Constantine & Lockwood, Ltd.

Instructive Interaction: Making Innovative Interfaces Self-Teaching

Larry L. Constantine, University of Technology, Sydney
Lucy A. D. Lockwood, Constantine & Lockwood, Ltd.

Abstract. An innovative approach to enhancing ease of use and learning for novel user interfaces is described. Instructive interaction comprises a body of techniques based on a learning-by-doing model that is supported by three design principles: explorability, predictability, and guidance. Taken together, these principles form the basis for creative designs that can support highly efficient production use by experienced users while also enabling new users to understand and make effective use of an unfamiliar system almost immediately. The underlying principles of instructive interaction are presented here and an assortment of specific techniques based on these principles is described.

Keywords: usage-centered design, user interface design, innovation, standards, learning, intrinsic help, novice users, single-trial learning, anticipatory learning

Innovation and Convention

It is a perennial design conundrum: to break new ground or to hoe the same familiar furrow. User interface and interaction design have always been pulled by two forces. On the one side are the forces of progress and innovation, and on the other are the forces of consistency and convention. Designers are encouraged to be creative and to invent improved software, yet they are also exhorted to stick with the standards and to provide designs familiar to legacy users. Users themselves plead for new systems that are easier to use, yet they balk at novelty and complain about having to learn new procedures or techniques.

Instructive interaction is an approach we developed to help reconcile these competing demands. It draws upon and integrates a diverse body of novel and long established techniques that can make even radical designs self-teaching. In our usage-centered approach to design (Constantine and Lockwood, 1999), the driving force is finding the design that best supports easy and efficient usage. Innovation is a frequent consequence of usage-centered design, because radical improvements in usability

Expanded original preprint of paper appearing in *User Experience, 1* (3), Winter 2002. The authors may be contacted at Constantine & Lockwood, Ltd., 58 Kathleen Circle, Rowley, MA 01969; tel: 1 (978) 948 5012; fax: 1 (978) 948 5036; mailto:lconstantine@foruse.com | mailto:llockwood@foruse.com

often require significant departures from previous versions and existing standards and conventions.

The techniques of instructive interaction make possible novel designs in the interest of efficient and effective user performance without sacrificing ease of learning. Indeed, the combination of instructive interaction with usage-centered design has led to genuine breakthroughs in usability (Constantine, 2002; Constantine and Lockwood, 2002; Windl and Constantine, 2001).

Beyond Help

Help facilities are supposed to ease the learning process for new users, but the help supplied is all too often not helpful. Many users, conditioned by countless frustrations, avoid supplied help at any cost, even when they are completely flummoxed by unfamiliar and unconventional features. Part of the problem is the isolation of help facilities from the user interface they are intended to support. Separate help facilities take you out of the working context, making it easy to lose track of what you were doing and difficult to connect what you learn back to where you must apply it.

It has long been our contention that users who know their own jobs should not need elaborate help or extended tutoring to use a system that supports their work (Constantine, 1991). In this light, help-seeking by users is a seen as a symptom of failure—on the part of the designers and the design. This utopian goal is actually realizable with the proper techniques. In one project, a Web-deployed classroom information management system for teachers, the system was designed to be fully usable based on a one-page tutorial (Constantine, 2001; Constantine and Lockwood, 2002).

Achieving this level of ease of learning requires that systems be designed to be self-teaching. The premise of instructive interaction is that users best learn how to use a system by using it, not through explicit or independent instruction. In this approach, help and guidance are not distinct parts of the system that are separated from the main body of the user interface. The user interface itself subtly and implicitly guides users in the correct usage of the system. The structure, appearance, and behavior of the user interface taken as a whole provide all the needed help, guidance, and instruction.

Learning by Doing

The techniques of instructive interaction are based on how human beings learn to perform tasks. People learn to do things by hearing (being told or instructed), seeing (being shown), or by doing. Although there are individual differences and preferences for one mode or another, learning by doing is usually the most important and effective, particularly for complex tasks and skills, such as using sophisticated software tools. Proficiency, it can be argued, comes only from doing and only from repetition. Indeed, at the most basic level, nearly all human learning is based on repetition—either by repeated association of stimuli or by trial-and-error—which is a strong argument against novelty in user interface design.

Fortunately, the learning curve can sometimes be compressed. The so-called sauce-béarnaise syndrome, which led to the modern theory of learning preparedness, is one well-established example (Seligman and Haber, 1972). The phenomenon is named for the strongly conditioned aversion to sauce béarnaise a psychologist developed after

one particularly unpleasant learning experience. In this form of learning, a single experience of a particular type (a novel taste followed by delayed aversive stimuli, such as nausea or vomiting) establishes an enduring aversion. While one would be hard put to find a way to make use of such an effect in user interface design, we have noted another form of single-trial learning that has proved of broad application in user interface design.

Anticipatory learning, or the "I-knew-it!" effect is a learning paradigm based on four elements: novelty, anticipation, action, and confirmation. In this learning model, the user encounters a novel feature or unfamiliar element on a user interface and hazards a guess about what it is or does or how it works. The user then tries it out and immediately is rewarded by discovering the guess was correct. Under such circumstances, many users will give a vigorous nod, make a triumphant fist-gesture, or exclaim "Yes!" or "I knew it!"

