NeXTstep Measurement Kit:

An Empirical Basis for User Interface Research

Alex Meyer
ameyer@phoenix.Princeton.EDU

4 May 1992

Submitted to the
Department of Computer Science
in partial fulfillment of the requirements for the degree of
Bachelor of Arts

Dedications

To my parents, for fueling me, financing me, and letting me steer.

To my grandparents, without whom none of this would have happened.

To all of my teachers from whom I learned.

To Heather, for emotional support and late-night food.

Abstract

The human-computer interaction literature relies almost entirely on subjective evaluations and contrived psychology style studies. Such methods are valuable, but fail to address typical users in typical situations. This paper presents a method for gathering objective data about the human-computer dialog during actual use. It is hoped that discussions about user interfaces

can, in the future, rely on broad-based empirical data. A package is described that adds data-gathering capabilities to the NeXTstep graphical user interface. This package is remarkably easy to add to existing software. Recent ideas in human-computer interaction are discussed, and an argument is made in favor of user-interface toolkits.

1 Introduction (the case for empirical data)

Computers continue to proliferate in the civilized world. More tasks are being computerized, and more important tasks are being entrusted to computers than ever before. Lay people's day-to-day interactions with computers are

increasing steeply. The computer industry continues to grow in size and importance. Now, even more, human-computer interaction commands attention.

Due to both market pressure and academic interest, user interfaces have become increasingly discussed. Computer manufacturers feel pressured to deliver products to which consumers will react positively. Hardware and software producers react to potential buyers who are looking for productivity-boosting systems. Furthermore, there is genuine concern about how best to reduce the effects of users' inevitable errors. Now that so many computer-controlled operations have non-trivial consequences, government agencies and others are funding research into easy-to-learn, foolproof systems.

This scenario has resulted in hundreds of books, journal articles, and anthologies. The literature abounds with everything from theoretical manifestos to narrative accounts. Researchers draw on cognitive science, gestalt psychology, graphic design, experimental psychology, and other fields in their pursuit of better ideas for the interface between the human being and the automaton. The astute reader notices, however, that the bulk of the literature relies on subjective evaluations and qualitative analyses.

This lack of scientific basis stems from a tradition of non-objective criticism in the field. Originally, the dialog contained only computer people. Generally, the user interface was the last part of a computer program to be written. Programmers did their best and offered suggestions to each other, but the

results tended to be dismal. This was before the Apple Macintosh (which included the opinions of non-programmers earlier in the design process). It has been said that The Macintosh has the first interface good enough to be criticized [NORM90, p. 209].

The Macintosh interface turned out to be an overnight sensation and a Pandora's box. This computer progressed into uncharted territory. Still, such territory remains quite difficult to chart: In numerous experiments comparing two alternative designs, subjects state a strong preference for one design but indeed perform faster with the other [FOLE90, p. 392]. The literature, not surprisingly, contained results that seemed contradictory, and often did not generalize well.

One of Apple's manuals states: The primary test of the user interface is its success with users [APPL87, p. 15]. Such a nebulous statement serves as a poor basis for true progress (although it was a giant step at the time). Even in 1990, interface design often involved more ESP than human factors. The following passage comes as advice from a recent book on user interfaces:

> interface designers can simulate potential users. Designers can go through the exercise of pretending to be a user and step through what they perceive to be typical user scenarios. [VERT90, p. 53]

Some, however, have realized that simulation and pretending are of limited

use. Drawing on the discipline of experimental psychology, they observe users in special environments as they perform prescribed tasks. They admit, however, that:

> This isn't the only way to observe users; in fact, it's one of the least scientific ways. But if you try this technique, you'll get lots of useful data for designing and revising your interface. [GOMO90, p. 86]

The more scientific ways have proved elusive. Attempts have been less than ideal due to small sample sizes, artificial environments, artificial tasks, and poor measurement techniques.

This paper presents a package, the NeXTstep Measurement Kit (NMK) which should help remedy the situation outlined above. It fills a need, but is not a panacea. NMK can provide researchers with empirical data about users' low-level actions. Such measurements cannot stand alone; they should complement subjective analysis and qualitative evaluation. High-level results that cannot explain low-level data must be discarded; high-level results that confirm low-level data will be that much stronger. The availability of such measurements represents a fundamental shift in the way user interface research can be done.

