has become second nature.

Now imagine a trip to England. You're about to drive through the city streets. You and the steering wheel in front of you are sitting (oddly) on the right-hand side of the vehicle. As you start to drive, now in the left-hand lane, you squirm as you see oncoming traffic approach you on your right. You come to an intersection and make a 90 degree turn. You really have to stop and think about what you're doing, don't you? Your habituation gained from years of driving is of no use to you now. In fact, it is most like hazardous to your well-being, as the habituation might take you straight back to the right-hand lane, on a collision course with oncoming traffic.

- **Wrist watches**—Have you ever been asked for the time, only to stare down at your watchless wrist? There was almost no thinking involved. Your actions were automated and efficient. You have grown accustomed to obtaining the time by looking at your watch that you normally wear on your wrist.
- **The Gas and Brake**—Briefly, think about the habituation you have developed based on the placement of the gas pedal and the brake pedal in a car. What would happen if the positions of those two pedals were reversed?

1.10.7 Synthesis

In all of the examples, the pattern of *discovery, recollection, practice (or repetition), decreased mental participation, automation,* and *efficiency* appears. Habituation disruption or habituation preemption undermines the user's automation and efficiency during task completion.

Now let's look at the habituation principle in the context of user interface design. I will try to expose good and bad design to the extent that it either accommodates or ignores habituation, respectively.

1.10.8 Example: Expanding Menus

Microsoft user interface designers introduced expanding menus in many of their Office 2000 products. Figure 1.8 shows the unexpanded and expanded versions of the View menu item in Microsoft Word 2000. A user expands the menu by clicking or hovering over the double

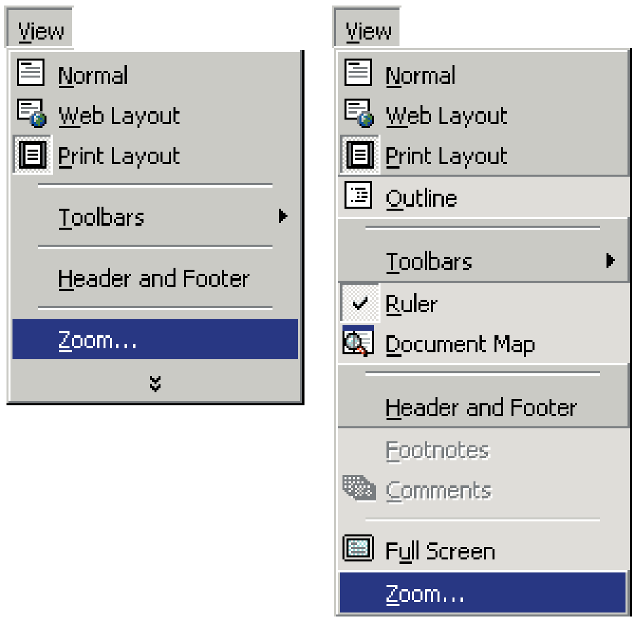


FIGURE 1.8: The unexpanded View menu (left) and its expanded counterpart (right) in Microsoft Word 2000.

v-shaped arrows on the bottom of the unexpanded menu. The rationale for such a user interface effect is well-intentioned. Microsoft interface designers hoped to make the user more efficient in the use of the Office products by showing only the most used menu items, while hiding the remaining items. With fewer menu items, there are fewer mental pattern recognition steps as the user sorts through the abbreviated list. However, the allure of increased efficiency might just be a mirage after all—enter NYSINYD.

NYSINYD is the evil brother of WYSIWYG. While WYSIWYG (what-you-see-is-what-you-get) is well-known as a desirable quality, NYSINYD (now-you-see-it-now-you-don't) is just the opposite. NYSINYD is the application playing the childhood game of "hide-and-seek" with the user interface controls. The problem with expanding menus arises when the user grows habituated to seeing a particular menu item, and then, at some later time, is unable to find the item because the application has hidden it in the unexpanded menu. NYSINYD is the enemy of habituation, and consequently, is the enemy of user friendliness and efficiency. NYSINYD causes habituation disruption.

Consider the following example. In the context of Fig. 1.8, the user first does a mental sort on the "View" menu. Upon not seeing the desired menu item, "Footnotes" for example, the user might

- do a horizontal sort, hunting through all the remaining unexpanded menus, thinking she choose the "View" menu in error, that the "Footnotes" subitem is a child of some other parent menu; or,
- do point and click to expand the "View" menu, followed by a *second* vertical sort of the newly expanded menu.

Alternatively, consider the case where the user wants to select the "Zoom..." menu item, but is slightly sloppy with guiding the mouse. The user might

- go to click the last item in the unexpanded list, misdirect the mouse just a bit so that the cursor hovers over the double v-shaped arrows, and the nudge the mouse up a bit in anticipation of selecting the "Zoom..." item. Unfortunately, the misdirected hover initiated the timer for the menu expansion action. The user clicks the mouse, but before the click gets processed, the menu expands, and the application interprets the click event as occurring on the "Document Map" menu item.

All three of these scenarios illustrate how the expanding menus design can cause the user to be *less* efficient, not more efficient, at navigating the user interface. The inefficiency stems from the fact that expanding menus inherently cause habituation disruption. Any user interface feature that promotes habituation disruption should be minimized, if not eliminated entirely.

Cooper and Reimann [Cooper, 2003] take a different yet equally negative point-of-view toward expanding menus. They comment that the expanding menus idea adds "confusing visual noise" to the interface and is "a massive train wreck" as a user interface idiom. They state, "Microsoft was clearly trying to make its menu items more usable, but in doing so it has struck a terrible blow to the fundamental reason menus exist, to teach users what functions are available." Though they do not mention the habituation principle by name, Cooper and Reimann do note that expanding menus "hide information" and "change [their] ordering", which disrupts the user's "motor memory of menu item location."

Instead of minimizing inefficiency, the expanding menus actually can exacerbate inefficiency. The strategy used by Microsoft designers seems to have backfired, all because the habituation principle was not given strong enough consideration when proposing this user interface design.

The better design simply is to leave the menus expanded at all times, while abolishing the expanding menus idea from the interface. The user will grow accustomed to the position of each menu item as is. There will be no "hide-and-seek" with the user interface; no habituation disruption. As the user gets more familiar with the program, the need to sort the menu items will decrease. The user, almost without thinking, will point and click right to the correct item, just as a practiced pianist anticipates and plays the correct piano key.

1.10.9 Example: Adaptive Palette

The adaptive palette idea, similar to the expanding menus idea, has efficiency as the motivation for its design. The adaptive palette is the combination of a button on a toolbar with a drop arrow that, when clicked, exposes a drop menu list. The toolbar button is updated every time a new tool is used, thus the palette is adaptive. The button will uniquely reflect (either with an icon, text, or both) the last used tool. Fig. 1.9 shows an example of an adaptive palette. If the user has a number of repeated tasks to complete with that tool, the

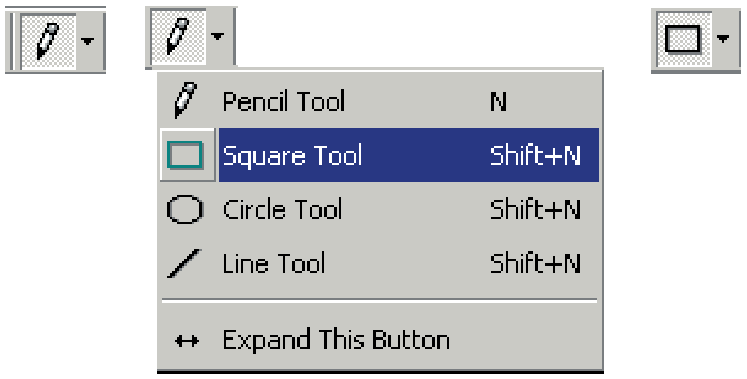


FIGURE 1.9: An example of an adaptive palette: The Tools palette from Adobe Acrobat 5.0. The Pencil Tool (left image) is currently active. A point and click of the drop arrow exposes the drop menu (center image). Selecting the Square Tool from from the list invokes the tool and updates the toolbar button with the new Square Tool icon (right image).

adaptive palette offers the user efficiency because a mental search operation is required only the first time to select the desired tool. Thereafter, the tool resides as the button, and the drop menu list need not be search through.

Now consider two implementations of the adaptive palette, one that supports habituation, and one that disrupts it. The pattern that supports habituation

- (ordering) keeps the same ordering of the drop menu regardless of which item appears as the current button in the toolbar, and
- (repetition) repeats the current button item in the drop menu list.

The example in Fig. 1.9 is an illustration of an adaptive palette that supports habituation. The list ordering never changes, and the currently selected item (e.g., the Pencil Tool) is repeated in the drop list. In contrast, the pattern that disrupts habituation

- (ordering) promotes the most recently used item to the top-most location in the drop menu while successively repopulating the list with the less frequently used items, and
- (repetition) does not repeat the currently selected tool in the drop menu.

At first it might seem that the latter alternative is more efficient:

- (ordering) Doesn't populating the drop menu with the most recently used tools near the top of the list save the user mouse hover time by making the path to the most popular items the shortest?
- (repetition) Isn't there is one less item in the drop menu to search through?
- (repetition) Isn't it obvious that the user would not select an item from the drop menu that is already currently selected (so why repeat this item both in the toolbar and in the drop menu list)?
- (repetition) Not repeating the current button item in the drop list takes up less screen space, doesn't it?

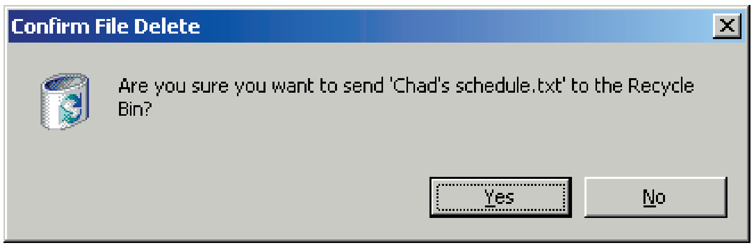
The answers to these questions are yes, yes, yes, and yes. However, this latter solution is actually *less* efficient because of the NYSINYD effect, which causes habituation disruption.

In contrast, the former pattern promotes habituation. The user can grow accustomed to simply selecting the desired tool at a known location in the list. Habituation will drive efficiency of the human-computer interaction more than any of the claimed efficiencies in the above-mentioned questions.

1.10.10 Example: Irrelevant Protections

Many applications attempt to protect the user from undesirable consequences by implementing user interfaces that ask the user to *confirm* the command the user just issued. In effect, the user issues the command twice. The user will grow accustomed to being second guessed by the application. Eventually, habituation takes over when the user issues the command, making the the designer's measures to protect the user from the user's actions completely irrelevant.

Consider Windows 2000 Explorer (explorer.exe, not Windows Internet Explorer, which is iexplore.exe). When I select a file, then press the DELETE key, the application issues the dialog box appearing in Fig. 1.10. Note that the "Yes" button is the default button, so



Are you sure you want to send 'Chad's schedule.txt' to the Recycle Bin?

FIGURE 1.10: The habituation principle makes this dialog box irrelevant.

all I have to do is press the ENTER key, and the application carries out my initial request to delete the file. This dialog is pretty useless in the context of the habituation principle. Indeed, whenever I go to delete a file now, I issue a quick sequence of *two* keystrokes, composed of the DELETE key followed by RETURN key. I don't cognitively process these keystrokes anymore. I just know this gesture deletes the file[1].

The important point here is to consider habituation when designing such user inter-

[1]Truth be told, I have actually tweaked the Windows registry to preempt the appearance of this dialog entirely, cutting in half the number of keystrokes Windows forces me to type to place a file in the Recycle Bin.

faces. The dialog box in Fig. 1.10 is poor design for the following reasons:

1. Habituation makes this dialog box irrelevant over time.

2. This dialog box unnecessarily interrupts the user's work flow (see Section **??**).

3. The act of deleting a file actually only moves the file from its present location to the Recycle Bin (a.k.a., the trash). The file really isn't deleted when I press DELETE with a file preselected. The file is really deleted only when I empty the Recycle Bin., And even then, the file is *still* recoverable with a file-recovery utility.

The better strategy for the user interface is to eliminate work flow interruption dialogs, such as the one shown in Fig. 1.10. Instead, user interface designers should focus on making the user process *reversible* (see Section 1.20), allowing users to successively recover previous states of their work. In addition to work flow interruption (cf., Section **??**), dialogs such as these should could signal application redesign is in order. If there is *really* something a user can do that is so dangerous that it requires the application to ask "are you sure" before executing, the designer might wish to consider whether the application should be capable of taking this action at all. Finally, it is important to reiterate the overriding point here—dialogs such as these soon become irrelevant protections because of the habituation principle.

1.10.11 Example: Dockable Everything

Note: This subsection contains only an outline of bulleted items (in place of written text with discussion and figures), indicating that the subsection is currently under development.

- "Dockable Everything" describes user interfaces in which the position of all user interface elements—toolbars, viewports, status bars, etc.—are movable and snap into place by docking next to one another.

- "Dockable Everything" delineates one extreme of the spectrum. At the other end of the spectrum, "One Control, One Location," the position of all user interface elements are unchangeable from the original "factory specified" location.

- Both "Dockable Everything" and "One Control, One Location" might have analogies to the Japanese manufacturing management doctrine called "**5S**", derived from the five Japanese words *seiro, seiton, seiso, seiketsu,* and *shitsuke*.
  - *seiri* (sorting out) conveys sorting out what is necessary and unnecessary; retaining the necessary and disposing of the unnecessary.
  - *seiton* (systematic arrangement) conveys the idea that there is "A place for everything and everything in its place," allowing efficient access without minimum difficulty.
  - *seiso* (spic and span) conveys cleanliness, keeping the work area free from dirt, continual cleaning of the environment, and finally eliminating the source of dirt.
  - *siketsu* (standardization) conveys that the continued practice of the first three S's lead to their unconscious implementation (e.g., *habituation*), and thus because part of day-to-day life.
  - *shitsuke* (self-discipline and sustain) conveys the need for self-discipline to keep focused on your objectives. Human nature is to return to the status quo and the comfort of the "old way" of doing things. Sustain helps to keep focus on the "new standard" for work place organization.
- "Dockable Everything" can cause quality assurance tests based on graphical locations to fail if, for some reason, the positions of the user interface elements change from the locations previously known and being tested against.
- "Dockable Everything" can make the application difficult to support. If, by way of "Dockable Everything," the user has put the application's user interface into a configuration that does not match (or at least is unrecognizable from)
  - the pictures in the manuals and online support
  - the desktop of the person providing Technical Support assistance over the telephone or by email

  then the user will encounter difficulty.

- "Dockable Everything" allows the user to feel empowered to make their workspace their own (e.g., make a house feel like a home). The user can optimize user interface controls for their own processes. If the user does not use a certain user interface element, the user has the ability to remove such elements from the user interface. Cross-apply with the *seiri*, above. This allows precious user interface real estate not to be wasted on controls that are never used.

- MSC.visual Nastran 4D is an example of software that strikes a middle ground between the two extremes of "Dockable Everything" and "One Control, One Location."

  - The "Objects" hierarchy tree remains located on the top, left-hand side of the view for all time (One Control, One Location).
  - The "Connections" list remains at the center, left-hand side (One Control, One Location).
  - The "Properties" list remains at the bottom left-hand side (One Control, One Location).
  - The toolbars can float free or dock on any of the four view borders—top, bottom, left, and right. The default position is the top.
  - The Windows Registry logs the positions of the dockable user interface elements on a per-user basis, allowing the user to return to the application with the user interface just as it was left.

- MSC.ADAMS Chassis 2003 has an example of how "Dockable Everything" causes their window pane show/hide mechanism to become unassociated.

1.10.12 Summary

- Habituation is the process of the user becoming less mentally involved with and more automated and efficient at performing a repeated task.

- Habituation disruption and preemption are viewed as undesirable since they hinder a user's ability to become more practiced and efficient at performing a task.