With this combination, our experience is that no repeated reinforcement or prolonged period of trial-and-error is needed. One use, and the user "gets it," thereafter correctly using whatever it is. From our observations, all of the elements of the pattern must be present: the user must recognize something as novel or unexpected, must guess at or anticipate some outcome, must take some action, and must receive strong and immediate confirmation through the anticipated outcome.

How does the designer exploit this paradigm? The nature of the user interface must be such as to encourage exploration and invite the user to make guesses about its functioning. These guesses must invariably be right or the user will not be taught how to use the new features but to avoid them. This means that the visual and interaction design must provide cues that subtly but infallibly guide users toward correct conclusions. The design must be such that it behaves as the user expects it to behave even if it incorporates features that the user has never seen before.

Principles of Instructive Interaction

Although instructive interaction employs many diverse techniques, all are informed by the same three broad principles that support rapid self-directed learning: explorability, predictability, and intrinsic guidance. These principles will be explained in this section and illustrated in the following section.

Explorability

Explorable user interfaces invite, by their appearance as well as behavior, user exploration, experimentation, and discovery. If users are to be able to teach themselves, the user interface must consistently allow and enable them to experiment and explore with little or no risk. This requires programming discipline as well as good design. The system must not penalize users for accidental or unintentional actions or punish them for mistakes. Most importantly, there must be no penalty for merely looking, or else users will quickly learn to avoid novel features or unfamiliar paths.

Explorability is the result of many small details, but three matters under the control of designers and developers contribute most strongly to an overall atmosphere of explorability.

- visible navigation
- consistent complete cancellation

- multi-level undo and redo

Visible navigation is a key to encouraging and supporting exploration of the user interface. Users explore and operate software more confidently when they can readily see where they are within the user interface and how they got there. Cooper (1995) has noted that many people use drop-down menus to explore what a new piece of software has to offer and how it is organized. Drop-down menus illustrate two aspects of explorability. First, the user can be certain that there is no penalty for just looking, since no action is taken until a selection is actually made. Second, the navigation scheme is visible; the user can see all available options along with the path through which these have been reached. Compare this to nested dialogs, where each new dialog box carrying the next level of possible actions hides all or part of the context that led to it. Even worse are full-screen designs that bounce the user from screen to screen or page to page with all trace of the trail covered over.

Explorability is more about the behavior of the user interface than its appearance. If users are confident that any action can be canceled or interrupted, they are more likely to try new features and thus learn how to use them. For this reason, all dialogs must include a cancel button which fully cancels any user actions. Its use at any time should be equivalent to never having opened the dialog box in the first place. This programming and design discipline allows users to freely play around with dialog boxes and the facilities they supply.

Facilities to undo and redo actions are at the core of explorability. These facilities must be ubiquitous and absolutely consistent. Users who encounter even a single isolated case in which the Edit | Undo option is mysteriously disabled will become more cautious and hesitant to try new things. A single level of undo is insufficient to fully support free exploration. In principle, infinite-level undo is needed, but in practice, a dozen or so levels is as good as infinite because users are seldom able to make effective use of more levels.

Predictability

To make possible the ideal of single-trial learning, user interface features must make their meaning and behavior obvious to the user. Where features are novel or take unfamiliar form, they must promote accurate anticipation and guessing by the user. It is tempting to employ the overworked malapropism “intuitive.” However, strictly speaking, a person can be intuitive, but a user interface that can easily be figured out by new users is better described as intuitable (Raskin, 1994).

An intuitable user interface is one for which the user can intuit the correct meaning and actual behavior of its features and facilities. In other words, the user can safely leap to conclusions based on first impressions without extensive thought or elaborate chains of reasoning. To promote single-trial learning, these spontaneous guesses and immediate impulses must be almost invariably right. In order for this to hold, familiar things must behave as expected, and novel or unfamiliar things must behave in ways that are reasonable and immediately understandable.

Consistency also promotes predictability. Consistency is both widely recognized as important in user interface design and widely misunderstood. While consistency with the general conventions of a particular software platform, such as Windows or the Macintosh OS, are important, internal consistency, within a given application, is even

more critical. A well-designed but non-standard interface that is internally consistent can still be easy to learn and to use.

Consistency is more than just a matter of using the same style of icons on all tool bars. Consistent behavior, organization, and appearance are all important. For ease of learning and use in innovative user interfaces, we have found that consistent behavior is more important than consistent appearance, and predictability is far more important than mere consistency.

When it comes to promoting learning and mastery, outright contradictions between different parts of a user interface or between expectations and performance are the greatest impediments. A design becomes dramatically more difficult if a control does one thing in one context and something completely different in another or if its behavior runs counter to what the user expects from its appearance.

Guidance

Although help is a critical factor in usability, most help facilities are distinct and separate components, explicitly summoned for assistance, and supplied in their own separate and often intrusive contexts apart from the context within which the user works. Guidance, by contrast, is an integral and inseparable aspect of the user interface itself. Rather than being stored in a separate place, it is ubiquitous and always available in context. Rather than having to be summoned, it is provided as needed and appropriate without requiring any special action or initiative on the part of the user.