First, previous results by this author, upon which NMK relies, will be discussed. Next, this paper evaluates recent ideas in human-computer

interaction. The choice of the NeXT platform is then discussed. A discussion of the NeXT development environment is presented, followed by a description of NMK. Finally, conclusions are offered.

2 Previous work

This paper takes, as its starting-off point, earlier research by this author. Such work has dealt with high-level issues informing the field of human-computer interaction. This previous work has resulted in some useful results, described below. Since then, the literature has grown. Some newer results are addressed below as well.

2.1 Previous Results

A computer with a graphics screen and a mouse can support innumerable human-computer interaction styles. The screen becomes the point of intersection between human and machine. Users cannot see the internal workings of computer systems. The only reality for the user is that presented on the computer screen [NORM88, p. 177]. For the user, the computer creates a mimetic world [LAUR86, p. 74]. The screen is the stage upon which both computer and user seem to act. On this stage, the actors constitute the graphical user interface (GUI).

Donald Norman, a cognitive scientist, has explored the process of tool

design from the standpoint of the user. Instead of memorizing all the aspects of a device, he explains, people form conceptual models in their minds. These models arise from subjective experience and allow people to use reasoning in order to determine how to utilize a device. Different people form different models, and most importantly, the designer's model often deviates sharply from users' models. A device has a system image with which users interact and from which they form their mental models. The system image originates from the design model, but is not synonymous with it. The key to the process, then, is for the designer to produce a system image that will result in the user forming an accurate conceptual model.

A good device should increase productivity. One method is to minimize the

ill effects of users' errors by preventing errors or by providing recovery. This motivation has resulted in specific guidelines for direct-manipulation GUI design:

*Visibility*Đeverything that the user might need to know should be represented in the display.
*Mappings*Đthe associations between graphical objects and their functions should be complete and clear.
*Ratio of controls to functions*Đeach interface item should provide as few functions as possible in order to avoid confusion.
*Modes*Đonly when necessary and clear should modes be used.

*Feedback*Đthe illusion of cause and effect should be maintained in order to empower the user.
*Errors*Đunintended user actions should be recognizable as such, and consequences should be minimal.
*Exploratory learning*Đusers should be able to familiarize themselves with a system via safe exploration.
*Experts*Đfeatures which help novices should not punish experts.
*Task orientation*Đthe task at hand, and not the computer system, should occupy the user's mind.

These guidelines represent high-level progress, but provide little concrete help

to the GUI designer.

In order to arrive at more detailed results, more specific data is required. Ideally, researchers would study users in actual-use situations, free of the artificiality of psychology studies. Ben Shneiderman, a human factors researcher, has called for monitoring of the user-computer dialog. By examining this dialog (along with other factors), scientists might be able to characterize the user's model of the system.

Computers lend themselves well to monitoring their own usage. Such an automated process should be simple to implement and virtually error-free. The proposed self-measuring GUI is shown conceptually below (Figure 1).

SMGUI.eps ¬
Figure 1: The Self-Measuring GUI, after [NORM88, p. 16]

A designer conceives of a design model, which eventually becomes a system. The system is mostly invisible, but its interface is the system image it presents to the world. By interacting with the system image, a user forms a personal user's model. It is assumed that the only channel of communication between computer and user is the GUI. Thus, by adding the proper recording capabilities to the GUI, it will be possible to analyze the factors which determine the user's model.

2.2 Newer ideas

In recent literature, researchers are beginning to consider the interface to be more of a problem than a panacea. Donald Norman clearly explains this idea.

> The interface is the wrong place to begin. It implies you already have done all the rest and now you want to patch it up to make it pretty for the user. That attitude is what is wrong with the interface. [RHEI90, p. 6]

> The real problem with the interface is that it is an interface. Interfaces get in the way. I don't want to focus my energies on an

interface. I want to focus on the job. [NORM90, p. 210]

Well-designed software should be human-friendly from the ground up. Thus, there exists no interface, just program.

According to this view, those who study computer-user interfaces have missed the point. Human-factors issues should be considered right from the beginning of the design process, before a single line of code has been written. This is known as user-centered system design. Brenda Laurel, a human-machine interaction consultant, poses the following scenario.