- NYSINYD (now-you-see-it-now-you-don't) is the strategy of selectively hiding and showing user interface controls. NYSINYD should be minimized because it
  - causes habituation disruption and thus undermines user efficiency,
  - forces a bi-modal user interaction as the application obligates the user hunt for controls at some times but not at others, and
  - creates a sense of interface instability. If the user does not understand the application's hide-and-seek strategy, the user will feel as if the user interface controls appear and disappear at random.
- User interface designers should capitalize on habituation by making placement of user interface controls (menus, buttons, etc.) persistent. Placement persistence will promote user habituation and thus foster user efficiency.
- Habituation can make "safeguards" implemented by the designer obsolete, as the user memorizes the necessary interaction pattern required to circumvent the safeguard. Designers should focus on making the software application reversible (cf., Section 1.20) rather than safe.

1.11 Help

AUTHOR: K. VINCENZ

CREATED: 02 SEPTEMBER 2003

REVISED: 30 SEPTEMBER 2003

For SimOffice, we have the unique opportunity to look at new and revolutionary ways to provide information to our users and close the gap between what our users know and what our products require our users to know. We need to provide information that our users really want to use, and the best way to do that is to make the information an integral part of the user interface, much like Microsoft and Intuit have done with the Office and TurboTax products. This is referred to as embedded help.

To explore embedded help, let's start with what users want, explain some of the benefits of embedded help, and provide examples of the different methods of embedded help.

1.11.1 What Do Users Want?

In the text *Nurnberg Funnel: Designing Minimalist Instructions for Practical Computer Skills* [Carroll, 1990], John Carroll writes that the order in which users look for help when they have a question while using a software product is:

- Try it and see what happens
- Ask another user
- Can the vendor
- Search online documentation
- Read the manual

That means that the two ways that MSC currently provides user assistance are the least likely ways in which a user will search for help. So, if users don't want traditional help and guides, what do they want? They want:

- Guidance through tasks

- Descriptions of fields
- Concepts and reference
- Examples and tutorials
- Installation and configuration information
- Help only when they want it

And, how can we provide this to them in a way that they will use it? By embedding much of it directly into the interface. Here are two examples (Fig. 1.11 and Fig. 1.12).

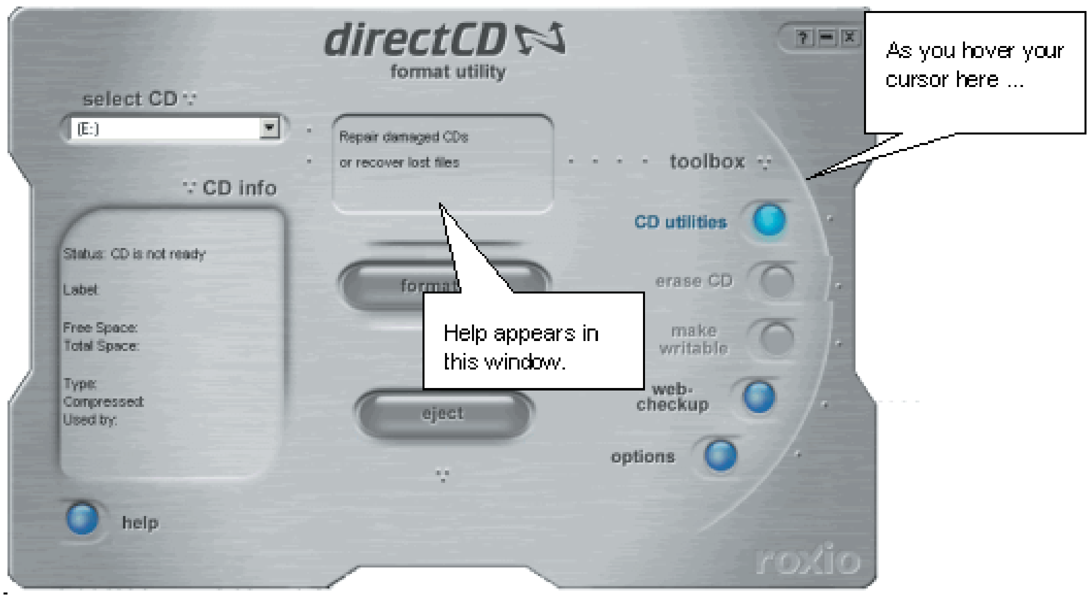
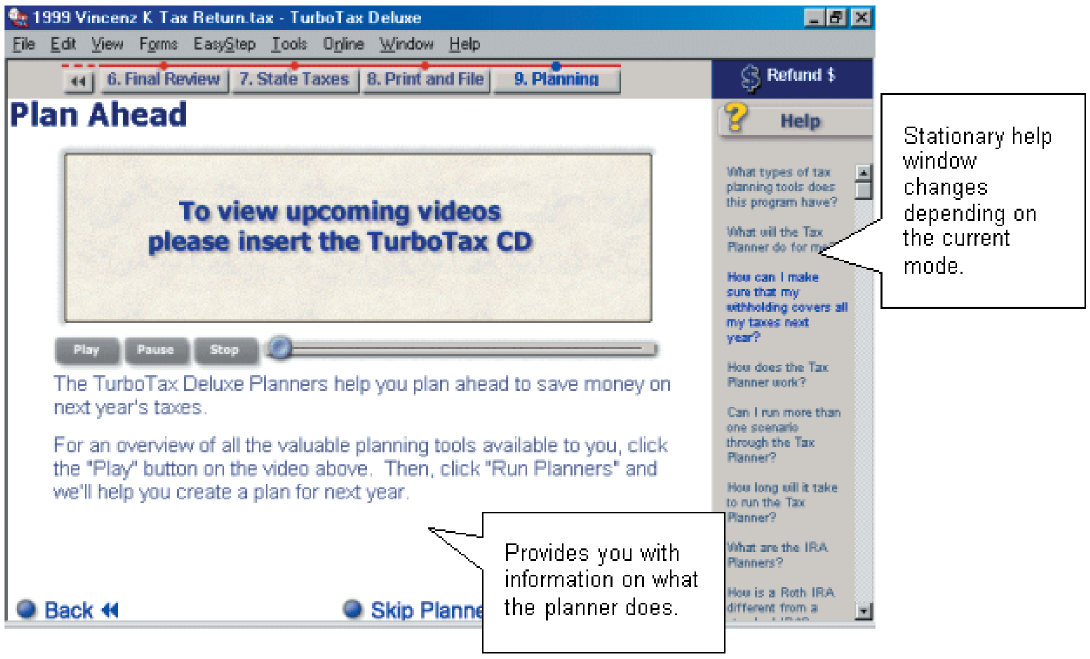


FIGURE 1.11: Roxio's directCD application with help embedded in the user interface. As you hover you cursor over the items on the right hand column, the help appears in the center window.

Embedded help:

- Provides information to users when it is needed and relevant-gets the information out of the guide and into the product.
- Lets them achieve more in less time because we provide them with product information that's never more than a keystroke away. In fact, we can develop assistance that is so well integrated into the interface that our users will never have to ask for it.


1999 Vincenz K Tax Return.tax - TurboTax Deluxe
File Edit View Forms EasyStep Tools Online Window Help
6. Final Review
7. State Taxes
8. Print and File
9. Planning
Refund $
Help
Play
Pause
Stop
Back
Skip Planne

Plan Ahead

To view upcoming videos please insert the TurboTax CD

The TurboTax Deluxe Planners help you plan ahead to save money on next year's taxes.

For an overview of all the valuable planning tools available to you, click the "Play" button on the video above. Then, click "Run Planners" and we'll help you create a plan for next year.

What types of tax planning tools does this program have?

What will the Tax Planner do for me?

How can I make sure that my withholding covers all my taxes next year?

How does the Tax Planner work?

Can I run more than one scenario through the Tax Planner?

How long will it take to run the Tax Planner?

What are the IRA Planners?

How is a Roth IRA different from a

Stationary help window changes depending on the current mode.

Provides you with information on what the planner does.

FIGURE 1.12: TurboTax application with help embedded in the user interface. The main window provides you with information on what the planner does. The items in the stationary "Help" window update based on the main window content.

- Eliminates initial frustration because users are much more likely to read the help **before** struggling and avoids the sense of failure associated with asking for help because the help is an integral part of the product.

1.11.2 How Do We Know Users Want This?

A few years ago Lotus [Lotus, 2001] gathered a group of Lotus SmartSuite users to discuss and evaluate new methods for providing user assistance-embedded help. After learning about new embedded-help methods, the users were each given $100 to spend toward the development of any or all of the new methods or traditional methods, such as user guides. Of the $1,300 to be allocated, the users only allocated $75 to writing a guide (6% of the total budget). They allocated the bulk of the money to developing the embedded help.

Not only have Lotus users realized embedded help provides more innovative and dynamic ways to provide user assistance, but Microsoft has built an entire release of its Office suite of products around the idea of enhancing the user experience, not through more and more features, but by providing user assistance embedded into the interface that enhances the user's productivity and ultimately the usefulness of the Office suite.

In addition, TurboTax has changed the way we fill out our taxes just by embedding help into the interface. You don't have to go to the IRS manual and look up what Question 12 means and calculate how many deductions you can take. Instead, that information is available to you on screen as you answer the question. Or, if it is difficult to answer, TurboTax steps through a series of questions to determine the number of deductions.

1.11.3 Benefits of Embedded Help

Providing embedded help is one of the highest values we can provide to our users at one of the lowest development costs. It will provide a quantum leap in usefulness that enhances the user's experience.

1.11.3.1 Fits with What Users Want and Need

Embedding help into the interface fits with users' requirements for quick, innovative access to information so they can get on with the tasks of virtual prototyping. It provides them with information that becomes a part of the user interface and, therefore, close to the task they are performing. They can get to "Aha!" more quickly and without interruptions.

1.11.3.2 Increases our Market Share

- Broadens our reach into new domains by providing application and domain knowledge into the product.
- Makes it easier for new users and non-traditional users to use our software because the knowledge will never be more than a keystroke away.

1.11.3.3 Saves Costs and Improves Quality

- Invests money on tangible assets that users can see and feel—their experiences—rather than on more costly and less tangible benefits such as new features.

- Reduces printing costs because we no longer depend on guides to provide information to customers.

- Ensures more complete and accurate information because we don't have to stop writing one month prior to a release to print guides.

1.11.3.4 INCREASES OUR LEADERSHIP Jumps us ahead of the competition, following the lead of Microsoft's XP technology. Microsoft is so sure that there is a huge market for embedded help that they have designed an entire release around it, and so should we.

1.11.4 Types of Embedded Help

The following are different methods for embedding help into the interface:

1.11.4.1 SEARCH To quickly get assistance, we can embed a text box into the interface that lets the user ask a question of the help. Because it is embedded into the interface, users don't have to first open the help window, and then search. They can search directly from the product. Microsoft Office 2000 and XP have this functionality. There is an **Ask a Question** box (see Fig. 1.13) on the menu bar where you can type questions:

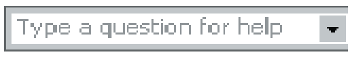


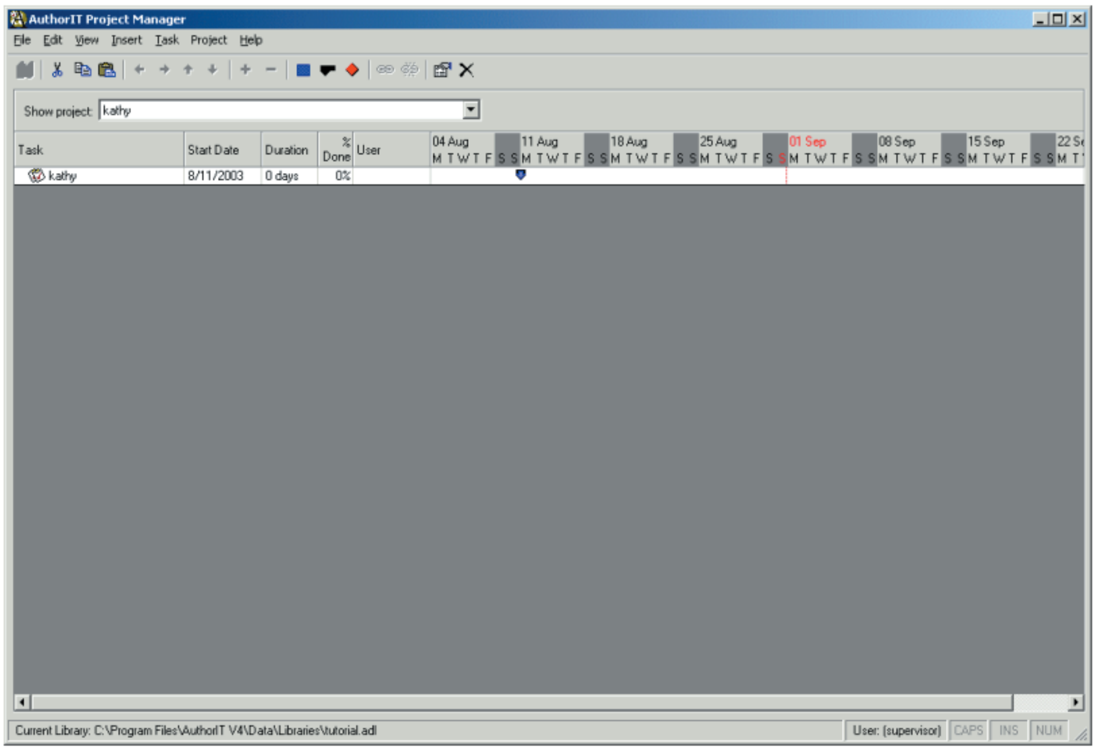
FIGURE 1.13: The **Ask a Question** box in Microsoft Office 2000 and XP.

Note: Whether or not we could have users actually enter a question (natural language query) would depend on the search engine that we implemented. Currently, the search engine used in MSC.ADAMS documentation does not have that capability.

1.11.4.2 PROCESS HELP With this technique, we reserve a portion of the window for displaying tasks that the user could perform to user get started. We ask: what would you like to do? The help would be context-sensitive so it would be relevant to the portion of the interface the user was currently using, such as the build mode.

The need for this type of help is evident when you think about the last time you learned a new product. When presented with a new product, users get what is called the ”lumberyard syndrome.” They’ve been taken to a lumberyard and been told to pick out what they need to build a house. Where do they start? We’d answer that by displaying some of the tasks that users could do in the current mode. This provides help without the user even asking. The user would have the option to close this window.

Here is an example of a product that gives you the lumberyard feeling (Fig. 1.14):



Show project: kathy

Task	Start Date	Duration	% Done	User
kathy	8/11/2003	0 days	0%	

Current Library: C:\Program Files\AuthorIT V4\Data\Libraries\tutorial.adl

User: (supervisor)

FIGURE 1.14: An application that gives the “lumberyard” feeling.

And two that do not give you that lumberyard feeling (Fig. 1.12 and Fig. 1.15). Placing a help window in the interface as shown Fig. 1.12 is referred to as stationary embedding. Its advantages are:

- Creates an always-available space from which the user can open help.
- Moves assistance into the user’s view, making it a familiar part of the interface.
- Minimizes the need to find, resize, and manage the help window.

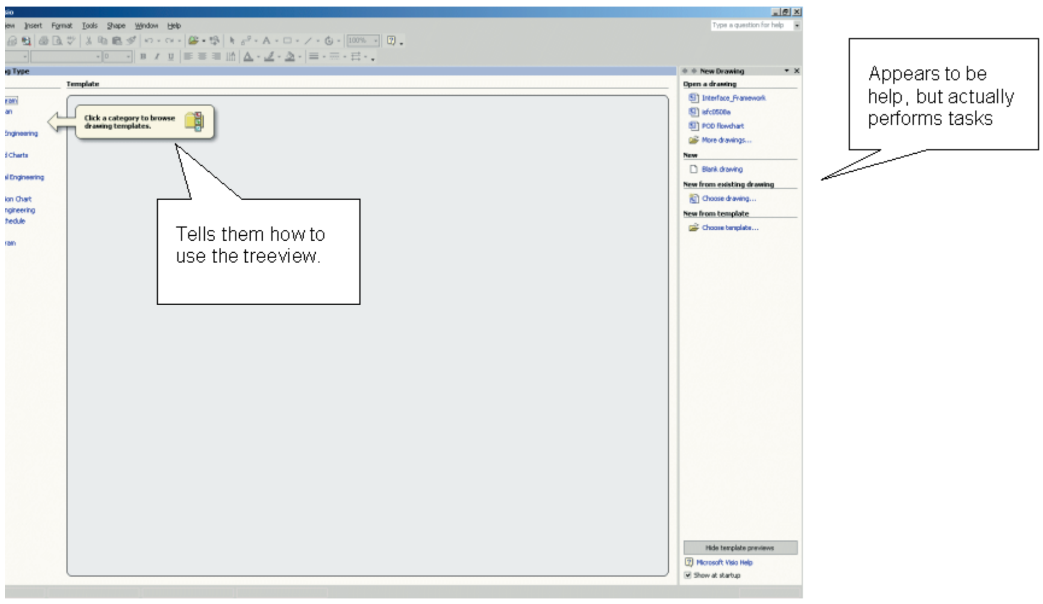

Click a category to browse drawing templates.
Tells them how to use the treeview.
Appears to be help, but actually performs tasks
New Drawing
Open a drawing
New
Blank drawing
New from existing drawing
Choose drawing...
New from template
Choose template...
Microsoft Visio Help
Show at startup

FIGURE 1.15: Microsoft Visio. The help on the left hand side of the screen tells users how to use the treeview. The right-hand-side column appears to be help, but actually performs tasks.