Guidance may be thought of as a specialized form of help, namely, intrinsic help. Intrinsic help is unobtrusively built into the user interface. In the next section, examples will be presented of how—by offering direction rather than assistance and by subtly channeling the user's attention and actions—the appearance, organization, and behavior of the user interface itself can combine to guide the user in how to use the system.

The objective of guidance is also to provide informative feedback that is intrinsic to the appearance and behavior of objects and in the immediate context within which users interact with the system. For example, instead of a modal message box that reports that an action is complete, the user should be able simply to see the results.

Techniques

The techniques of instructive interaction are an open-ended and somewhat loose collection of tricks and schemes informed by the common set of principles just outlined. Table 1 summarizes the techniques to be described here.

Table 1 – Instructive Interaction Techniques

Technique	Description
starters	messages that indicate first steps or suggest initial user actions
balloon tips or help	pop-up messages that appear on first encounter or when a user reaches a certain point in a process
screen tips	brief pop-up messages that appear after a delay when the mouse pointer remains over a tool or other object
progressive tool tips	two-level tool tips with an extended message appearing after a second delay; also called "cascading tool tips"
embedded prompts	hints, directions, or explanations embedded within the fields of user interface controls, such as text entry boxes, drop downs, and combo boxes
input prototypes	representative examples of acceptable user input, such as, "Time of day (e.g., 13:30):"
templates	models of the format of acceptable user input, such as, "Expiration date (mm/yyyy):"
start highlighting	graphic or other indication of the place to start within a dialog box or screen
workflow highlighting	a line or other graphic device that visually guides the user through the steps of a process
dynamic affordances and constraints	dynamic changes in appearance to signal allowed and disallowed actions
instructive animation	graphical animation that informs users about what is happening or how to use or interpret something
anticipatory action	automatic actions supplied by the user interface in anticipation of user needs
progressive enabling	the technique of enabling a series of user interface controls in their logical or required sequence
progressive disclosure	the technique of exposing or displaying user interface controls in their logical or required sequence
implicit antecedents	automatic insertion of missing user steps or actions when these are unambiguously implied
idiomatic parallels	using common arrangements, appearance, or hints across different interaction idioms or methods for accomplishing the same task
thematic variation	use of familiar or standard components and interaction idioms in new or non-standard contexts

Intrinsic help

Although intrinsic help is built into the user interface and automatically available without explicit user action, the guidance it supplies can be either implicit or explicit and may take either active or passive form. Examples of explicit guidance, shown in the mock-up of Figure 1, include "starters," that is, hints or messages on getting started, embedded prompts, examples and templates, and screen tips (or alt-text in Web applications), as well as the useful first-time-only variant on tool tips provided by "balloon tips" in Windows XP.

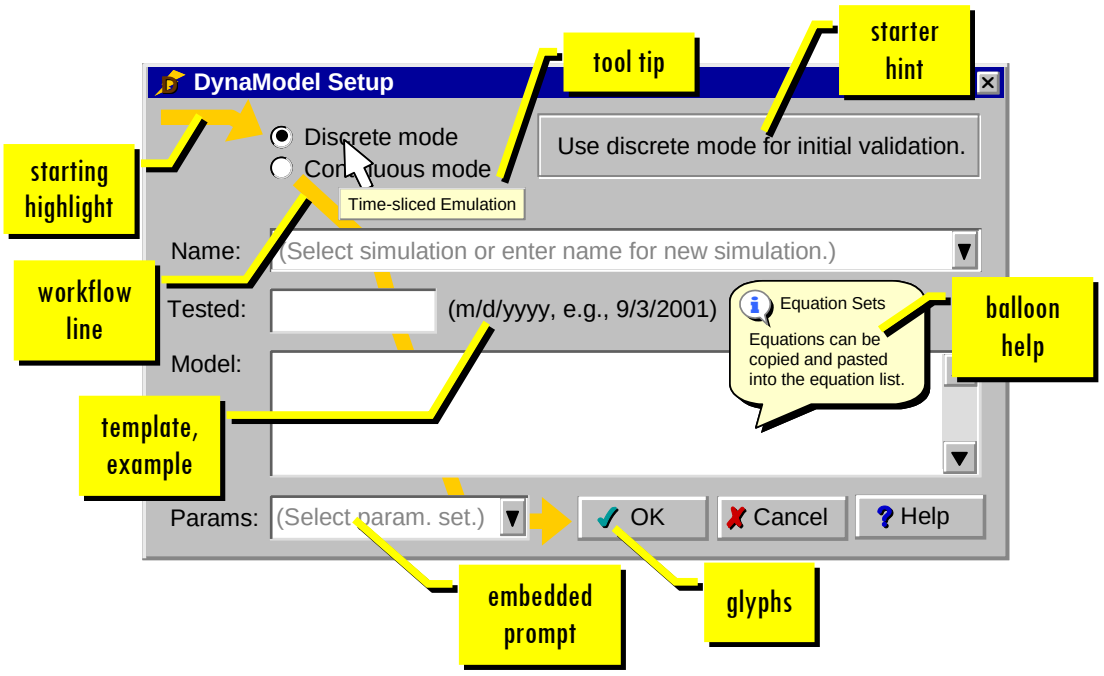


Figure 1 – Examples of intrinsic help.