An interesting possibility is that the discipline of human-computer

> interface design will disappear....Perhaps in the future we will finally give up the illusion of applications engineering and interface design as being two separate things. [LAUR90, p. xiii]

Although an oversimplification, the above statement echoes the growing sentiment that in order for all facets of a product to be human-usable, humans must be considered in all facets of the design.

Surely, this is a positive sentiment. The issue, however, is complex. A considerable amount of the recent literature seems to present user interfaces and user-centered system design as diametric opposites. In supporting their claims, researchers tend to compare badly designed straw man interfaces with

success-stories of user-centered system design. These strategies, however, are not mutually exclusive. Although each method alone may, if perfectly executed, result in an excellent end product, designers utilize both methods.

Theoretically, both strategies involve similar ideas. The major difference involves how far along in each process these ideas come into play. This paper explores the implications of these methods to the area of user-interface toolkits. User interfaces require only that a programmer install the appropriate elements from the toolkit in order to bridge the gap between user and program. At the other extreme, user-centered system design involves numerous interviews with potential users, the development of storyboards, mock-ups, and prototypes. The human-computer interaction paradigms are considered at the beginning

and the program grows up around them. What may appear to be a user interface item actually constitutes a custom-made interactor.

This second method has a few problems. First, it takes more work. This is not a valid criticism, of course, except when considered along with the other criticisms. Second, although this method may result in a product perfectly suited to its users, its entire design rests upon the initial specifications. Modifications and additions (the scope of which always seems to exceed predictions), require an almost complete overhaul and risk destroying the preexisting delicate balance. Third, code written in such a way may not lend itself well to reuse.

This paper takes the position that, while considering humans early in the

design process helps, good user-interface toolkits help more. All but the most trivial tasks require more descriptive channels of communication than are available in a GUI (or even a command line interface). Many mechanisms have been developed in attempts to compensate, but the problem remains. Some mappings of actions to functions will, necessarily, seem arbitrary. Good designers know, however, that arbitrary mappings become more palatable if standardized. Computer users can learn consistent conventions.

> in the future, the level of computer sophistication of workers will increase, as more users enter the workforce already computer-literate through computer use at home and school. [FOLE90, p.

348]

The concept of literacy in the above passage works well. Interface idioms, like spoken languages, require a modicum of familiarity. No interface that will do anyone much good will be instantly understandable to a computer-illiterate.

Obviously, a conscientious practitioner of user-centered system design will use consistency whenever appropriate and might even take advantage of user-interface toolkits. Herein lies the essential power of toolkits: they are general.

When speaking of generality, one must consider the computer itself. Some consider the computer to be a tool. Drawing an analogy from a Turing machine, the computer is all tools. Thus, a human-computer interaction

scheme should potentially be all interfaces. One might argue that computers are not very useful in this raw potential state; that concrete, specific applications make computers useful, but one must consider what role computers should play. Computers can be tools, like shovels and rakes and implements of destruction, or they can be devices with the power to energize and transform. One need only look at recent history to determine which definition people have chosen.

Toolkits, then, must go hand-in-hand with the new incarnations of computers. This calls for a separate field of user-interface research. People in this field will create interface techniques that advance the transformational magic expected of computers. The products of such research will, of course,

be general. This situation represents an improvement over scenarios in which programmers merely see the interface as a means of meeting an expectation. This paper makes the case that user-interface research and development can and should exist as an independent discipline. On the motivation of this conclusion, an interface toolkit is chosen, to which empirical data gathering capabilities are added. The chosen toolkit, the NeXTstep Application Kit, involves a user-interface management system (UIMS), Interface Builder. This provides an additional benefit over user-centered system design.

> UIMSs can increase programmer productivity (in one study, up to 50 percent of the code in interactive programs was user-interface

> code), speed up the development process, and facilitate iterative refinement of a user interface as experience is gained in its use. [FOLE90, p. 457]

The NeXTstep Measurement Kit adds to the virtues listed above.

3 Choice of platform

Given the task of adding statistic-gathering extensions to a GUI, the question arises, Which GUI? Many platforms and even more GUI's exist. Each has different pros and cons: size of installed base, ease of programming,

standardization, etc.

3.1 X Windows

The original plan had been to write a self-measuring GUI package for X Windows. This platform is non-proprietary and boasts a large, multi-architecture installed base. X Windows's unique features include the ability to access the pointer motion history leading up to a certain time.