Given that MSC products perform very complicated operations, retaining a portion of real estate for the help window may not be possible. In addition, the help itself would have to be very focused, otherwise, it may require more space than can be allocated, and would need to be a separate window.

1.11.4.3 TIPS Many people call Tips of the Day "Random bits of help that have nothing to do with the task at hand." Therefore, we give the user the ability to set the help window to display tips on how to use features of MSC products more effectively, and display these tips in an organized manner. Microsoft Office uses a light bulb, shown in Fig. 1.16, to

FIGURE 1.16: The Microsoft Office lightbulb is used to indicate a tip.

indicate a tip. Many applications also show the user the tip using an embedded movie (Show me) or actually perform actions for the user in the software.

1.11.4.4 SCREEN TIPS/DIALOG-BOX HELP The help that users will most often use is screen tips, which include tool tips and what's this help. In addition, they will use dialog-box help, which explains all the options in a dialog box. Users select this type of help if they aren't sure what a specific command or button does or if they want to know more about an option in a dialog box. Screen tips show information about different elements on the screen.

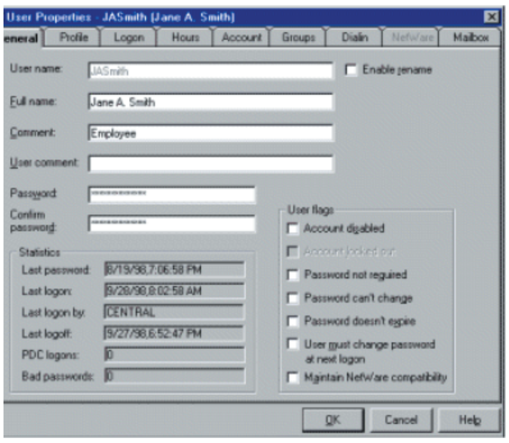
Common ways to access screen tips and dialog-box help:

- For help with a menu command, toolbar button, or screen region, on the "Help" menu, click **What's This?**, and then click the area for which you want help.
- For help with a dialog box option, click the question mark button $\boxed{?}$ in the upper right-hand-corner of the dialog box, and then click the option.
- To see the name of a toolbar button, rest the pointer over the button; the name appears.
- To see more about all the options in a dialog box, press **F1**.

The **What's This?** and $\boxed{?}$ help traditionally appear as pop-ups over the item, while the dialog-box help appears in a separate window. We could also reserve a portion of the dialog box for this type of help to appear in-a dialog-box companion help window.

Fig. 1.17 shows examples of the same dialog box, without a dialog-box companion help window and with.

1.11.4.5 WIZARDS AND INTERVIEWERS - INSTRUCTIONAL EMBEDDED HELP As users use MSC products, they are often required to enter domain knowledge (knowledge that only the most expert users have). To assist them, we can interview them, asking questions at the point that they need it most-when entering values and selecting options. This is called instructional embedded help. For example, while selecting an integrator for a simulation, we can develop a set of questions about their model and recommend an integrator and ADAMS/Solver settings (an integrated "10 steps to happy simulations"). At the end of the interview, the MSC product would apply the settings for the user. We could also make the interviewer an option, so expert users wouldn't have to use it.

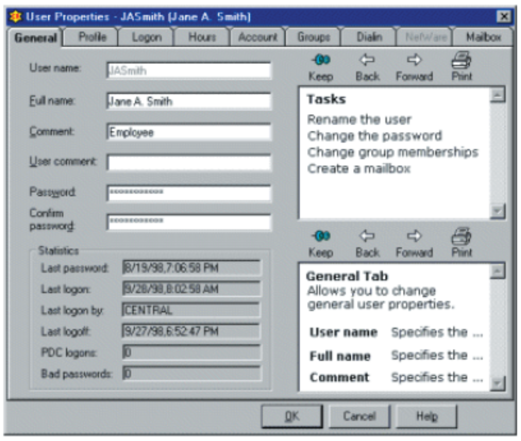


User Properties - JASmith (Jane A. Smith)

User name: JASmith
Enable rename
Full name: Jane A. Smith
Comment: Employee
User comment:
Password: ***********
Confirm password: ***********

Statistics
Last password: 8/19/98,7:06:58 PM
Last logon: 9/28/98,8:02:58 AM
Last logon by: CENTRAL
Last logoff: 9/27/98,6:52:47 PM
PDC logons: 0
Bad passwords: 0

User flags
Account disabled
Account locked out
Password not required
Password can't change
Password doesn't expire
User must change password at next logon
Maintain NetWare compatibility



User Properties - JASmith (Jane A. Smith)

User name: JASmith
Full name: Jane A. Smith
Comment: Employee
User comment:
Password: ***********
Confirm password: ***********

Statistics
Last password: 8/19/98,7:06:58 PM
Last logon: 9/28/98,8:02:58 AM
Last logon by: CENTRAL
Last logoff: 9/27/98,6:52:47 PM
PDC logons: 0
Bad passwords: 0

Tasks
Rename the user
Change the password
Change group memberships
Create a mailbox

General Tab
Allows you to change general user properties.

User name Specifies the ...
Full name Specifies the ...
Comment Specifies the ...

FIGURE 1.17: The same dialog box, without and with a dialog-box companion help.

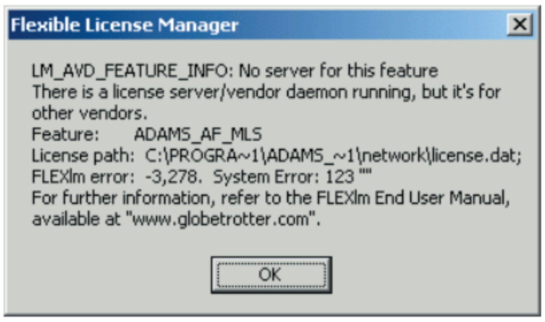
You can see uses of interviewers throughout the TurboTax product. (For fun examples of interviewers, go to SelectSmart, and answer questions on various issues to find candidates that are the most appropriate to your views. I was surprised at the answers!).

1.11.4.6 HELP MESSAGES NOT ERROR MESSAGES

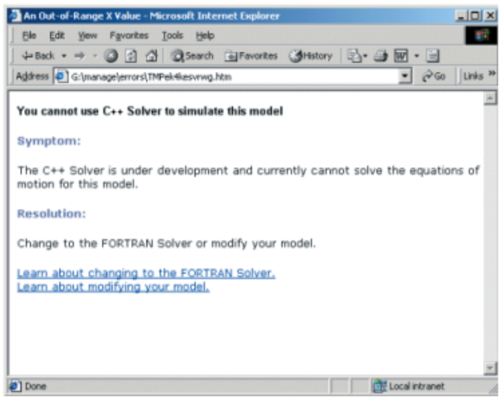
For the last two releases of MSC.ADAMS, we've been working on improving the quality of error messages. For SimOffice, we need to build on that, and begin by making the error messages help messages. The error messages could appear in help window with information about:

- What went wrong
- Why (if helpful the user)
- What they can do to resolve the error

Fig. 1.18 shows two examples of error messages. Which would you want to appear? By placing the error message in a help window, the error message can then link to other help information. Also, if possible, we could automate the corrective actions. Why just tell our users what they should do to correct the problem? Instead, why not have a button that asks them if the product can perform the corrective steps: adjust settings, change the model? Now that's a really helpful message.



```
LM_AVD_FEATURE_INFO: No server for this feature
There is a license server/vendor daemon running, but it's for
other vendors.
Feature:      ADAMS_AF_MLS
License path:  C:\PROGRA~1\ADAMS_~1\network\license.dat;
FLEXlm error:  -3,278.  System Error: 123 ""
For further information, refer to the FLEXlm End User Manual,
available at "www.globetrotter.com".
```


An Out-of-Range X Value - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address G:\manage\errors\TMPek4kesvrwg.htm
Go Links
Done
Local intranet

You cannot use C++ Solver to simulate this model

Symptom:

The C++ Solver is under development and currently cannot solve the equations of motion for this model.

Resolution:

Change to the FORTRAN Solver or modify your model.

Learn about changing to the FORTRAN Solver.
Learn about modifying your model.

FIGURE 1.18: Two contrasting examples of error messages.

1.11.4.7 LINKS TO MSC WEB SITE To facilitate providing our users with the most up-to-date information and allowing us to continue to improve the user assistance we provide, we need to provide links to the MSC Web Site directly from SimOffice. For example, by providing links to the Web, we could let users access technical resources and download free product enhancements-all without leaving their SimOffice product.

A couple of ways to do that are:

- Providing a **SimOffice on the Web** command on the **Help** menu to let them go to their product's support Web site, which would be a gateway to white papers, release notes, latest tips and tricks, frequently-asked questions, knowledge base, ASK archives, and help on the Web.

- If the correct topic doesn't appear in the help, the user would have the option to click **Look for more help on the Web** at the bottom of the list of topics.

Because of file size and installation times, we could have some topics only available from the Web or have any movies, Show Me examples, or example files accessible only from the Web. These items could have an indicator, such as the word (Web) in front of them to let the user know it's only on the Web.

1.11.4.8 AGENTS An agent keeps users in the know about MSC products by informing them when the latest software patch, new documentation, or knowledge base articles are available, or by automatically searching the help on the Web or the knowledge base when no help is available for a topic.

1.11.4.9 User Preferences

Studies have also shown that customers want to be able to customize the help they receive, so they only get the help that applies to them. In Microsoft Office, you can select the types of tips you receive. For example, if you prefer using the keyboard to using the mouse, you can have the Assistant display tips on shortcut keys. In other products, you can say whether or not you only want to see advanced topics or topics for a certain product.

1.11.4.10 Quick Demos, Tutorials, Product Tours, and Examples-E-Learning

The goal of these types of help is to get users started quickly, allowing them to focus on the larger tasks, not the subtasks within subtasks. We want them to get interested in our products, and to sustain that interest as they learn the products. Quick demos, tutorials, and product tours are usually presented to the user on the opening screen, with buttons, such as Take a Product Tour, Learn the Basics, and others.

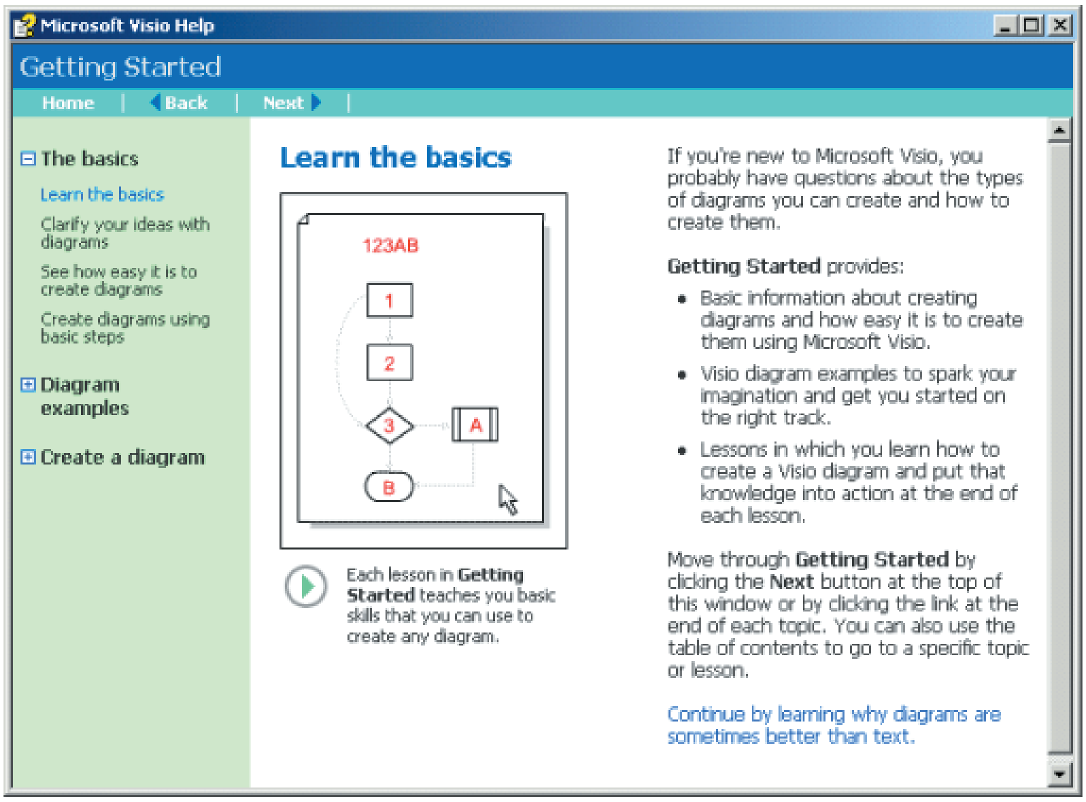
These types of help also use the techniques most often found in electronic learning (e-learning) to make the help more fun, visual, and appealing. Some of the e-learning techniques that we could use in the tutorials and traditional help, include:

- Quizzes to ensure they understand what they have learned.

- Pictures with overlays instead of just lists of text.

These techniques will enhance the learning experience and match the learning style of younger engineers who want information in a much more visual format.

Fig. 1.19 shows an example of the Getting Started tutorial for Visio. As you open the Getting Started tutorial, the diagram in the center draws itself, showing you some Visio techniques.



Getting Started

The basics

Learn the basics

Clarify your ideas with diagrams

See how easy it is to create diagrams

Create diagrams using basic steps

Diagram examples

Create a diagram

Learn the basics

Each lesson in **Getting Started** teaches you basic skills that you can use to create any diagram.

If you're new to Microsoft Visio, you probably have questions about the types of diagrams you can create and how to create them.

Getting Started provides:

- Basic information about creating diagrams and how easy it is to create them using Microsoft Visio.
- Visio diagram examples to spark your imagination and get you started on the right track.
- Lessons in which you learn how to create a Visio diagram and put that knowledge into action at the end of each lesson.

Move through **Getting Started** by clicking the **Next** button at the top of this window or by clicking the link at the end of each topic. You can also use the table of contents to go to a specific topic or lesson.

Continue by learning why diagrams are sometimes better than text.

FIGURE 1.19: Getting Started in Visio.

1.11.5 Traditional Help and Guides-Still?

1.11.5.1 TRADITIONAL HELP SYSTEM A traditional help system is always necessary, providing the user with a place to go to look for information and the ability to print that information. Traditional online help provides all the information about a product, with good searching, navigation, printing, and indexing capabilities to promote serendipitous learning. The ways you can use a traditional help system include:

- View the Table of Contents.
- Search for specific words or phrases using the Search engine or Index.

The traditional help system generally appears is in a separate window, and launched either from the Help menu or from the stationary help window, with a button, such as Learn more.

1.11.5.2 Guides

As the Lotus focus group acknowledged, although they often request a printed or online guide, they don't use them and find them ″boring and hard to use.″ In Lotus' analysis:

> Although there's a lingering romance with big, traditional manuals among users, most seem willing to accept smaller and different types of printed documentation than would've been the case just a few years ago. They often find traditional printed documentation bulky, boring, and difficult anyway. Much more attractive is short, "quick and dirty" introductory booklets [Lotus, 2001].