Embedded prompts are a particularly versatile tool for making interfaces self-teaching. Because an embedded prompt appears inside a control, such as a text box or drop-down selection list, it does not take up extra screen space. Unlike a tool tip, it is available without delay. Typically, embedded prompts are rendered in an alternate text style—such as grayed, italicized, or parenthesized—and vanish the instant the user uses the component. Thus they are accessible to the puzzled beginner but are easily ignored by the confident expert.

Implicit guidance also takes many forms. The preferred or common starting point in a dialog might be highlighted to draw the user's attention, or a colored line might indicate a typical workflow or task sequence. Semantically or operationally related controls can be highlighted with common colors, shapes, or glyphs. The Borland-style glyphs on **OK**, **Cancel**, and **Help** buttons are an example that makes it easier for users to quickly locate and use the primary navigation controls in dialog boxes.

Dynamic affordances and constraints

The twin concepts of affordance and constraint (Norman, 1988) are versatile means for guiding and instructing users. Affordances communicate to the user an invitation to particular actions or uses; constraints limit or restrict these. A ubiquitous example in

graphical user interfaces is “push affordance,” the way in which a visually protruding rectangle on a screen invites a user to “push” it by clicking on it.

Dynamic affordances and constraints are dynamic changes in appearance that invite and channel user actions. Dynamic affordances and constraints were used with great success in the award-winning design for the Siemens STEP 7 Lite integrated development environment for automation programming (Constantine, 2002; Windl and Constantine, 2001). For example, in Figure 2, dynamic affordances and constraints guide PLC programmers in the correct configuration of hardware for complex automation applications. As the mouse pointer drags a particular hardware module toward the graphically represented equipment racks, slots that can accept that particular type of module are highlighted and ones that cannot are overlaid with the international symbol for prohibited action.

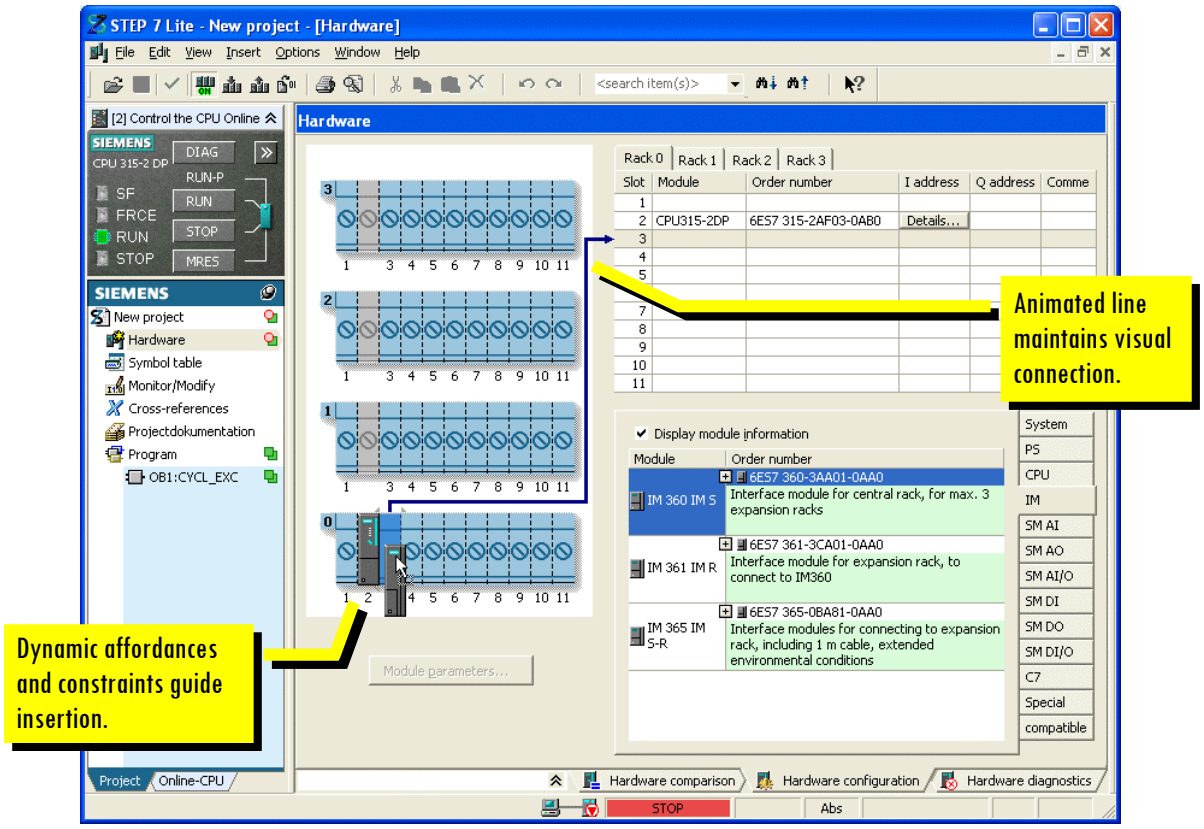


Figure 2 – Dynamic affordances and constraints illustrated

Instructive animation

Animation can show users what is happening and can aid in interpreting a complex process or visual presentation. For example, by following the user’s mouse movements and selections, an animated colored line visually links the two synchronized views shown in Figure 2. What might otherwise have been a visually confusing relationship between graphic and tabular information in different spatial arrangements was made instantly understandable.