The plan, as such, was deemed unworkable due to a number of factors. The X Windows environment resists efficient coding. The X Library presents innumerable options and special cases, without benefit of standards. This freedom turns to chaos, especially in light of X's convoluted approximation of

object-oriented programming. As a result, many toolkit packages have sprung up, each of which somewhat reduces the complexities of X Library programming. Currently, many layers of these toolkits exist. Even worse, not all X Windows systems support all documented features. It was decided that, since too many toolkits exist already, since writing a toolkit is so involved, and since the X programming environment is overgrown, other platforms should be explored.

3.2 IRIS

As a no-nonsense graphics workstation, the IRIS seemed promising. Building a self-measuring toolkit from the ground up would not pose any difficulty.

Getting programmers to use it, however, would. Most IRIS programmers write graphics code and don't care too much about the user interface. Very few applications of general interest exist on the IRIS, anyway. The built-in interface is simple, and appropriate for the machine. In any case, the IRIS runs X Windows for those who desire more complicated applications. The IRIS would not provide the best platform for a self-measuring package.

3.3 Macintosh

If the Mac truly has the first interface worth critiquing, it seems appropriate to add analysis capabilities to it. The Mac does have a single, consistent interface. Also, programming the Mac is generally straightforward. The Mac

interface, however, is proprietary property. Details about its internal workings cannot be obtained. Additionally, many layers of backward-compatibility issues mire coding projects. Without detailed documentation, writing the self-measuring GUI on the Mac seems unfeasible.

3.4 NeXT

The NeXT computer embodies the latest user-interface technology available on a wide scale. This platform offers a number of advantages over the others considered. The NeXTstep user interface embodies the single standard interaction paradigm for NeXT hardware. This interface takes the form of the NeXT Application Kit, an object-oriented library. Application Kit programs

utilize Objective-C which allows the incorporation of objects from the object hierarchy in an elegant fashion. NeXT's documentation includes detailed descriptions of the entire Application Kit. Thus, only the additional functionality for each user-interface element must be written. The Interface Builder allows custom objects to be included in applications with ease. This platform has proved quite suitable.

4 The NeXTstep development cycle

Modern interactive programs depend on an event loop or a variant thereof. The center of the program, the event loop, involves an almost endless cycle of

waiting for the user to do something and then reacting to it. The actual algorithms of the program assume the role of satellites (Figure 2).

OldStyle.eps ¬
Figure 2: A Standard Event Loop Program

Algorithms, and other code that encapsulates the specifics of the program, become active only when the event loop calls them. Thus, the important parts of a program are really the satellites. The rest of the code seems similar from application to application. NeXTstep provides this part already implemented.

4.1 The Application Kit

NeXT's toolbox relies strongly on its object-oriented language, Objective-C. This language, a superset of C, provides many of the features of C++ without some of the more complicated issues. It also allows dynamic linking of objects at runtime, a feature upon which Interface Builder and NMK depend. Objective-C supports inheritance (sadly, not multiple) and class equivalence.

The Application Kit provides an entire hierarchy of user-interface and support classes which can easily be used as-is or subclassed to provide special functions. The class named Application contains the event loop (Figure 3).

AppKit.eps ¬
Figure 3: A Standard NeXT Application Kit Program

Other objects, for instance of class Button (which visually contains a ButtonCell), act as satellites or handlers for the Application. These interface objects have, as their targets, objects of custom classes that implement the specifics of the application. Such a method meshes quite well with event-loop-style design.

4.2 Interface Builder

The Application object and its associated interface objects and algorithm

objects contain links to each other which form a complicated network. This network makes up, in rough manner, the user interface. Different applications have different networks, but the networks are usually the same for each invocation of the same program. For this reason, NeXT provides a mechanism for archiving and restoring all the objects and connections in such a network. These so-called nib files encapsulate the interface (or components of it) and allow the interface to be developed separately from other parts of the program.

The Interface Builder, in its simplest form, represents a tool for the creation and maintenance of nib files. One adds standard NeXTstep interface items such as menus, windows, buttons, and text-boxes to a program by click-and-

drag techniques. A special Inspector window allows the modification of object attributes. Interface Builder provides a simple, graphical way to make all the connections between objects (even those which are invisible). Programmers can even develop palettes of custom objects and load them into Interface Builder.