So with the move to embedded help would be the elimination of most guides, and the beginning of a new type of guide, *strategy guides*. They would be called, *Getting Results with...* These strategy guides would introduce users to many concepts of using the product, including a tutorial. The strategy guides, in addition to quick reference cards, white papers, and installation instructions, would be the only printed information we would provide. All our other development activities would be spent on online and embedded help.

1.11.5.3 Implications of Recommended SimOffice Embedded Help

- Much of the user assistance recommended here is help that is designed into the software—not as an afterthought. It is not part of a separate development effort-but as part of the same development cycle, indeed part of the same code, as the software application itself.
- Writers and developers must work even more closely—requiring them to remake their relationship.
- Writers will have to learn how to use different authoring tools, including coding tools.
- To ensure that we are successful and that all applications in SimOffice implement the embedded help, we should start with just a few of these recommendations.
- We must continue to address the issues of users lack of domain knowledge to use

our products. Often, in traditional help systems and in embedded help, we are only helping them use the tools, not helping them know what values to enter.

1.11.6 Conclusion

Embedding user assistance into our interface will fundamentally change how users feel about MSC products. MSC.ADAMS or MSC.MARC intimidating? Not anymore. The revolution away from guides and help and towards embedding help into the interface has already started, and MSC should lead it in the CAE market. We are the leaders in the CAE market based on the functionality and breadth of our software, so let's extend that leadership to providing a remarkable user experience. Let's set ourselves apart as a solution that makes it easier and faster for users to learn and use virtual prototyping.

1.11.7 Further Reading

- [Carroll, 1990] [Carroll, 1998] [Grayling, 1998] [Millar, 1998] [Horton, 1993] [Lotus, 2001] [Zubak, 2001] [Zubak, 2001b] [Scanlon, 2001] [Houser, 1999] [Miller, 2001] [Mobley, 2003] [Mueller, 2003]
- [User Interface Engineering, 2001] [User Interface Engineering, 2001b]
- [User Interface Engineering, 2001c] [User Interface Engineering, 2001d]

1.12 Icons

AUTHOR:

CREATED:

REVISED:

- See [Horton, 1994].
- Content to come.

1.13 Idioms

AUTHOR:

CREATED:

REVISED:

- Content to come.

1.14 Localization and Internationalization

AUTHOR: K. VINCENZ

CREATED: 28 OCTOBER 2003

REVISED: 04 NOVEMBER 2003

1.14.1 Definition

Internationalization is making applications portable between languages or regions, while localization is adapting the application for specific languages or regions. By creating software that takes into consideration basic localizability guidelines, MSC can increase the quality, accuracy, and user-friendliness of the international application. MSC can also significantly reduce the cost of localizing the application into different languages.

The Mozilla Localization Web site (http://www.mozilla.org/docs/refList/i18n/) recommends keeping the following phrases in mind as you create your MSC application:

- One code base for the world
- English is just another language

Note: Many sites abbreviate:

- Localization as **L10n**
  - L + 10 letters + n; uppercase L to distinguish it from the number 1.
- Internalization as **i18n**
  - i + 18 letters + n; lowercase i to distinguish it from the number 1.

1.14.2 Goals of Internationalization

According to Sun Microsystems, Inc., an internationalized project has the following characteristics:

- With the addition of localization data, the same executable can run worldwide.

- Textual elements, such as status messages and the interface component labels, are not hardcoded in the program. Instead they are stored outside the source code and retrieved dynamically.
- Support for new languages does not require recompilation.
- Culturally dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language.
- Can be localized quickly.

1.14.3 Localization Factors in the User Interface

The following are the most common localization factors in user interface design are given in Table 1.2:

Factor	**Item**
Language	Strings in the interface: - Messages - Menus and dialog boxes - Controls
Images	Icons and bitmaps
Keyboard	Access and shortcut keys

Locale-specific settings	- **Date/time format**—Time zones, types of clocks (12- or 24-hour) separators, order of day/month/year. For example:
	`English:  Tuesday, October 12, 1954`
	`Spanish:  martes 12 de octubre de 1954`
	`Japanese:  1954#10&12$`
	- **Calendars**—Japanese, the Buddhist era, the Hijri, the Hebrew lunar, and the Taiwan; starting day of the week
	- **Number formats**—- Decimal and thousand separators, display of negative numbers, placement of percentage sign
	- **Currency formats**—Symbol and format
	- **Sort order and string comparison**

TABLE 1.2: Most common localizaton factors in user interface design.

1.14.3.1 LANGUAGE Because text grows when you localize an application, be careful when designing the following user interface components:

- Messages
- Menus and Dialog Boxes
- Controls

Be sure also to follow the writing guidelines in Localization Factors in Documentation (Section 1.14.5).

Messages

English text strings are usually shorter than localized text strings in other languages. According to the Microsoft XP Localization Guidelines, an average of 30% more space should be added for any text. Depending on the language and the phrase, the localized string might even require twice as much space.

Menus and Dialog Boxes

As with messages, menus and dialog boxes can grow when the application is localized. The following is an example from the Microsoft Visual Basic Internalization Issues Web site. It shows two identical dialog boxes in the Automated Teller Machine sample application. Extra space was allocated in the dialog box to allow for text expansion.

- Knowing that text can grow, plan your interface so that controls don't have to be resized or other elements redesigned when localized. In addition, alignment is different for different languages (left-to-right—English; right-to-left—Arabic, Hebrew; this is referred to as mirroring).
- In menus and dialog boxes, avoid crowding status bars. Even abbreviations might be longer or simply not exist in other languages.
- See Figures 1.20 and 1.21.

Controls

Avoid using controls within a sentence. You might want to place an interface control within a sentence. For example, you might want to give users a drop-down menu to make

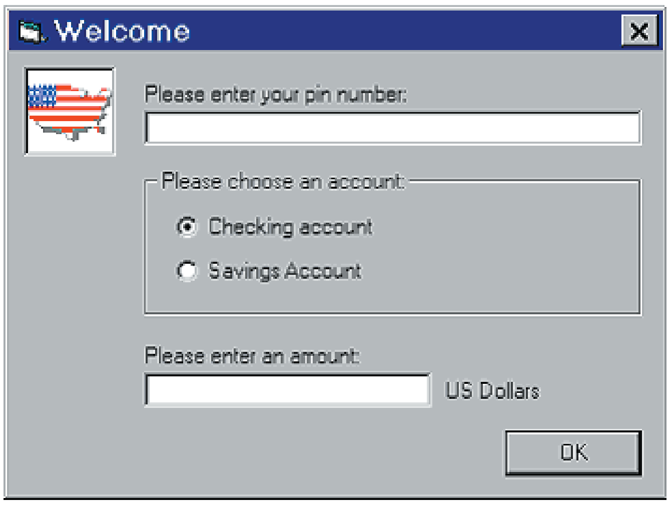


FIGURE 1.20: An English-localized dialog box.

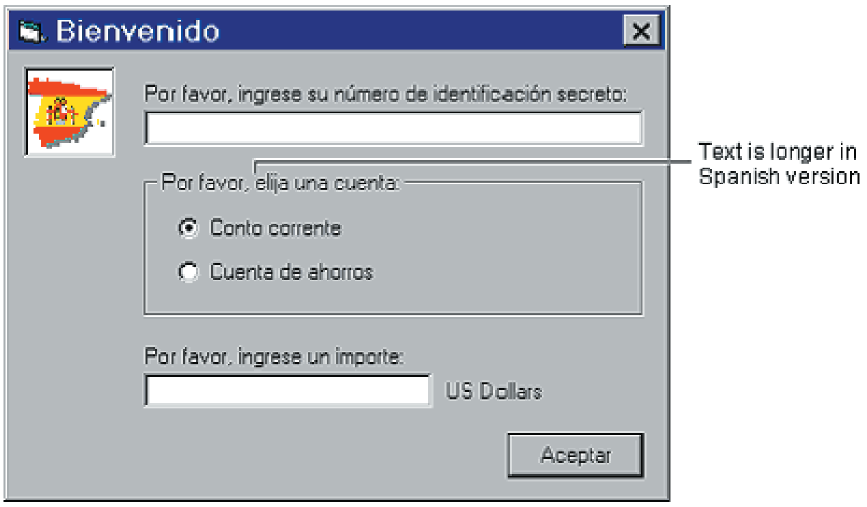

Bienvenido
Por favor, ingrese su número de identificación secreto:
Por favor, elija una cuenta:
Conto corrente
Cuenta de ahorros
Por favor, ingrese un importe:
US Dollars
Aceptar
Text is longer in Spanish version

FIGURE 1.21: A Spanish-localized dialog box.

a choice within a sentence. This practice is not recommended because to localize a sentence that includes interface controls, the localizer often has to either change the position of the controls (if possible) or be content with an improper sentence structure.

1.14.3.2 ICONS AND BITMAPS

Create icons and bitmaps that are culturally neutral:

- Avoid using people, stereotypes, faces, gender, or body parts as icons. Such images may not easily translate or could be offensive. When you must represent people or users, depict them as generically as possible; avoid realistic depictions.

- Avoid using bitmaps that are not an international standard. The bitmaps shown in Fig. 1.22 (a)–(b) represent a mailbox in the United States, but many users from other locales will not recognize it.
- Avoid using bitmaps that contain text. Text growth might become an obstacle.
- Avoid punctuation marks or alphabetic characters, which do not translate to other languages.
- Make sure that bitmaps or icons are culturally sensitive. What may be acceptable in one locale may be inappropriate or offensive in another.
  - Avoid using gestures, such as:
    - Thumbs up, see Fig. 1.22 (c). It is offensive in Italy.
    - OK, see Fig. 1.22 (d). It means *zero* in France.
    - Stop/Halt, see Fig. 1.22 (e). It is offensive in Greece.
  - Avoid mythological and religious symbols.
  - Avoid animals.

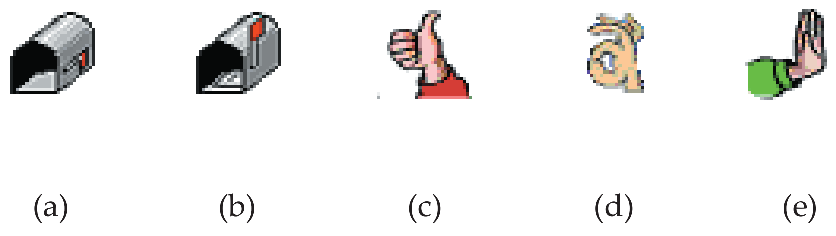


FIGURE 1.22: Icons that do not localize well.

- Use color carefully:
  - Use color only to reinforce the primary message.
  - Use color where the context does not suggest symbolic or religious interpretations.

- – Clearly define the color codes and make the color scheme explicit.
- – Test the use of color on users to ensure it is not misinterpreted.

- Avoid puns and analogies. Puns require subtle knowledge of the language. For example, do not use a picture of a mouse for the screen pointer, because in some languages the screen pointer name is not the name of a small rodent.

1.14.3.3 ACCESS AND SHORTCUT KEYS Different locales have different keyboard layouts. Not all characters are in all keyboard layouts. Make sure you can use all keyboard combinations you assign. On Windows, you can check the keyboard assignments by selecting the keyboard layout of a different locale (Control Panel -¿ Keyboards -¿ Input Locales).

1.14.4 Localization Factors in Code

While coding an application, follow the localization standards set for the operating system or development framework (QT) on which you are developing your application. Microsoft provides general instructions at Globalization Step by Step:
http://www.microsoft.com/globaldev/getWR/steps/wrguide.mspx

Be sure to consider:

- **Dates, Time, and Currency**—Never hardcode dates and currency as strings in your code. Store them internally in a locale-independent way.
- **Calendar**—Provide ways to display information in any calendar.
- **Numeric Values and Separators**—Store the value independent of the locale conventions for decimal points, thousands-separators, decimal digits used, or whether the number format is decimal. Be sure the application can display a number as a normal decimal number, currency, or percentage.
- **Sorting Text**—Sorting text means ordering text according to language conventions. There are many languages that have more complex rules for sorting than English.

Microsoft also recommends the following specific instructions, the complete text of which is available at:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/modcore/html/deconlocalizationconsiderationsinwritingcode.asp

- Use unique variable names.
- Use dynamic or maximum buffer sizes.
- Avoid composite strings.
- Avoid strings that contain a preposition and a variable.
- Avoid using compounded variables.
- Avoid dividing sentences across multiple strings.

1.14.5 Localization Factors in Documentation

Much has been written about translating documentation (guides and online help) into different languages. Generally, using good, clear writing ensures that your documentation can be successfully translated into various languages. In addition, the guidelines for Icons and Bitmaps (Section 1.14.3.2) also apply to figures and graphics in documentation. Here is a summary of the material that has been written on translating documentation.

1.14.5.1 Layout

- Use standard fonts that contain the characters used in other languages, such as accent marks. Avoid the use of italics and underlining, if possible, because it can distort many Asian characters. Do not use ALL CAPS because it won't convey a message in a language that does not use uppercase or lowercase.
- Use style sheets to standardize the formatting of text. Do not override the styles.
- Avoid the use of forced "top of page" or "page break" utilities because a break in one language may not be a break in another. Use ample widow protection instead.
- Use lists instead of paragraphs; use tables instead of prose.

1.14.5.2 Graphics

- Avoid text in a graphic. Instead, use callouts in the word-processing tool, which can be more easily translated.
- Avoid cultural references in graphics. See Section 1.14.3.2, Icons and Bitmaps.

1.14.5.3 Content and Writing Style

- For tutorials, provide culturally relevant example and tutorial subjects.
- Write simply:
  - Use the simplest verb forms.
  - Construct simple sentences
  - Use indicative and active voice.
  - Use standard grammar.
  - Avoid verbs having two or more words (Instead of *put up with* - tolerate; Instead of *make a fool of* - embarrass).
- Be consistent.
  - Instead of writing a concept differently each time, find the best way to explain it and use it consistently. This not only helps the user learn the concept, but also automated translation tools, which can significantly cut down on localization costs, only work if standard phrases are being used.
  - Use consistent spelling (do not use both traveling" and "travelling" even if they are both accepted). Again, this is important for automatic translation tools, because the utility may not handle the variation. The first spelling offered in a dictionary is usually preferable to alternate spellings.
- Avoid compound nouns. In English, you can string several nouns together without adding a preposition or a sub-clause. You usually cannot do this in other languages, which requires the translator to figure out which nouns belong together. Inserting

prepositions when writing in English would clarify the meaning. For example, “Site Server LDAP Service directory server” should be changed to “Directory server for the LDAP Service of the Site Server.”

- Define abbreviations or acronyms.
- Avoid Latin abbreviations.
- Choose words carefully:
  - Choose words with one or few meanings. Using words with more than one meaning in different contexts can require extra translation work. For example, time can mean time of day or duration, map can be a noun or a verb.
  - Avoid wordy expressions for time, place, and relationship (Instead of *three-month interval of time* - three months; instead of *300 square meters of area* - 300 square meters).
  - Choose words that are easy to pronounce.
  - Do not coin words that are not needed. Define all special terms in a glossary.
- Be culturally sensitive:
  - Avoid figurative language, especially sports or military images (huddle, game plans, tackle, blitz).
  - Avoid colloquialisms, regionalisms, and slang (that dog won’t hunt).
  - Avoid humor, wit, sarcasm, or irony.
- Be polite.

1.14.6 References

- Microsoft Globalization Step by Step
- Microsoft XP Localization Guidelines
- Microsoft Visual Basic Internalization Issues

- Mozilla Localization and Internationalization Guidelines
- Technical Topics, Globalization, Sun Microsystems, Inc.
- The Icon Book, Visual Symbols for Computer Systems and Documentation, William Horton, John Wiley & Sons, Inc., 1994.
- Twenty-Five Tactics to Internationalize Your English, Edmond H. Weiss, Intercom, May 1998.
- Graphic Design with the World in Mind, Nancy A. Locke, Intercom, May 2003.
- MSC.ADAMS Technical Writing Style Guide Hub, Grammar Tips

1.15 Locus of Attention

AUTHOR:

CREATED:

REVISED:

- Content to come.