Progressive tool tips

Tool tips (or screen tips, as they are coming to be called) are widely used in modern graphical user interfaces, not only to identify tools but also to supply information on other visible elements, including data fields. To overcome some of the limitations of conventional screen tips, we created progressive (or cascading) screen tips (Constantine and Lockwood, 1999: 242), which provide an expanded secondary text after an added delay.

The availability of additional detail and the full capability of progressive screen tips are communicated to the user by a combination of passive presentation, affordance, and instructive animation. Progressive tool tips are marked by a distinctive graphic device in the lower right corner of the initial text (Figure 3), which users quickly learn to associate by experience with the availability of additional information.

Simply by moving the mouse pointer into or clicking within the primary text area the user can force display of the secondary text at any time without waiting. This capability is communicated to the user by giving the primary text "push affordance" and by animation that shows it depressing and releasing when the secondary text is automatically displayed. In the example of Figure 3, the secondary tip also includes a hyperlink into the regular help files, thus providing a smoothly integrated help facility that is highly adaptive to user needs.

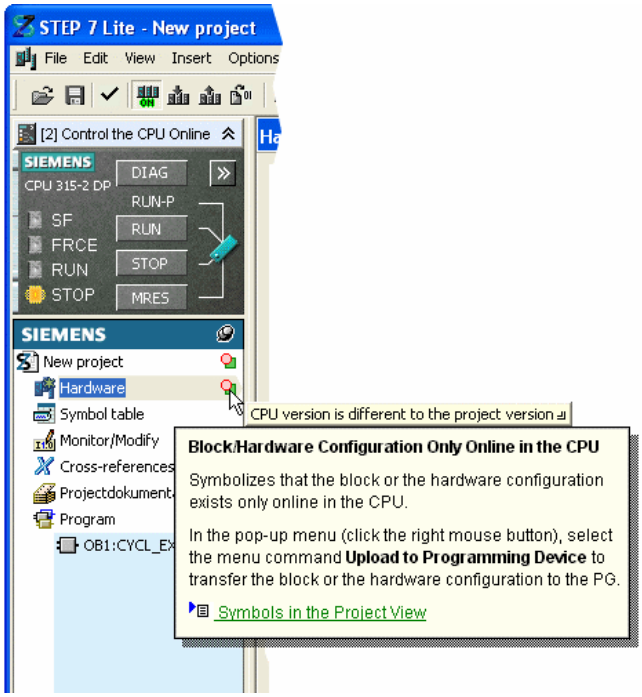


Figure 3 – Progressive tool tip illustrated.

Anticipatory action

A well-designed user interface can sometimes anticipate probable, necessary, or desirable steps by a user, not only saving steps but also instructing the user by exposing options and implicitly suggesting appropriate actions. For example, the dialog box in Figure 4 begins with the **School** drop-down already open, and the next drop-down opens as soon as a school is selected, thus saving user steps and cueing the user on successive actions.

We used this technique with marked success in a system designed for lesson planning by classroom teachers (Constantine, 2001; Constantine and Lockwood, 2002). A novel control for configuring and navigating within complex documents opened automatically whenever a lesson plan was opened, thus ensuring that new users discover its capabilities while also making it available when needed.

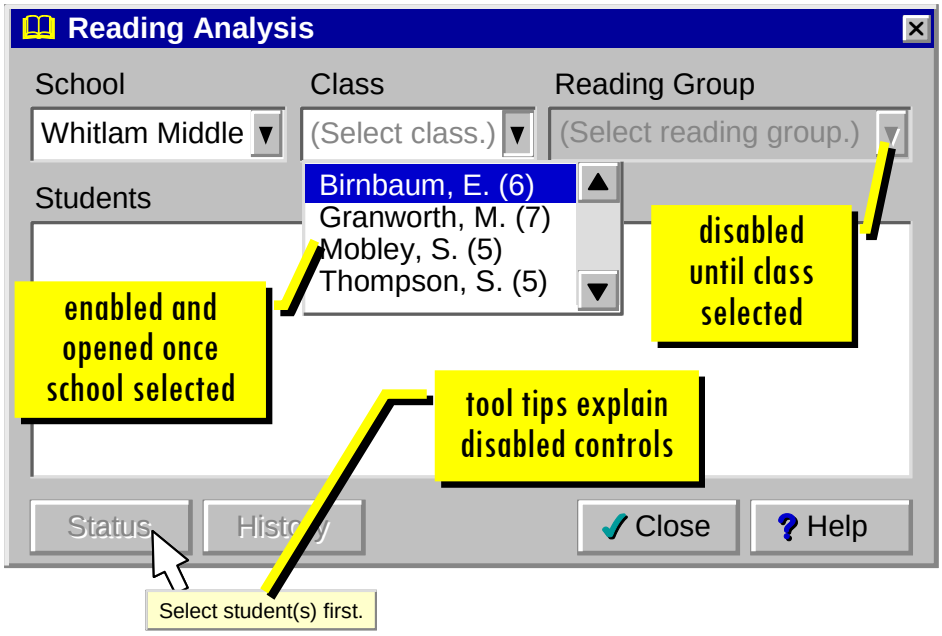


Figure 4 – Anticipatory action and progressive enabling illustrated.