All the files needed for an application become part of a project, which Interface Builder manages. It allows developers to specify the connections between all aspects of the application. Further, Interface Builder allows the programmer to specify the external interface for a new class, and Interface Builder will generate the program stubs. It also generates a Makefile for compiling and building the application.

For obvious reasons, most NeXT programmers choose to use Interface Builder. In fact, the NeXT documentation practically discourages the development of software without Interface Builder. Thus, Interface Builder stands poised as the central hub of the development cycle. NeXT program development generally starts and ends with Interface Builder. Thus, any attempts to add self-measurement to NeXTstep must embrace the Interface Builder development process.

5 The NeXTstep Measurement Kit

NMK offers programmers a package which allows them to collect statistics

about how users interact with the interfaces they implement using NeXTstep. The package requires a minimum of effort to install. Additionally, NMK operates virtually transparently. It has been implemented according to standard NeXT guidelines and in no way interferes with Interface Builder or standard coding practices.

The kit contains two modules, Recorders and Historian. Both are documented in their own separate documentation. In order to utilize NMK, a programmer must add Recorders to the program of interest. Once installed, Recorders takes care of measuring the NeXTstep GUI and storing the data in a transcript file. Recorders maintains one transcript file (which contains running totals) for each application. A developer can view the data contained in a

transcript file by means of Historian. This program provides a browser-style interface that allows users to explore all the data collected.

5.1 Recorders

At first, NMK was intended to include a loadable palette of interface objects for Interface Builder. Such a strategy would have allowed the rapid development of self-measuring applications, but would not have benefited already-written applications. Through a fortuitous combination of Objective-C features, however, NMK requires no special palettes. It can add empirical data gathering to any standard NeXT application for which source code is available.

This powerful feature derives from Objective-C's poseAs method. PoseAs

allows a subclass to assume the identity of its parent, provided that the subclass does not declare any additional instance variables. Thus, by sending FooCover a message to poseAs Foo, all messages intended for Foo (which would be part of standard NeXTstep) will be handled by FooCover (which would be part of NMK). Because FooCover cannot have any additional instance variables, a special technique must be used to store statistics.

Objective-C allows objects, once allocated, to be assigned a name and an owner object. Programs can retrieve objects by name and owner. In NMK, special custom objects handle the data storage and recording for each type of NeXTstep item. An object such as FooCover creates a recording object for itself the first time it becomes active. This recording object receives a

predetermined name and FooCover owns it. In subsequent calls, FooCover accesses its recording object by the predetermined name and uses itself as the owner, thus avoiding the necessity of an instance variable. Aside from its measurement functions, FooCover inherits the standard behaviors of Foo. The following diagram shows a simplified version of how this discussion works in NMK (Figure 4).

NMK.eps ¬

Figure 4: A Standard NeXTstep Measurement Kit Program

In the diagram, Cover Object corresponds to FooCover; RVars Object

corresponds to FooCover's recording object. Interface Object would be Foo, except that no Foo Object is really created. It is shown to indicate that most of the cover object's behavior derives from the object for which it is posing. RApplication is a subclass of Application that sends the requisite poseAs messages discussed above. The important concepts to remember are that: RApplication acts just as Application; as far as RApplication is concerned, a FooCover is a Foo; algorithm objects can't tell the difference between FooCovers and Foos.

At the foundation of Recorders lies the transcript file. A special custom class, TranscriptManager provides access to transcript files for both Recorders and Historian. TranscriptManager implements a simple database

management module. The data it manages contains some structured fields that apply to all NMK records and some free-form data that the client specifies. Once added to a transcript file, records remain forever.

TranscriptLinker acts as the liaison between TranscriptManager and the recording objects. RApplication creates a TranscriptLinker and gives it a name and owner that the recording objects use to access it. TranscriptLinker handles the problem of associating each recording object (and hence each NMK item) with its proper data across multiple invocations of the application. Each NMK item generates an identifying key based on its attributes (such as name and location). TranscriptLinker uses these keys to keep the links between items and records straight. TranscriptLinker learns about items only

after they have been invoked. This late-linking strategy results in no records being kept for interface items that are never used. Nevertheless, if an item links to the TranscriptLinker once, its record remains in the transcript file, even if the item never links again.