Notes

- Work flow interruption

1.16 Modes

AUTHOR: C.B. HOVEY

CREATED: 08 AUGUST 2003

REVISED: 08 DECEMBER 2003

Modes can be found almost everywhere, in many devices used day-to-day. In some cases, the modes even announce themselves, the word MODE appears attached right on the device. Fig. 1.23 shows the presence of a mode in the controls of an air conditioner.

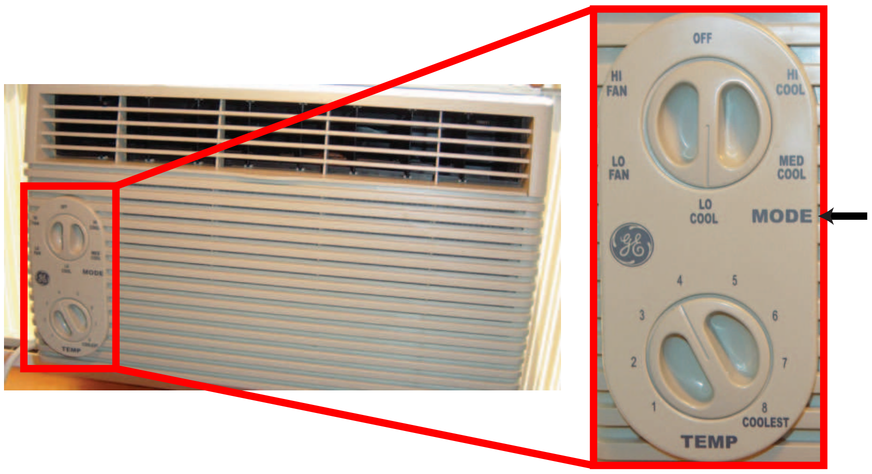


FIGURE 1.23: A General Electric in-window air conditioner (left) with a close up (right) of the control panel. Notice the MODE label (marked by the arrow) and the six modes: OFF, HIGH FAN, LO FAN, LO COOL, MED COOL, and HI COOL.

Although modes in their own right are not bad things, modes come inextricably coupled with something called a mode error. If a mode were a rose, then a mode error would be the thorns. Modes, on the outset, might seem like a sweet-smelling solution, but their use in software applications inevitably leads to mode errors that prick the user.

Mode errors often occur accidentally when the user's Locus of Attention (see Section 1.15) does not cognitively include the mode itself. Mode errors cause frustration when the user cannot "escape from the mode". The user may be observed hitting the keyboard wildly, asking aloud, "How do I get *out* of here?" or "How did I get *in* to here in the first place?"

User interface designers have long been aware of modes and mode errors. This concept was adroitly captured in a 1981 article in BYTE magazine by Larry Tesler of Apple Computer, who describes his aversion to modes. Tesler explains,

> "As I write this article, I am wearing a T-shirt given to me by a friend. Emblazoned across the chest is the loud plea: **DON'T MODE ME IN**. Surrounding the caption is a ring of barbed wire that symbolizes the trapped feeling I often experience when my computer is "in a mode" [Tesler, 1981].

What follows is a definition and several illustrations of modes and mode errors. Alternatives to modes, namely quasi-modes and modeless designs, are then presented.

1.16.1 Example: Car Crash

In many theatrical performances, movies, and television comedies, a there is an often-used genre of joke called the expectation-outcome mismatch. The mismatch occurs when a character takes some action, but the result of that action turns out to be different from the character's expectation. Often, because of the unexpected outcome, the character falls into a state of confusion and frustration. Upon seeing the character's discombobulation, the audience laughs and enjoys the performance. The audience takes part in a bit of *schadenfreude*, which is a German word that translates to the *taking delight in the misfortune of others*. The following is an example of such a joke containing the expectation-outcome mismatch.

1. The character is in a rush and not really paying attention to what he is doing.

2. He runs from the front door of his house out to his car, which is parked in the driveway, front bumper close to (but not touching) the closed garage door.

3. The hurried character jumps into his car, shifts from PARK into REVERSE (the car has automatic transmission), turns and looks back over his shoulder at the street behind him.

4. He "floors" the gas pedal.

5. He drives the car right through his garage door, crashing into everything in the garage, pieces of garage door falling on to his car.

The character has just fallen victim to a **mode error**. The character believed he put the car into the REVERSE mode. However, in his hurried, inattentive state, the character actually put the car into the DRIVE mode. In the context of this DRIVE mode, the character's action (stepping on the accelerator) produced an outcome (driving forward through the garage door) that was different from his expectation (driving backwards from the driveway to the street). These elements—expectation, mode, action, and outcome—are all constituents of a mode error, which arises naturally from a mode. Both a mode and a mode error are now defined.

1.16.2 Definitions

Software has a **mode** if that software can be put in two or more distinct states, each of which responds differently to the same user action. Each mode corresponds to a particular state. If there is only one possible state, then the software is said to be **modeless**. The definition of a mode is explained graphically in Fig. 1.24.

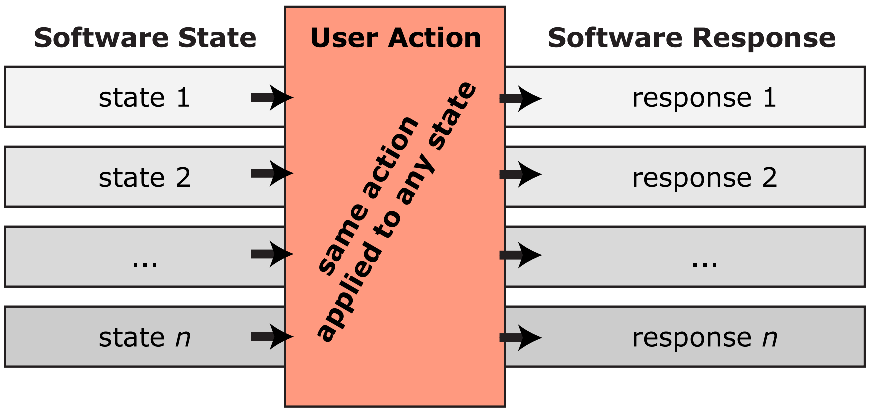


FIGURE 1.24: The mode definition: Software is **modal** if it has two or more states, that under the same user action, produce different responses.

It is important to emphasize that the software states and the software responses are unique while the user action is always identical. To make this definition concrete, consider the definition of a mode applied to the car crash example (Fig. 1.25).

A **mode error** is defined as the user experiencing a response from the software that is different from the expected response, due to the presence of a mode. Thus, for the particular car crash example, we see that the mode error is manifest as if some "wires were getting

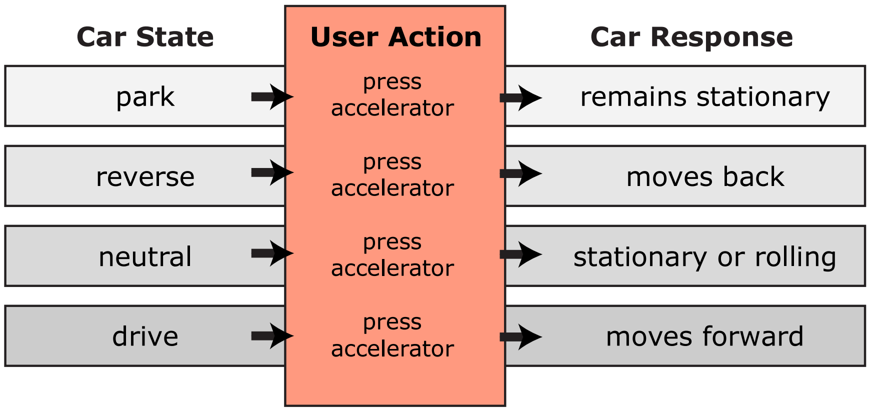


FIGURE 1.25: The car crash example illustrated with the mode definition: The four states of the drivetrain in an car with automatic transmission, PARK, NEUTRAL, DRIVE, and REVERSE, are **modes** that when acted upon by a user with a single gesture (pressing the accelerator), produce four different responses. In the case of the NEUTRAL car state, the response depends on whether the car is parked on a flat surface (stationary) or on an incline (rolling).

crossed" (Fig. 1.26). Recall these elements—expectation, mode, action, outcome—were

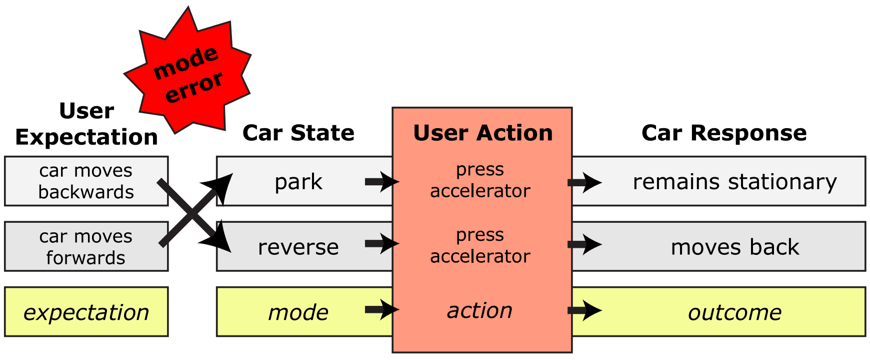


FIGURE 1.26: A mode error is illustrated as the so-called "wires-crossed" event, causing unexpected (and typically unwanted) results.

previously mentioned as the constituents of a mode error. Figure 1.26 shows a concrete example of these constituents.

While mode errors might be good for comedy, they are potentially very bad for software. Modes errors are a likely consequence when software is designed with modes. Now that both a mode and a mode error have been introduced and defined, let's look at a few more examples to elaborate on the types of errors caused by modes.

1.16.3 Example: CAPS LOCK

hAVE YOU EVER BEEN TYPING FOR A SHORT DURATION WITHOUT LOOKING AT THE RESULTS, ONLY TO LATER SEE THAT YOUR SENTENCE LOOKS LIKE THIS ONE? You see the leading "h" is not capitalized, yet all subsequent letters are. Somehow, the CAPS LOCK key on the keyboard got engaged without you being aware of it. You were in the CAPS LOCK mode, which is different from the standard input mode of a keyboard.

Here we see an example of a mode and a mode error that probably has been experienced by anyone who has used a keyboard. Consider the two modes, diagrammed in tabular form:

Keyboard State	User Action	Keyboard Response
CAPS LOCK mode →	keystrokes composing the sentence →	incorrect lead sentence
standard mode →	keystrokes composing the sentence →	correct lead sentence

The consequence of this mode error involves a time consuming *do-undo-redo* sequence. To create the sentence leading this section, I would need to type approximately 140 keystrokes (assuming no keystroke errors) without the CAPS LOCK key engaged. If the CAPS LOCK key were engaged, I would have to type approximately 420 keystrokes (assuming I use the BACKSPACE key to delete the sentence character-by-character). Clearly, mode errors create more work for the user. The *do-undo-redo* sequence caused by the mode error **triples** the amount of work that the simple *do* requires.

One parting thought on this CAPS LOCK example: Suppose I were to write,

> "!@#$!#, here I am again, stuck in a mode and I can't get out of it. The modes in this software really frustrate me and waste my time."

You would probably understand !&@#!$*% to be a sanitized way of including some profanity in my sentence. Have you ever stopped to consider the origins of such "words?" After reading about modes and mode errors, the answer should be clear. When trying to type in a sequence of numbers (such as a phone number), users get caught with the CAPS LOCK key on. Instead of seeing the telephone number, such as (800) 543-2167, users would get (*))) %$#-@!^&. Their verbal outburst (!&@#!$*%) upon experiencing the mode error and seeing `!&@#!$*%` seem to always go hand-in-hand. It is true that this mode error—typing symbols instead of numbers—is more applicable to the typewriter than to the computer keyboard. Most computer keyboards have eliminated this common mode error by ignoring the CAPS LOCK state when number keys are pressed. Nonetheless, the origin of "words" such as `!&@#!$*%` to mean profanity remains an interesting illustration of a mode error.

1.16.4 Example: Toggling Flashlight

Consider a common flashlight with a single push button to articulate between the OFF and ON states. If the flashlight is currently OFF, a press and release of the button turns the flashlight ON. Likewise, if the flashlight is currently ON, a press and release of the same button turns the flashlight OFF. This button control is referred to as a **toggle**, and thus I refer to a flashlight having this type of control as a toggling flashlight.

The problem inherent to all toggles is this: **You must know the current state before you can determine whether or not you want to press the toggle button.** Consider the following scenario for clarity. Let's say you're hiking in the woods. Your backpack is completely filled with your supplies. Unfortunately, the flashlight you packed (which is a toggling flashlight) is at the bottom of the backpack. You want to make certain that the flashlight is OFF so that you don't drain the batteries. Without unpacking a thing, you search for the outline of the flashlight by pressing through the material on the bottom of the backpack. You finally find the flashlight and can feel push button control through the material. You push the button once. You think you just turned the flashlight OFF, . . . or did you? Were you mistaken with your original suspicion that the flashlight was ON? Have you now actually turned the flashlight ON? There is no sure-fire way to confirm that you have flashlight in the OFF position without unpacking everything in the backpack, and **seeing** the current state of the flashlight.

As you might suspect, modes are the root cause of problems with toggle controls. Consider the two modes, diagrammed in Fig. 1.27. There are multiple different states that the object of interest can be put into that, under the identical user action, produce unique outcomes.

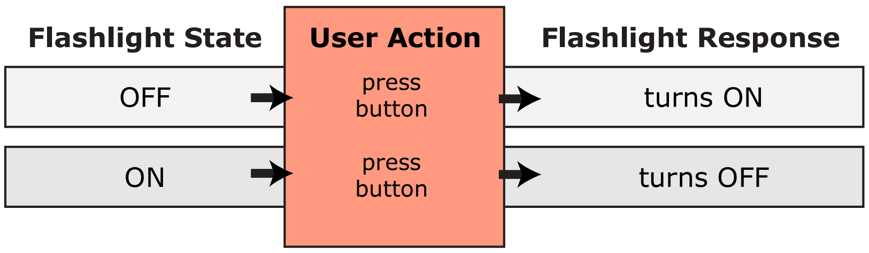


FIGURE 1.27: The toggling flashlight in the context of the mode definition: The same user action produces two different outcomes depending on the mode.

A simple solution to this problem would be to change the push button control to a slider control. With a small amount of familiarity with the flashlight, the user would learn that the slider pushed completely toward the head of the flashlight turns the light ON; whereas, the slider pushed complete toward the tail of the flashlight turns the light OFF. Note how these two configurations reinforce the subtle notion of "toward the light bulb" creates the light; "toward the batteries" stores the light. Even through the material of a backpack, the user could articulate the slider control forward and backward, leaving the slider in the backward position and confirming the current state of the flashlight to be OFF.

If toggles must be used, there are ways to mitigate the negative side effects they produce. First, the toggle can be given both a planar and depressed state. The button's elevation in the planar configuration is even with the elevation of the material surrounding the push button control. This is the OFF position. The user then presses and releases the button. The button moves in and out, but comes to rest at an elevation lower than the surrounding material. This is the ON position. The difference in elevation of the push button helps to differentiate between the two states. Second, an indicator light can be added to the button region. When the button is in the OFF position, the indicator light is extinguished. When the button is in the ON position, the indicator light is illuminated. These two embellishments to the standard push button toggle make it much more "user friendly."

One final but important note on this example. The so-called **blind test** is an excellent tool to evaluate between a mediocre user interface and an excellent user interface. Basically, ask yourself if a blind person, after being informed on how to articulate the controls of a device, could successfully navigate the interface. As you can see in this example, the simple toggle control *fails* the blind test, even if the indicator light embellishment is used.

Only the toggle with the planar/depressed configurations passes the blind test, and it does this because its embellishment makes the toggle behave more like a slider control. Notice that the slider control passes the blind test without any embellishment.

It should be mentioned the likelihood of any application's graphical user interface passing the blind test is practically zero. For example, checkboxes, which operate as toggles, are used extensively throughout many modern-day applications. Checkboxes unquestionably fail the blind test, whereas a switch or slider passes. However, compared to switches and sliders, checkboxes consumer considerably less user interface real estate, a resource that is often scarce in interface design.