Progressive enabling and disclosure

Progressive enabling and disclosure are techniques for unobtrusively walking users through a series of actions or steps. In the example of Figure 4, only the **`School`** drop-down is initially enabled. When a school is selected, the **`Student list`** becomes populated, and the **`Class`** drop-down is enabled and opened. The command buttons for **`Status`** and **`History`** are enabled only after one or more students is selected.

This approach, especially combined with anticipatory action as illustrated here, is self-teaching and can prevent or reduce user errors. Users will not waste time trying to select a reading group unless a class has been selected and will be spared getting a modal message box with "Error: No student is selected" from using the **`History`** button out of turn. Note, however, that even disabled controls should present screen tips. As suggested in Figure 4, these should explain the reason the control is disabled.

In progressive disclosure, controls, fields, or panels do not appear until they can logically be used. Because controls that suddenly appear (or disappear) from the user interface are visually and cognitively disruptive to the user, progressive disclosure should be used judiciously. It is most defensible where, as in Figure 5, the display differs radically depending on the user selection or prior action. Progressive disclosure avoids cluttering the visual field with controls and fields that are not of interest without taking the user out of the current context. Note that the display panel of Figure 5 itself appears only after a status or history report has been selected.

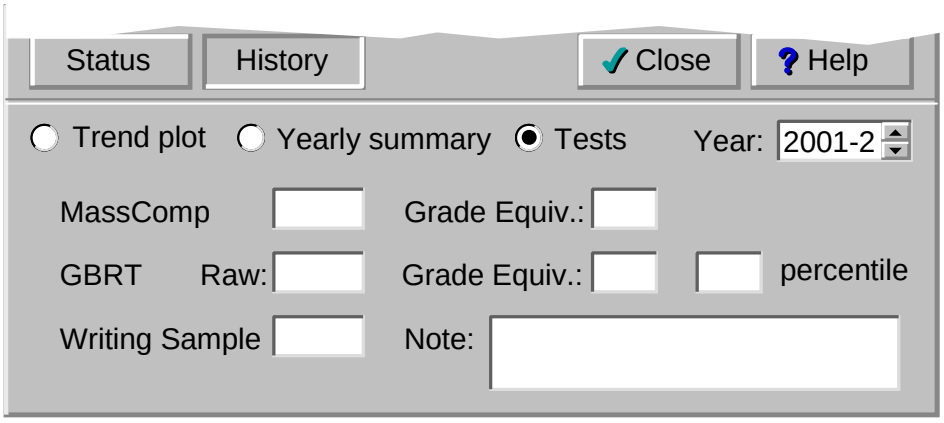


Figure 5 – Progressive disclosure: form selection.

These two techniques can often provide an in-context alternative to wizards. Although popular, wizards have a number of disadvantages. They disrupt tasks by taking the user out of the working context, inhibit skill building by hiding actual operations from the user, and are often experienced as plodding, inefficient, and unreasonably rigid. By keeping the user within the working context, progressive enabling and disclosure more effectively promote user familiarity and skill with available features.

Implicit antecedents

Progressive enabling and disclosure are most appropriate where there is a required sequence that is both necessary and logical to the user. All too often, though, user interfaces impose a fixed and artificial order that may make sense to computers and programmers but not necessarily to users. In some cases, this program-oriented logic is built into the prescribed behavior of basic user interface components. For example, if a user clicks on a tool button in one child window when another child window is active, the expected action does not occur. Instead, all that happens is to shift the focus, making the new window active and bringing it to the visual foreground. The user must click again to trigger the desired action. Although the behavior is standard and completely logical from a programming perspective, even highly experienced users can be caught unaware by this need to double-click at some times but not others.

The intention of the user in virtually all such cases is clear: to trigger the selected action. The shift of focus from one child window to the other is implied—an implicit antecedent.

Implicit antecedents avoid imposing an artificial order or overly rigid logic by automatically supplying missing steps to make interaction both more efficient and more natural. Consider the task of printing out slides 14-24 using the portion of a print dialog shown in Figure 6. By a rigorously correct but irritating logic, the user must first select the radio button for **Slide** before the text entry box becomes enabled to allow slide numbers to be typed. However, an attempt by the user to type numbers into the text box unambiguously implies selection of the radio button as an antecedent—an implicit antecedent.