The logical conclusion then must be this: The art of interface design must accommodate a balance of competing interests. The blind test may offer ways to improve clarity of the user interface, while a simple checkbox may provide a resource-efficient alternative deemed acceptable by designers and users alike.

1.16.5 Example: UNIX command line errors

Bradford *et al.* published an analysis of errors committed by users of the UNIX command line. For details of the study, see [Bradford, 1990]. The researchers logged 6,112 erroneous commands made by users. Of this pool

- 48% of the errors were mode errors caused by users invoking commands on the wrong state of the operating system,
- 38% of the errors were spelling errors, and
- 14% of the errors were miscellaneous errors.

Here is an interesting example of a large study implicating modes as the most significant source of user error. Cooper and Reimann broaden this topic, contending that mode errors on a command line environment may be more prevalent than in a graphical user interface. "In a command-line environment, modes are indeed poisonous. However, in the object-verb world of a GUI, they aren't inherently evil, although *poorly designed* modes can be terribly frustrating." [Cooper, 2003] Applying Cooper and Reimann's comment, we might conclude that the incidence of a user committing a mode error in a *graphical* user interface would be less than 48%, the number cited by Bradford *et al.* for the *command line* interface.

1.16.6 Quasi-Modes

While typing, I can produce identical results with both the CAPS LOCK key and the SHIFT key: THE LETTERS I TYPE TURN OUT TO BE CAPITALIZED. The difference between the CAPS LOCK key and the SHIFT key is that the former induces a mode whereas the latter does not. Holding the SHIFT key down while typing is an example of a **quasi-mode**.

There is a subtle yet important difference between modes and quasi-modes. Both elicit two unique responses from the software. Modes use multiple software states and one user action. Quasi-modes use a single software state and multiple user actions. Quasi-modes not only eliminate the burden of the user needing to figure out the current mode, they also reinforce the one-to-one relationship between a certain gesture and a certain outcome. Figure 1.28 illustrates this important difference with the CAPS LOCK key and SHIFT key example.

For cases when a user wishes long sentences or paragraphs to appear all in capital letters, the quasi-mode solution, however, is not practical. Indeed, this exact use case seems to be the most logical reason to have the CAPS LOCK key in the first place. Though the quasi-model solution for typing capital letters may be, at times, unreasonable, it is meant to illustrate a point—quasi-modes offer an alternative to a modal design.

Quasi-modes have been used successfully in many software applications. For example, MSC.visualNastran 4D 2003, software for motion and stress analysis of 3D solid-geometry assemblies, makes use of quasi-modes for its view manipulation. There is only one standard view mode. If the user presses the middle mouse button down, drags the mouse, and then releases the middle button, the assembly in the view rotates. If the user makes the same gesture but in addition, presses and holds the CTRL key, the view pans instead of rotates. This type of view manipulation with the use of quasi-modes has been met with strong user endorsement. Moreover, the user does not get "stuck" in a particular view manipulation mode.

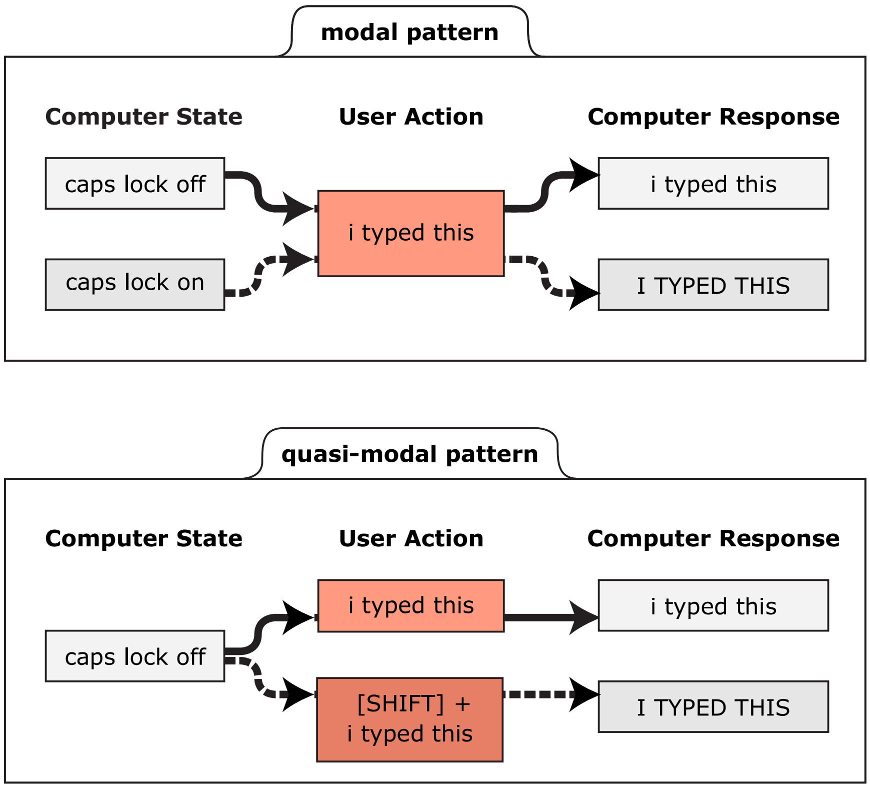


FIGURE 1.28: Demonstration of identical responses with a modal and quasi-modal design.

1.16.7 Modeless Design

Modeless designs avoid the use of modes entirely. Such designs allow the user's workflow to proceed unobstructed by the particular mode. Modal designs keep a one-to-one correspondence between user action and program response.

A modeless design in the context of the mode definition has just one state, unique user actions and corresponding unique responses. This pattern, shown abstractly on the right in Fig. 1.29, stands in contrast to the modal pattern, shown abstractly on the left in the same figure.

1.16.8 Taming Modes

There may be situations when the use of modes is acceptable. In desktop publishing programs, Adobe Illustrator for example, it may be useful to have the user in "paint mode" to do a number of repeated paint-fill operations on many different regions. It is inefficient

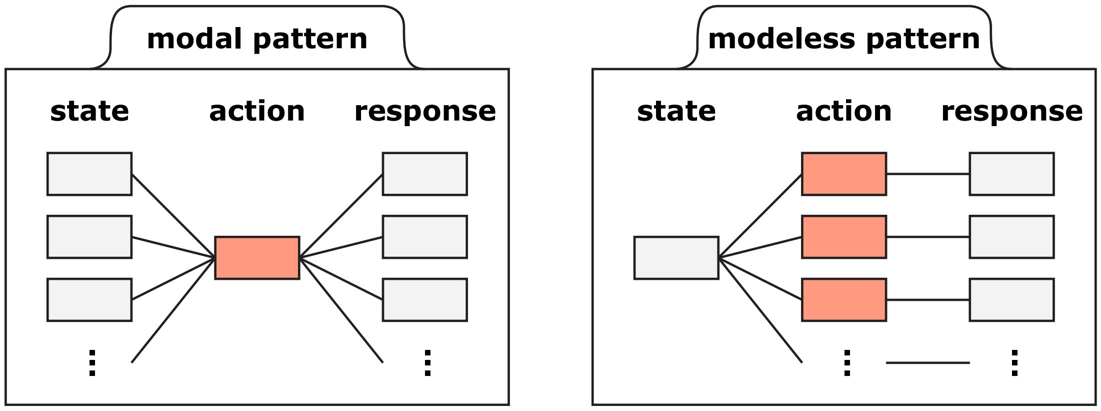


FIGURE 1.29: The abstract pattern of a modal design (left) and in contrast to the abstract pattern of a modeless design (right).

to have the user repeatedly pick the paint tool for each of the numerous paint operations. Additionally, holding down a quasi-mode key may be cumbersome.

If modeless designs and quasi-modes do not adequately meet the design goals, and a modal design seems to be the only satisfactory alternative, consider this: Tame the mode by trying to preempt the mode error. At a minimum, the mode should incorporate visibility and escapability:

- **Visibility**—bring modes into the user's Locus of Attention (see Section **??**). The visibility should continuously remind the user of the mode.
  - **Mode status notification**—A nice example of mode visibility is shown in Fig. 1.30. The transparent overlay buttons shown in this figure augment the status bar in the lower left-hand corner of the application window.
  - **Cursor updates**—Another example is to change the look and feel of the cursor from the standard cursor (often an arrow in many desktop publishing and CAD packages) to a cursor that reflects the mode. Microsoft Paint, for example, uses a "bucket pouring paint" cursor when the program is in the "fill region" mode. MSC.visualNastran 4D changes the cursor from the arrow icon to a rotate icon, for example, when the user has selected a "sticky" view manipulation command.
- **Escapability**—modes, especially modes alternative to the standard mode (e.g., CAPS

LOCK), should always be escapable. The user should never feel "stuck" in a mode.

- ESC **key escapes the mode**—It is natural that the user should hit the ESC key to exit the non-standard mode; the application should provide this feature.
- **Reversible process**—Escaping from the mode should have no deleterious effects on the user's processes and data—applications should not shut down and the user's work should not be erased.

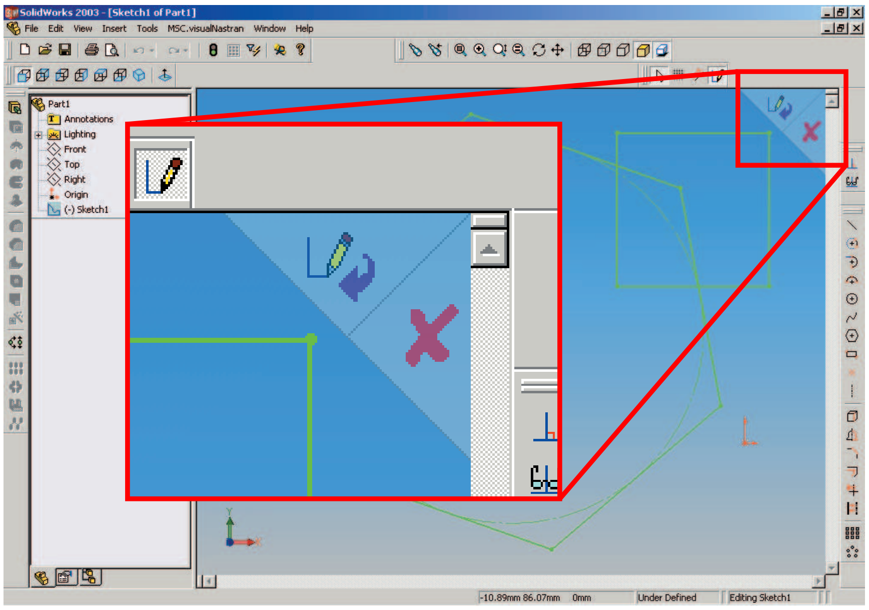


FIGURE 1.30: Demonstration of mode visibility: A screen shot of SolidWorks 2003, software for constructing 3D solid-geometry models, in SKETCH MODE. A semi-transparent overlay appears in the upper right-hand corner of the view when the user is in SKETCH MODE. The overlay has two regions that serve as buttons to either escape from the sketch mode and return to the standard mode (the pencil with the arrow), or to discard all modifications to the sketch and then return to standard mode (the 'x').

1.16.9 Measure of Modelessness

Since we have discovered that modeless software user-friendly and modal software is potentially user-hostile, it would be helpful to quantify exactly how good a particular user interface feature or scenario is. The lack of modality for a software application can be quantified by the following simple measure: Let n_{modes} be the number of modes that, under the same user action, gives distinct responses. Note that n_{modes} must be an integer in the domain $[1, \infty)$. Then the **Measure of Modelessness** is defined as

$$M_{\text{modelessness}} = \frac{1}{n_{\text{modes}}}. \tag{1.2}$$

Let's abbreviate $M_{\text{modelessness}}$ as simply M for the remaining discussion. Note that M is in the range $(0, 1]$. Software that has $M = 1 = 100\%$ is completely modeless and considered the best design possible. Software that has $M \rightarrow 0$ means the software is nearly completely modal.

Table 1.3 summarizes the calculated value for M for each of the foregoing examples.

Section No.	**Example**	M
1.16.1	car crash	25%
1.16.3	CAPS LOCK	50%
1.16.4	toggling flashlight	50%
1.16.6	SHIFT key + keystroke	100%

TABLE 1.3: Tabulation of the measure of modelessness for each of the foregoing examples in this section. The closer the value of M to 100%, the better.

1.16.10 Summary

- Modes, although commonly used, can represent a bad design if they often cause users to make mode errors.
  - Mode errors frustrate users—the expectation and outcome get mismatched.
  - Mode errors amplify the amount of work required by the user. Users must undo and redo their work when a mode error occurs.
- Quasi-modes and modeless designs are alternatives to modal designs.
- An exceptional user interface design will pass the blind test, which states that a blind person, once accustomed to the interface, could operate the interface without problems.

- Toggles are inherently modal and thus can lead to mode errors. A slider or a switch is a good alternative to a toggle. Toggles do not pass the blind test; sliders and switches do pass the blind test. However, sliders and switches consume considerably more user interface real estate than toggles.

- The degree to which a design is modal can be quantified with the Measure of Modelessness.

1.17 Monotony

AUTHOR: C.B. HOVEY

CREATED:

REVISED:

- Content to come.

1.18 Neutrality

AUTHOR: C.B. HOVEY

CREATED: 12 SEPTEMBER 2003

REVISED: 23 SEPTEMBER 2003

Note: Sections containing only an outline of bulleted items (in place of written text with discussion and figures) signifies that the section is currently under development.

- Overview - world is getting smaller, technology and in particular software makes the world smaller. We wish to sell software to all markets, many of which exist outside the main country and culture of development.
- Diverse development teams foster neutrality. Neutrality fosters user acceptance.
- Definition of Neutrality - to come.
- Neutrality violations often occur in icons, idioms, metaphors, and text—links neutrality with the user interface. Metaphors often do not cross cultural boundaries particularly well.
- Example: Intel Corporation
  `http://www.intel.com/intel/finance/prin_resp_bus.htm`
  Intel respects, values and welcomes diversity in its workforce, its customers, its suppliers and the global marketplace. Intel will comply with applicable laws and provide equal employment opportunity for all applicants and employees without regard to race, color, religion, sex, national origin, ancestry, age, disability, veteran status, marital status, sexual orientation or gender identity. This applies to all areas of employment. Intel also provides reasonable accommodation to disabled applicants and employees to enable them to apply for and to perform the essential functions of their jobs.
- **Age Neutrality** - avoid associating “old” with “bad” and associating “new” with “good.” “Old” could mean “outdated” or “antiquated;” it would also mean “wise”

or "experienced." Likewise, "new" may mean "fresh" or "modern", but may also mean "naive" or "inexperienced." Avoid passing a morality (good vs. bad) on "old" and "new." Stay objective, stating time passage by dates and times.

Example of age neutrality – Instead of "Replace the old file with file with the new file," use "Replace the existing file with this one?" – example from the Windows 2000 operating system.

- **Cultural Neutrality** In Anglo cultures, the owl symbolizes wisdom and knowledge. Owls used with librarians. Owls used with marketing campaigns to decrease pollution "Give a hoot, don't pollute." In Latin (specifically Mexican) cultures, the owl symbolizes witchcraft and the supernatural. Using an icon with an owl may make sense in an Anglo culture, but might communicate an erroneous message in a Latin culture.
- **Cultural Neutrality** Regarding radio boxes, check boxes, etc., in Japan, a check box with an 'x' (e.g., [x]) signifies a *deselected* item (Cite Alan Cooper). In the Windows operating system, the 'x' in the dialogs and windows means "to close the window."
- **Economic Neutrality** Avoid associating "poor" with "bad" and associating "rich" with "good." E.g., "Using an explicit time stepping scheme with a stiff system may cause drift and therefore provide a *poor* approximation to the system's behavior."
- **Language Neutrality** Chevrolet Nova. Example of cross-cultural insensitivity. The words *no va* in Spanish translate to *it doesn't go*. Why didn't the Chevrolet Nova sell well in Spanish speaking countries?
- **Language Neutrality** and **Regional Neutrality** The "Stop" sign as an icon that is not neutral. Insert picture of octagonal red sign with "S T O P" written on it. In Puerto Rico, for example, the signs are octagonal and red, but read "P A R E" instead (language neutrality). In Japan, the stop signs are not octagonal, instead they are triangular (regional neutrality).
- **Language Neutrality** - avoid slag and idiomatic expressions, e.g., "as the crow flies," "kill two birds with one stone," or "leave no stone unturned."