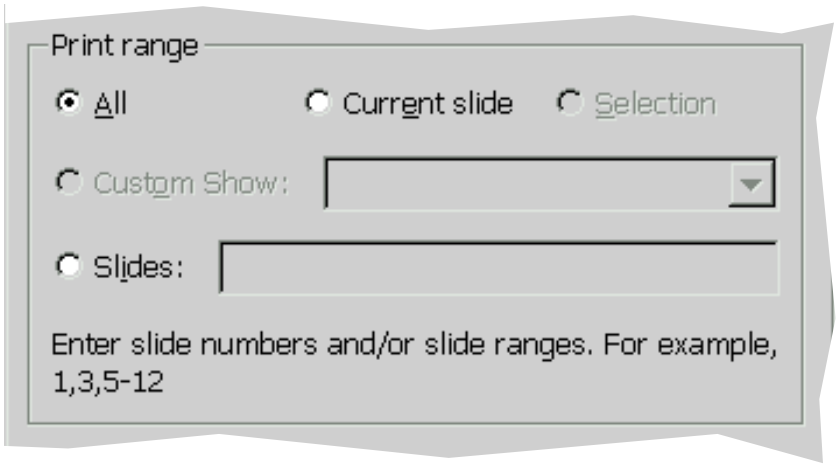
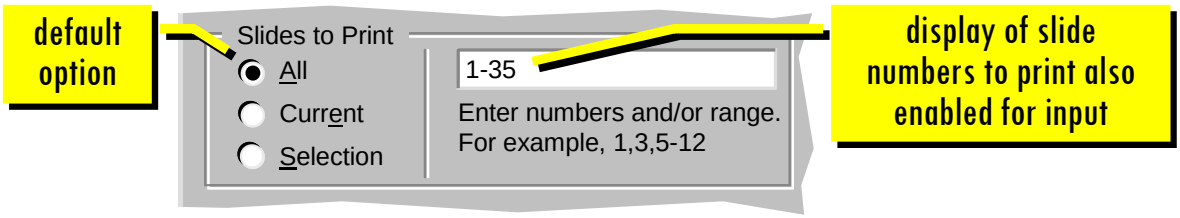
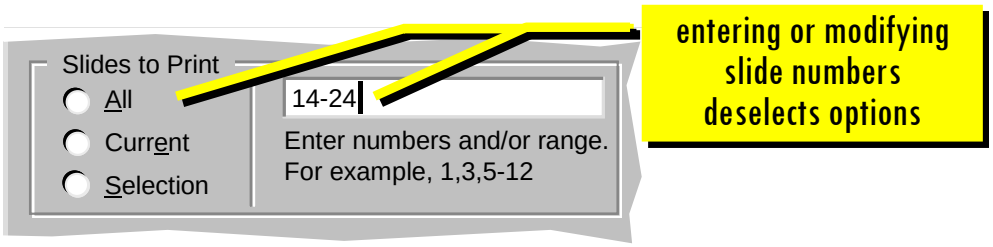


Figure 6 – Rigidly enforced "programming" logic (PowerPoint 2000).

In our experience, such cases of an imposed logic often expose opportunities for more radical redesign that yield marked improvements in usability. In Figure 7, the text box is always enabled, inviting the user to type in or to edit its contents. As shown in Figure 7(a), it has been overloaded to provide feedback in context by displaying the actual numbers of the slides represented by the selections **All**, **Current**, and **Selection.** This overcomes another usability problem in the original design, namely that the modal dialog box covers up part of the visual context so the user may not be able to see which slide is current or which slides have been selected.



(a)



(b)

Figure 7 – Flexible logic with implicit antecedents.

As shown in Figure 7(b), typing anything into the text box deselects all of the standard options, any one of which could again be selected. This behavior violates the rigid rules for radio buttons but is completely logical to users. Although some users (and most

user interface “experts”) remark on such a departure from standards when they first encounter it, when the departure reflects the intrinsic structure of the task—as it does here—nearly everyone figures it out immediately and uses it without further difficulty.

Parallel interaction idioms

Flexibility can contribute to ease of learning by allowing the user to accomplish tasks according to their own spontaneous impulses and idiosyncratic styles of thought and work. Good user interfaces accommodate various working styles by supporting alternative methods of operation or interaction idioms (Cooper, 1995), such as, point-and click use of tools or keyboard access to menus.

Alternate interaction idioms can be linked by using consistent arrangements or visual patterns. Such linkage aids learning and promotes progressive acquisition of advanced skills. In the screen-shot fragments of Figure 8, showing a customized version of Microsoft’s PowerPoint, buttons on the tool bar are arranged in the same order as the sub-menu entries, glyphs on the menu items mirror the icons on the tool bar, and tool tips cue the user regarding keyboard access.