- **Political Neutrality** - avoid attaching good and bad to “conservative” or “liberal”. Instead of “Your mesh size estimate is too conservative,” try “Your mesh size estimate is too small.” Instead of “Make liberal use of elements at points of stress concentration,” try “Use a relatively higher number of elements at points of stress concentration.”

- **Gender Neutrality** - At one time, students were taught to use “he” as a personal pronoun when the gender could be taken to use either “he” or “she.” This teaching is antiquated. At one time, most engineers in the United States were males, and perhaps in that era (e.g., 1950s), the exclusive use of “he” was accurate.

  Suggestion: Replace personal pronouns with more specific, gender-neutral pronouns, such as “the user” or “the engineer.” Some authors prefer to randomly interchange their use of “he” and “she” throughout their text, though this might lead to confusion if the text is not written carefully.

- Summary

  – Cross-apply Neutrality to Affordance, Icons, Idioms, and Metaphors.

  – Cross-apply with Regionalism, Localization, and Globalization.

1.19 Plastification

AUTHOR:

CREATED:

REVISED:

- Content to come.

1.20 Reversibility

AUTHOR: S. GOODSON

CREATED: 21 OCTOBER 2003

REVISED: 06 NOVEMBER 2003

1.20.1 Definition

Reversibility refers to the ability of the user to cancel or annul the effects of their actions. It is most commonly provided as some form of undo/redo. It doesn't directly help the user accomplish a goal, but mitigates the cost of mistakes and wrong turns. Perhaps more importantly, it also facilitates exploration. The user is free to experiment in order to learn how the program works, and to try out possible changes to their document or data. As a side benefit, a robust undo facility also removes any need for annoying "Are you sure" dialogs.

1.20.2 Types of Undo

- **Blind vs. Explanatory**—Blind undo provides no indication of what action will be undone. It is often more pleasant to use an explanatory undo, which provides a textual description of the action to be undone.

- **Single vs. Multiple**—Single undo is simpler, both to implement and for the user to understand, and is adequate in many situations. There is usually no separate redo; a second undo undoes the original undo. The obvious disadvantage is that the most recent change may not be the one the user wishes to undo. This problem is addressed by multiple undo, which stores a sequence of the most recent actions. Actions are undone and redone in a strict linear order. While clearly an improvement, multiple undo also has some problems from the user's perspective. It can be cumbersome to undo a long sequence of operations, and the user may be forced to throw away some desirable changes in order to undo a previous undesirable operation.

1.20.3 Other Reversibility Mechanisms

- **Category specific undo**—Undo/Redo is typically a global operation in the program. The previous action is undone regardless of where it occurred. In a component based program, this behavior requires the explicit cooperation of all the different components. An alternative to having a single global undo would be to have different categories of independent undo-able actions. For example, the user might be offered the ability to undo the previous formatting operation, or the previous text insertion, or even the previous action on the current selection, without affecting operations in the other categories. Such a feature is not commonly implemented, and while it does improve on one weakness of multiple undo, users may find it a bit too unfamiliar.

- **Deleted data buffers**—Users commonly employ undo in cases where they have deleted some data, but for some reason they wish to recover or view that data again at a later point. This goal could be more easily accomplished if the program stored all the deleted data and allowed the user to browse through it. Rather than retracing their steps using multiple undo, the user could simply go and get the deleted data directly. This would be especially useful when there's a large number of intervening operations the user doesn't wish to discard.

- **Milestoning**—Users occasionally want to back up long distances to return to a previous known state. In these cases, the fine grained control of multiple undo is unnecessary and cumbersome. A convenient alternative is a milestoning mechanism that takes a complete copy or snapshot of the user's data, either at the user's request or at some other significant point. The program then keeps track of the copy, and can later offer the user the option of reverting to one of the previous milestones.

References: Chapter 12 of About Face 2.0 [Cooper, 2003]

1.21 Selection

AUTHOR: S. MACMILLAN

CREATED: 02 SEPTEMBER 2003

REVISED: 02 SEPTEMBER 2003

```
Statement
----------
A task to do one thing once should not be N times
as long and hard for N items.

The picking and selection methods will allow actions to be
applied to collected sets of entities in a timely and appropriate manner.

Picking   :: Interactive

       User wants to initiate an action on a set of objects that they have
       just, or are about to , select "on the fly".

            : Shapes : Polygon, bounding box, circle etc
              : Add/Remove region selected once created
                   ( Control/Shift modifiers )
                + is this reasonable in a 3d model as it is in a 2d space
                   like photoshop??
            : Add/Remove Modify contents
              : How long is selection stored, do we name it ?
                    ( How are we going to be naming/referencing things in general )
                 ( Do we update the contents as they are "Selected" or
                    when action is about to be performed.
                     ( User could have particular shape selcted and want to use
                       it as a snapshot to different parts of the model )
              : Do we have a quick interactive tool and then a
                "more.." type tool like searching in email programs
                to allow more options.

            : Filtering of entities to be picked
```

```
              : Dimension; Type; Properties ( Material; Thickness; Mass etc..)
            :
            : Dimensional picking; picking volumes ??
            :
            : Motor control of users.
              : zooming of selection regions etc
                 ( Magnifying glass next to pointer etc )
            : Realistic representations of objects
            : Levels of Detail ( LOD )
            :
            : Feedback to users of current entity state.
               : Pre picking hilighting
               : Picked object reinforcement

Selection :: Superset including describable sets

      ( Within graphical window for model interaction ; there is also selection
        from lists of items which has overlap to some of these ideas;
        ie uing describable actions; filters; treemaps etc )

        User is creating sets of objects to be stored for some user determined  period
          ( work session, db lifetime ... ) . These sets can be reused and
          manipulated to have actions performed on them.

                * eg Nodes AND elements ON surface BETWEEN -5 < x < 0 etc
                * eg ALL internal FACES of SOLIDS  ( for pressure loading say )
                *
                * What about masks for storing interactive picks
                    : To allow reapplication
                    : To remove user concern over losing pick
                *
                * Manipulation of selected sets ie addition/subtraction/boolean operations
                *

Review: CAE,Graphics, Animation packages and Games,  Other

       : Patran, Hypermesh, Ansys, Adams
```

```
          * Patran; Interactive;Active Set, Lists; Groups
          * Hypermesh collectors
          * Adams; Selection tool via names
       : Photoshop
          : Selection tools for "cropping" add/remove of regions
          : Masks
          : Selection from long lists ( ie Fonts )
       : Maya, Animation Master, 3D Studio Max, LightWave 3D, Softimage
       : Sims ( Pie menus ) , Diablo, etc
          : Actions on an entity
       : Firebird with Extensions ( Tabs/pies )

NB: We need to come up with a vocubulary for these different things that will
    enable us to communicate correctly about them.
   ( sets/groups/names/selected/lists  etc etc )

NB the picking/selection tools have to interact alongside the other tools that
   are manipulated from the pointing device.ie rotations etc.
   Users must be able to know where they are and not lose functionality because
   of incorrect interactions.
```

1.22 Toggles

AUTHOR:

CREATED:

REVISED:

- Content to come.

Notes

- Blind test
- Flashlight

1.23 User Level Myth

AUTHOR:

CREATED:

REVISED:

- Content to come.

1.24 Visibility

AUTHOR:

CREATED:

REVISED:

- Content to come.

1.25 Vocabulary Part 1—Noun-Verb and Verb-Noun Gestures

One could think in an abstract sense of human-machine interaction (HMI) as a type of discussion going on between two parties. The first party, the computer in this case, takes input from the second party, the user, in the form of gestures, such as keystrokes and mouse clicks. The user input gets processed and synthesized by computer applications into an output—perhaps in the form of a document or print out. There is a certain action-reaction dialog that transpires between the user and machine. By studying this dialog, user interface designers can learn what frustrates users, what pleases users, and why.

In the human-machine dialog, one can break down many of the tasks a user completes into a single sentence. This sentence will have both a noun and a verb. In the written language, the noun is generally a person, place, thing, or idea; the verb is generally an action or a state of being. In the human-machine language, the noun is generally the item being acted upon by the user, and the verb is the particular action the user imparts on the noun. The following example illustrates this concept.

1.25.1 Example: Bold Text

Let's say a person is using a word processing application to transcribe an audio recording of President John F. Kennedy's address at Rice University on the United States space effort. The person might want to put a sentence in bold text to indicate the emphasis of the speaker. An excerpt of the text would appear as follows:

> The exploration of space will go ahead, whether we join in it or not, and it is one of the great adventures of all time, and no nation which expects to be the leader of other nations can expect to stay behind in the race for space.
>
> Those who came before us made certain that this country rode the first waves of the industrial revolutions, the first waves of modern invention, and the first wave of nuclear power, and this generation does not intend to founder in the backwash of the coming age of space. **We mean to be a part of it–we mean to lead it.** For the eyes of the world now look into space, to the moon and to the planets beyond, and we have vowed that we shall not see it governed

by a hostile flag of conquest, but by a banner of freedom and peace. We have vowed that we shall not see space filled with weapons of mass destruction, but with instruments of knowledge and understanding. [Kennedy, 1962]

1.25.2 Definitions

Now consider the following three possible ways in which the user could have created this paragraph. I assume the existence of a "bold" button contained within a formatting pallet of the word processing application. I also assume the "bold" button to have a toggle (OFF / ON) operation.

1. **Noun-Verb** The user first selects the noun and then selects the verb. In words, the user does "sentence–format it". The user
   (a) types the two paragraphs completely,
   (b) selects the "We mean to be a part of it..." sentence,
   (c) clicks the "bold" button.
2. **Verb-Noun** The user first selects the verb and then selects the noun. In words, the user does "format–the sentence". The user
   (a) types the two paragraphs completely,
   (b) clicks the "bold" button (hopefully, the cursor changes from a "selection" icon to a "paint-the-bold" icon to reinforce the user's cognition of the current mode),
   (c) selects the "We mean to be a part of it..." sentence.
3. **Modal Operation** Though not meant to be the focus of the discussion here, modal operation is yet another way in which the example paragraphs can be created. For a discussion on modes and their potential pitfalls, see Section 1.16. For completeness, the steps are included here. The user
   (a) types up to the sentence ending in "...coming age of space.", and then
   (b) clicks the "bold" button (toggle ON),

(c) types the "We mean to be a part of it..." sentence,

(d) clicks the "bold" button (toggle OFF),

(e) types the remaining sentences.

On the surface, all three methods seem like perfectly reasonable means to an end. I agree. Moreover, I advocate that well designed software should accommodate for all three of these use cases—the software should not box the user into a corner. *However*, the need often arises for a particular implementation to embrace one of these three methods. In general, the noun-verb construction should be preferred over the other methods. The noun-verb pattern is more efficient and more robust than the other patterns. By efficiency, I mean how quickly can the user accomplish a goal, getting from point "A" to point "B". By robustness, I mean how solid is the application along the path from "A" to "B". Along this path, efficiency is the quantity, robustness is the quality.

The efficiency is evaluated in terms of (a) GOMS timing, (b) Locus of Attention, (c) toggle interpretation, and (d) mode error timing. The interplay of modelessness with Locus of Attention demonstrates the robustness.

1. **Efficiency**

   (a) **GOMS timing**—The GOM keyboard-level model (see Section 1.8.2.1) applied to each of the three use cases produces some interesting results. We see that the noun-verb and verb-noun yield identical theoretical timings. The modal operation produces a more efficient timing, as expected, since some of the time gets absorbed by the mode itself. Interestly, if only the keyboard is used in the modal operation, the theoretical timing drops by more than 50% because many of the homing and pointing inefficiencies of the mouse are eliminated.

      i. noun-verb: (step 1b = $HMPKMPK$) + (step 1c = MPK) = 8.35 seconds.
      ii. verb-noun: (step 2b = MPK) + (step 2c = $HMPKMPK$) = 8.35 seconds.
      iii. modal: (step 3b = $HMPKH$) + (step 3d = $HMPKH$) = 6.90 seconds.
      iv. modal (keyboard only): (step 3b = MK) + (step 3d = MK) = 3.10 seconds.

   (b) **Locus of Attention**—The noun-verb pattern is more efficient than the verb-noun pattern because the former requires one fewer change of the user's locus of attention. Considering each change in locus of attention ΔL to be equivalent to mental preparation step M ($M = 1.35$ seconds) of the GOMS timing, the following calculations are made:

i. noun-verb:
   - L_1 = sentence–find and select,
   - L_2 = "bold" button–find and select,
   - $L_2 - L_1 = \Delta L$ = 1.35 seconds.

ii. verb-noun:
   - L_1 = sentence–triggers need for the bold format,
   - L_2 = "bold" button–find and select,
   - L_3 = sentence–find and select,
   - $L_3 - L_1 = 2\Delta L$ = 2.70 seconds.

(c) **Toggle Interpretation**—Both the verb-noun and modal operation use cases requires the interpretation of a toggle state. For the user to invoke the bold ON state, the user must verify the toggle is in the OFF state prior to taking action. This interpretation is required only once for the verb-noun case (item 2b), but is required twice for the modal case (item 3b and 3d). Again, using a toggle interpretation to be one mental preparation step M:

i. verb-noun: 1.35 seconds.

ii. modal (either keyboard alone or keyboard + mouse): 2.70 seconds.

(d) **Mode Errors**—Since both the verb-noun and the modal operations set the application in a mode, the user is susceptible to mode errors. The amount of time required for a user to recover from a mode error is highly variable and thus not easily quantified. For this particular mode error, recovery time is likely to be on the order of M, defined above. The recovery time is scaled linearly by the number of modes to give a total estimated time.

i. verb-noun: $2 \times M$ = 2.70 seconds.

ii. modal (either keyboard alone or keyboard + mouse): $2 \times M$ = 2.70 seconds.

2. **Robustness**—In contrast to efficiency, robustness is more difficult to articulate. An application is robust if the perceived quality of the application is not easily undermined. The Measure of Modelessness for each of the three use cases is 100%, 50%, and 50% for noun-verb, verb-noun, and modal, respectively. This measure is linked to robustness. Once trapped in a mode error, the user experiences an expectation-outcome mismatch (see Section 1.16.1). Ultimately, the user blames the application for this mismatch, since the user issued a command (which worked previously when the application was in the right mode) and the application failed to properly process that command. The noun-verb pattern provides a modeless operation that prevents the user from ever walking into the mode error trap.

Now apply this context to the bold text example. The noun-verb pattern minimizes errors because the state change (e.g., from non-bold to bold) occurs with the user's Locus of Attention (see Section **??**) is on issuing the command (the verb). In contrast, the verb-noun pattern invites errors because the state change occurs when the user's Locus of Attention is on the noun (the verb has already been preselected). If there is a minor disruption in the user's work flow, the user may forget the verb preselection, make a selection, and witness unexpected results. The basic drawback of verb-noun is that it sets up a mode—after clicking the "bold" button, next user command will be in the bold text mode.

	noun-verb	**verb-noun**	**modal (k+m)**	**modal (k only)**
GOMS timing	8.35 s	8.35 s	6.90 s	3.10 s
Locus of Attention	1.35 s	2.70 s	–	–
Toggle Interpretation	–	1.35 s	2.70 s	2.70 s
Mode Errors	–	2.70 s	2.70 s	2.70 s
Total	**9.70 s**	**15.10 s**	**12.30 s**	**8.50 s**

TABLE 1.4: Tabulation of time efficiency calculations for four use cases of making text bold. Note that (k+m) denotes the use of keyboard and mouse, whereas (k only) denotes the use of the keyboard without use of the mouse. Note how efficient the modal (k only) case can be if the user is successful in avoiding mode errors.