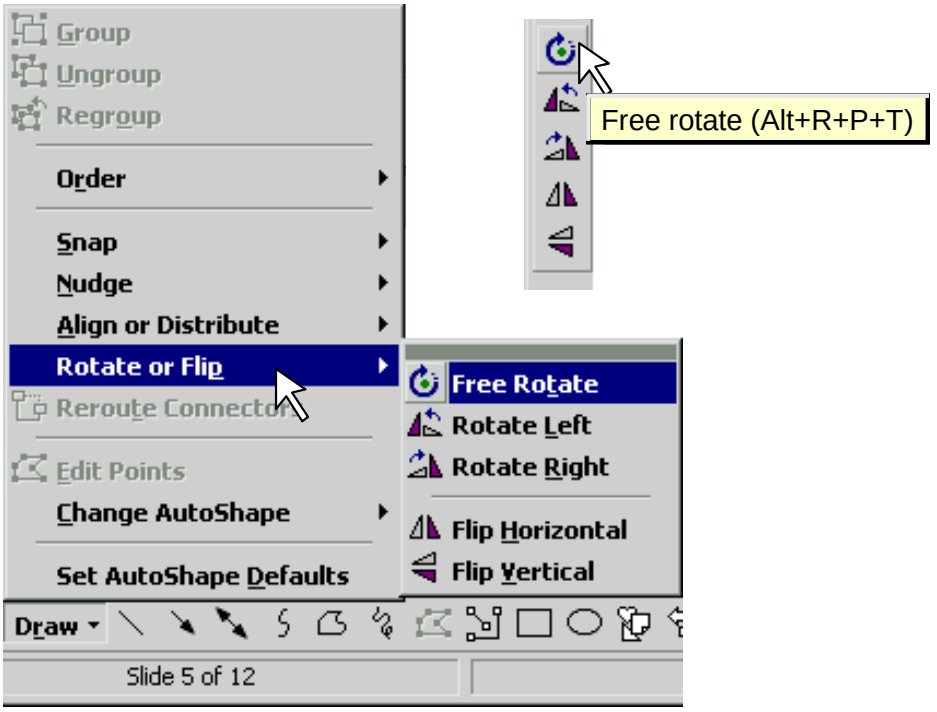


Figure 8 – Idiomatic parallels illustrated (PowerPoint).

Theme and variation

In many cases, the best solution to a given design problem does not use standard components or behaviors but some task-oriented variation. Many standard, well-established controls, visual elements, and interaction idioms can be pressed into service in novel ways, provided that the new functions are consistent and logical extensions of the standards.

For example, many users find the standard spin box commonly used to set clock time (see Figure 9) to be clumsy and inconvenient. (Actually, the standard time-setting spin box only looks conventional; its behavior—spinning the hours up and down after

clicking within them—is bizarrely non-standard.) Twin spin boxes offer more straightforward and efficient operation, but the presentation is visually fragmented and no longer looks like a time value. The dual-spin box at the bottom of Figure 9 is a non-standard but natural choice. Although programmers and psychologists alike have protested that such a design would be confusing, in fact, users instantly figure out its use and behavior, which is confirmed and reinforced when they try it. Such design by theme and variation promotes single trial learning by appealing to the familiar without violating expectations.

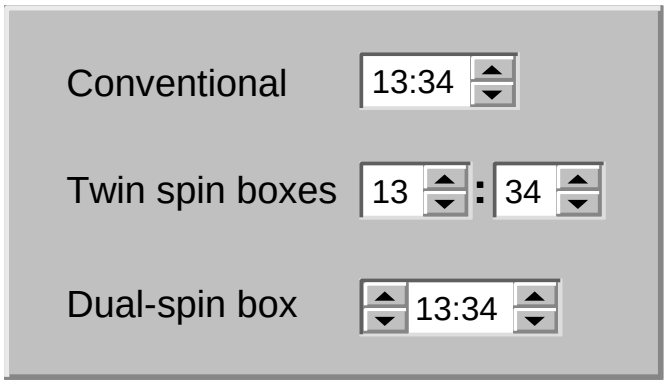


Figure 9 – Variations on a time-setting theme.

Toward Familiar Novelty

When consistency and conventions are elevated to the status of immutable imperatives, usability is sacrificed and users suffer. Significant improvements in ease and efficiency of use often require creative departures from standards and accepted practice. However, useful innovations in visual and interaction design should not burden the new user with a long and frustrating learning process.

Instructive interaction, based on anticipatory learning and the principles of explorability, predictability, and guidance, can make it possible for new users to quickly learn the meaning and operation of novel features and facilities, in many cases after only a single trial.

The specific tricks and techniques presented in this article are merely intended to illustrate the range of possibilities for instructive interaction. Creative designers will no doubt continue to add to the toolkit of instructive interaction techniques by inventing new visual designs and interaction techniques that succeed because they fit the tasks and intentions of users.

References

Constantine, L. L. (1991) “Toward Usable Interfaces: Bringing Users and User Perspectives into Design,” *American Programmer 4* (2).

Constantine, L. L. (2001) Design studies 1-3. http://foruse.com/Resources.htm#Articles

Constantine, L. L. (2002) Design profile in *interactions 9* (2), March/April.

Constantine, L. L., and Lockwood, L. A. D. (1999) *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design.* Reading, MA: Addison-Wesley.

Constantine, L. L., and Lockwood, L. A. D. (2002) "Usage-Centered Engineering for Web Applications," *IEEE Software, 19* (2): 42-50, March/April.

Cooper, A. (1995) *About Face: The Essentials of User Interface Design.* Foster City, CA: IDG Books.

Norman, D. O. (1988) *The Design of Everyday Things.* New York: Basic Books.

Seligman, M. E. P. and Hager, J. L. (1972). "Biological boundaries of learning. The sauce-bearnaise syndrome." *Psychology Today, 6*: 59-61, 84-87.

Windl, H., and Constantine, L. L. (2001) "Performance-Centered Design Platinum Award of Excellence: STEP 7 Lite Hardware Configuration." http://foruse.com/pcd/

Learn more about usage-centered design practice and principles at http://www.forUse.com.