1.25.3 Example: MSC.visualNastran 4D Paint-the-Constraint

In 2002, MSC Software released a version of MSC.visualNastran 4D (vN4D) that touted a new, ease-of-use feature called "Paint the Constraint." This feature, an embodiment of the verb-noun pattern, was said to be more efficient for the user in some cases than the traditional method of creating mechanical constraints, which was a noun-verb construction. The efficiency claim was based on use cases such as the following: Given a simulation with two bodies in the world, the user wishes to connect point P_1 on body B_1 with point P_2 on body B_2 with a spring. The vN4D application provided a "Paint the Constraint" palette (Fig. 1.31) that the user could "dip" the cursor into, and then proceed to paint the two endpoints of the spring—one point on each body. The application would dynamically update the selection cursor after the "Paint the Spring" palette item was pressed. After the palette selection and prior to the first click, the application showed a spring constraint cursor (Fig. 1.32–left) and a status bar informing the user to pick the beginning point of the spring. Between the first and second clicks, the application showed a slightly different spring constraint cursor (Fig. 1.32–right) and a status bar informing the user to pick the second and final point of the spring.

The best-case scenario for the "Paint the Constraint" feature was this:

1. H—User moves hands from keyboard to mouse,
2. P—points to the spring button (assumed already the "current" button)
3. K—clicks the spring button and invokes a "Paint the Spring" mode
4. PK—points to and selects point P_1 on body B_1,
5. PK—points to and selects point P_2 on body B_2.

Applying GOMS timing gives $HMPKMPKMPK$ = 8.35 seconds. Compare this to the traditional noun-verb approach:

1. H—User moves hands from keyboard to mouse,
2. PK—points to and selects the "coord" button (a "coord" is a term used by vN4D to denote a marker or a position in space),

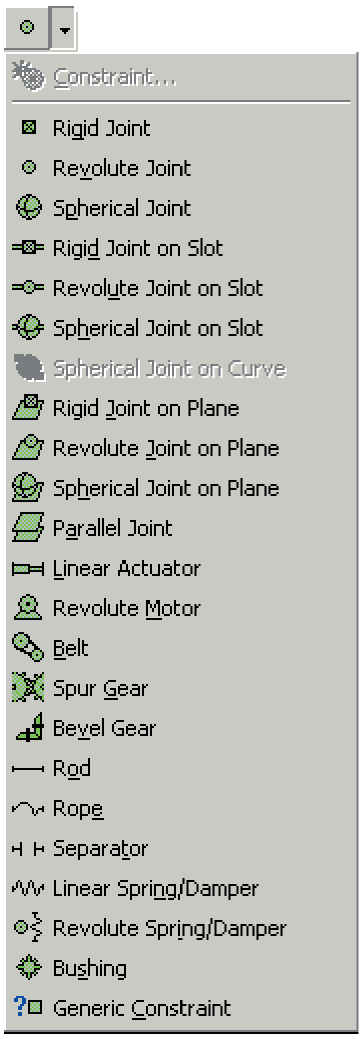


FIGURE 1.31: The "Paint the Constraint" palette of MSC.visualNastran 4D.

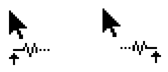


FIGURE 1.32: The dynamic cursor update in MSC.visualNastran 4D while painting the first point of a spring constraint (left image) and painting the final point of the spring constraint (right image).

3. PK—points to and selects point P_1 on body B_1,

4. PK—points to and selects the "coord" button again,

5. PK—points to and selects point P_2 on body B_2,

6. H—holds down the SHIFT key for multi-selection,
7. PK—points to and selects point P_1 on body B_1,
8. PK—points to and selects the "constraint" button,
9. PK—select "spring" from the list of the constraints
10. PK—point to and click the "OK" button to create the spring constraint

Applying GOMS timing gives $H + 4(MPK) + MH + 4(MPK)$ = 0.4 + 4(2.65) + 1.75 + 4(2.65) = 23.35 seconds. Comparing these two use cases, the noun-verb construction takes approximately three times longer than the verb-noun counterpart. The savings of the "Paint the Constraint" feature arise because the two endpoint "coords" are automatically created when the user selects P_1 and P_2.

Here is an example of a use case where based on expected timing alone (ignoring delays caused by shifts in locus of attention, toggle interpretations, and mode errors), the verb-noun pattern actually serves the user better than the noun-verb pattern. Indeed, it was the use cases such as these that motivated the "Paint the Constraint" feature. However, there remain many common use cases where the verb-noun is either less efficient or impossible. Consider the following limitations:

- **Verb-noun efficiency is undermined if view manipulation is required.** Verb-noun requires that both points P_1 and P_2 be visible to the user in the view. If only a single point is visible at one time, the user is forced to *suspend* the "Paint the Constraint" gesture and issue one or more view manipulation commands to bring the second point into view. The "select point, view manipulation, select point" scenario can become less efficient and certainly less robust than simply preselecting all the nouns and then applying a single verb to them. With verb-noun and view manipulation, the user becomes prone to mode errors, not knowing whether a click of the mouse will select a point or rotate the view, for example. Moreover, the user may accidentally abandon the "Paint the Constraint" mode for example, by pressing ESC one too many times, escaping from both the nested view manipulation mode and the painting mode. Consequently, the user will go to click the endpoint of the constraint, only to observe the beginning point has been lost.

- **Verb-noun is restricted to pickable entities only.** If the user wishes to create a constraint between any point P_i that is on the *interior* of the body, the user cannot use the "Paint the Constraint" feature. This is a limitation to **all** painting-type gestures. Point or feature selection on the interior of bodies is a common CAE operation. Thus, "Paint the Constraint" (and thus the verb-noun) might be more efficient for some special cases, such as demonstrations at conferences in front of potential customers. However, for most of the use cases, the "Constraint-Paint It" pattern (noun-verb) is more robust.

- **Verb-noun does not scale as efficiently as noun-verb.** Consider a table consisting of two columns of data. The first column contains the beginning points of the constraint; the second column contains the ending points of the constraints. Once the table has been created, applying constraints between all pairs of (beginning, end) points is as simple as selecting the two columns (the nouns) and then selecting the spring constraint (the verb). This noun-verb pattern is robust because simply adding to and removing from the list of nouns is very efficient. The noun-verb construction scales as n (noun) + 1 (verb). In contrast, the verb-noun construction scales as n (verb + noun).

  For concreteness, consider the action of applying a pressure load (the verb) to 100 faces (the noun). The noun-verb pattern would require 101 selections (100 clicks to select the faces and then a single click to apply the pressure). The verb-noun pattern would require 200 selections (select the pressure, then select the face, repeated 100 times). Of course, this calculation assumes the verb is a "single shot" operation. Alternatively, the application could be put into a "apply pressure" mode, which then would allow the user to do a single verb selection followed by 100 noun selections. So with the use of a mode, we can regain the smaller 101 selection number. However, trouble then arises when the user accidentally selects an unwanted face *while in the select face mode*. To deselect the face that just received the paint-the-pressure action, the use must *escape from* the paint the pressure mode, erase the erroneous applied pressure, and then reenter the pressure mode to continue selecting the remaining faces. The scenario portrayed here is perhaps overly optimistic too, as it assumes the user perfectly navigates the two modes and does not commit any mode errors in the process. Additionally, the number of changes in locus of attention causes the user unnecessary mental gymnastics. The user must

- L_1 = focus on the painting operation,
- L_2 = focus on the face with the unwanted pressure,
- L_3 = escape from the paint mode,
- L_4 = delete the unwanted pressure,
- L_5 = reinvoking the paint mode,
- L_6 = focus again on the remaining faces.

In contrast, the number of locus of attention changes with the noun-verb pattern is much smaller. Erroneously selected faces never get coupled to the paint operation. The user just

- L_1 = focuses on assembling all faces, added and removing if necessary,
- L_2 = applies the pressure to the selected faces.

Cross-applying the previous estimates for shifts in locus of attention, the expected timings for these two scenarios are 6.75 seconds for the verb-noun case and 1.35 seconds for the noun-verb case. Again we see an example where locus of attention changes propagate less efficient, more cumbersome user interfaces.

1.25.4 Example: Hewlett-Packard reverse polar calculators

Hewlett-Packard calculators popularized a way of entering numbers and calculating results that differed from most calculators. Hewlett-Packard used the reverse polar notation (RPN). RPN centers around a stack (the nouns) and provides various operations (the verbs) such as enter, add, subtract, multiply, etc., that pop and push the stack. RPN is distinctly a noun-verb pattern because the numbers that are to be operated on are entered first, the operations on those number are completed second.

When using a RPN calculator for the first time, a person usually obtains results that don't make sense to them. A user thinks of addition, for example, as $3 + 4 = 7$, and accordingly makes keystrokes as

[3], [+], [4], [=]

A RPN calculator requires a slightly different input syntax:

[3], [ENTER], [4], [+]

This scenario shows an excellent example of the habitation principle (see Section **??**). For one reason or another, we are accustomed to reading $3+4 = 7$, as opposed to 3, ENTER, $4, +$. This habit, in the way a user thinks of addition, makes it natural for a user to enter keystrokes into a calculator in the same order, $3 + 4 =$. RPN calculators require users to suspend their reliance on habitation (at least until the user can develop new, RPN-like habits). But what does RPN offer in return? The answer is efficiency.

Hewlett-Packard calculators became wildly popular with engineers and physicists, largely because the stack operations provided keystroke efficiency when calculations became lengthy and were composed of several intermediate results. The same lengthy calculation of a traditional input calculator required several more keystrokes (especially the [=] key). Moreover, the absence of a stack and a single-value memory register in a traditional calculator made accounting for intermediate results nearly impossible. In contrast, with a RPN calculator, a user could easily pile up n number of intermediate results on the stack, and then simply press $n - 1$ operation buttons to compute the final result.

The opinions of RPN calculators usually fell into two polarized groups—you either loved it or you hated it. Ultimately, whether or not a calculator was RPN was one of the single most important factors in whether a user would purchase the calculator—some people couldn't live without it; others couldn't add two numbers together with it if their life depended on it!

This example illustrates a point important to the design of user interfaces. Designers must accommodate the habits and preferences of the users. As an example, an engineering analysis program that had a calculator built into it might want to offer both a standard mode and a RPN mode. The two modes would be laid out clearly on the calculator; the mode is *never* hidden from view (mode visibility). Two radio buttons would be used to indicate the mutually exclusive selection between standard and RPN mode. The calculator would default to the same mode in which it was last used.

1.25.5 Summary

- Nouns and verbs are the building blocks of human-machine dialog.
- The user employs verbs (commands) to instruct the application to take some action on the nouns (objects).
- The interaction between nouns and verbs can be implemented by
  - noun-verb ordering
  - verb-noun ordering

  or, though undesirable, through the use of a modal operation.
- A good user interface will accomodate all types of users—those who prefer noun-verb interaction, those who prefer verb-noun interaction, and those who prefer modal interaction.
- The application should prefer noun-verb interaction because it is more efficient and robust than verb-noun interaction.
  - Mode errors exacerbate verb-noun efficiency.
  - Locus of Attention shifts inherently bloat verb-noun patterns.
  - Verb-noun is restricted to pickable entries only.

1.26 Vocabulary Part 2—Attributes are Adjectives

AUTHOR:

CREATED:

REVISED:

- Content to come.

Bibliography

[Apple, 2002] Apple Aqua Human Interface Design Principles, Apple Computer, Inc., (Last Updated June 2002), (http://developer.apple.com/documentation/UserExperience/Conceptual/Aqua

[Bradford, 1990] J.H. Bradford, W.D. Murray, and T.T. Carey, *What kind of errors do UNIX users make?*, Proc. IFIP INTERACT'90 Third International Conference Human-Computer Interaction, Cambridge, UK, 27-31 August 1990, 43-46.

[Berkun, 1999] Scott Berkun, *How To Avoid Foolish Consistency,* Microsoft Corporation, September/October 1999, (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnhfact/html/hfactor8_5.asp).

[Carroll, 1990] John M. Carroll, *The Nurnberg Funnel: Designing Minimalist Instructions for Practical Computer Skills,* MIT Press, 1990.

[Carroll, 1998] John M. Carroll, *Minimalism Beyond the Nurnberg Funnel*, MIT Press, 1998.

[Cooper, 2003] Alan Cooper and Robert Reimann, *About Face 2.0, The Essentials of Interaction Design*, Wiley, 2003.

[Grayling, 1998] Trevor Grayling, *Fear and Loathing of the Help Menu: A Usability Test of Online Help,* Technical Communication Journal, Second Quarter 1998.

[Horton, 1993] William Horton, *Let's Do Away With Manuals... Before They Do Away With Us,* Technical Communication, First Quarter 1993.

[Horton, 1994] William Horton, *The Icon Book: Visual Symbols for Computer Systems and Documentation*, John Wiley & Sons, Book and Disk edition, March 31, 1994, (http://www.amazon.com/exec/obidos/tg/detail/-/0471599018/qid=1064943501/sr=1-9/ref=sr_1_9/103-3645089-3434432?v=glance&s=books).

[Houser, 1999] Bob Houser, *Should We be Writing Strategy Guides,* User First Services, 1999.

[Kennedy, 1962] John F. Kennedy, *Address at Rice University on the Nation's Space Effort*, Houston, Texas, September 12, 1962, (http://www.jfklibrary.org/j091262.htm).

[Lotus, 2001] Lotus SmartSuite User Assistance Team, *Exploring New Ideas for User Assistance: Focus Group Research at Lotus,* (http://www.workwrite.com/hm/0998/focus1.htm), June 6, 2001.

[Millar, 1998] Carol Millar, *Making Manuals Obsolete: Getting Information out of the Manual and Into the Product,* Technical Communication Journal, Second Quarter 1998.

[Miller, 2001] Bill Miller, Pat McCandless, and Edward Jones, *Rethinking the Online Help Paradigm,* (http://www.workwrite.com/hm/0998/rethink1.htm), June 6, 2001.

[Mobley, 2003] Mobley, Knight, and Meserth, *Designing Embedded Help to Encourage Inadvertent Learning,* Technical Communication, Volume 50, Number 1, February 2003.

[Mueller, 2003] Mueller, *Developing an Embedded Help Solution,* Technical Communication, Volume 50, Number 1, February 2003.

[Nielsen, 2002] Jakob Nielsen and Morgan Kaufmann, *Coordinating User Interfaces for Consistency*, The Morgan Kaufmann Series in Interactive Technologies, 1st edition (January 15, 2002).

[Norman, 1989] Donald A. Norman, *The Design of Everyday Things,* Currency Doubleday, 1989.

[Norman, 2002a] Donald A. Norman, *Affordances and Design,* (http://www.jnd.org/dn.mss/affordances-and-design.html), 2002.

[Norman, 2002b] Donald A. Norman, *Affordance, Conventions and Design*, (http://www.jnd.org/dn.mss/affordances-interactions.html), 2002. 1

[Raskin, 2000] Jef Raskin, *The Humane Interface*, Addison-Wesley, 2000. 2

[Scanlon, 2001] Tar Scanlon and Carolyn Snyder, *Beyond Help: Bridging Gaps with Affordances*, (http://www.workwrite.com/hm/0998/afford1.htm), June 6, 2001. 3

[Spolsky, 2001] Joel Spolsky, *User Interface Design for Programmers*, APress, 1st edition, June 26, 2001, (http://www.joelonsoftware.com/uibook/fog0000000249.html). 4

[Tesler, 1981] Larry Tesler, *The Smalltalk Environment*, BYTE magazine, August 1981, (http://www.byte.com/art/9608/sec4/art3.htm). 5

[Tognazzini, 2003] Bruce Tognazzini, *AskTog First Principles* (http://www.asktog.com/basics/firstPrinciples.html). 6

[User Interface Engineering, 2001] User Interface Engineering, *Docs in the Real World*, (http://www.world.std.com/uieweb/realdocs.htm), June 27, 2001. 7

[User Interface Engineering, 2001b] User Interface Engineering, *When to Develop a Wizard*, (http://www.world.std.com/ uieweb/wiz_art.htm), June 27, 2001. 8

[User Interface Engineering, 2001c] User Interface Engineering, *Making Online Information Usable*, (http://www.world.std.com/ uieweb/online.htm), June 27, 2001. 9

[User Interface Engineering, 2001d] User Interface Engineering, *Designing for Doing*, Conference Presentation, 2001. 10

[Zubak, 2001] Cheryl Lockett Zubak and Ralph Walden, *You want to embed, but where are the tools?*, Work Write, Inc., 2001. 11

[Zubak, 2001b] Cheryl Lockett Zubak, *Design Models for Embedded Help*, Work Write, Inc., (http://www.workwrite.com/hm/0998/embed1.htm), June 6, 2001. 12