NMK measures buttons, sliders, scrollers, and menu items. NeXTstep uses objects of class ButtonCell to implement buttons, so NMK creates ButtonCellCovers that pose as ButtonCells. Objects of class RButtonVars keep the recording variables for ButtonCellCovers. Things work similarly for SliderCellCover and RSliderVars, ScrollerCover and RScrollerVars, and MenuCellCover and RMenuVars.

In general, NMK items count the total amount of time that users interact with

them. They also measure the time between interactions, the number of interactions, and the results of interactions. For buttons and menus, the results tell whether the user canceled the button-press action. For sliders and scrollers, the results tell where along the range of possible values the items have been set. Scrollers also measure which of the different parts of the scroller the user interacted with. The Recorders documentation discusses this in more depth.

Recorders can be added to an existing project via Interface Builder. The entire process takes about ten minutes. Details are provided in the separate Recorders documentation.

5.2 Historian

Although technically less complicated than Recorders, Historian is equally important. It provides the researcher's eye to the data. Historian organizes transcript file data into a hierarchy. The top level contains the major categories of data: buttons, sliders, scrollers, and menus. Within each category, Historian lists items by their keys. Historian utilizes a standard NeXTstep Browser item in order to allow users to view the hierarchy. When a user selects an item in the browser, a special second window, called a Presenter appears. These windows display the data for the selected item. There are ButtonPresenters, SliderPresenters, ScrollerPresenters, and MenuPresenters. Historian should be self explanatory to anyone who has

experience with the NeXT computer. Further details appear in Historian's separate documentation.

6 Conclusions

NeXTstep Measurement Kit provides a remedy for the current lack of empirical evidence to back up claims about user interfaces. The value of such data has yet to be determined. Nevertheless, NMK provides data for a situation in which none was previously available. At present, the data NMK keeps is quite low-level. It will be able to identify frequently (or infrequently) used aspects of an interface, but it is still of dubious value for illuminating users' states of mind.

User interface design remains a viable independent field. While user-centered system design represents a powerful and useful technique, its claim that the interface, by its very nature, causes problems does not stand up. User interface toolkits, because of their generality, standardization, and capacity for innovation will help to fuel the evolution of advanced computing.

The NeXT environment provides an excellent platform for user interface research. The NeXTstep Measurement Kit provides low-level statistical recording capabilities to standard NeXT applications. NMK can be added to existing applications quite easily.

References

[APPL87] Apple Computer, Inc.; *Human Interface Guidelines: The Apple Desktop Interface*; Addison-Wesley, Reading, MA; 1987.

[CYPH86] Allen Cypher; "The Structure of User's Activities"; *User Centered System Design*; Donald A. Norman and Stephen W. Draper, Editors; Lawrence Erlbaum Associates, London; 1986; pp. 243±263.

[FOLE90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes; *Computer Graphics: Principles and Practice*; Addison-Wesley, Reading, MA; 1990.

[GOMO90] Kathleen Gomoll; "Some Techniques for Observing Users"; *The Art of Human-Computer Interface Design*; Brenda Laurel, Editor;

Addison-Wesley, Reading, MA; 1990; pp. 85±90.

[HUTC86] Edwin L. Hutchins, James D. Hollan, and Donald A. Norman; "Direct Manipulation Interfaces"; *User Centered System Design*; Donald A. Norman and Stephen W. Draper, Editors; Lawrence Erlbaum Associates, London; 1986; pp. 87±124.

[LAUR86] Brenda K. Laurel; "Interface as Mimesis"; *User Centered System Design*; Donald A. Norman and Stephen W. Draper, Editors; Lawrence Erlbaum Associates, London; 1986; pp. 67±85.

[LAUR90] Brenda Laurel; *The Art of Human-Computer Interface Design*; Brenda Laurel, Editor; Addison-Wesley, Reading, MA; 1990.

[NEXT90a] NeXT Computer, Inc.; *NeXTstep Concepts*; NeXT Computer, Inc.,

Redwood City, CA; 1990.

[NEXT90b] NeXT Computer, Inc.; *NeXTstep Reference*; Volumes 1 and 2; NeXT Computer, Inc., Redwood City, CA; 1990.

[NORM88] Donald A. Norman; *The Psychology of Everyday Things*; Basic Books, New York, NY; 1988. Available in paperback as *The Design of Everyday Things*.

[NORM90] Donald A. Norman; "Why Interfaces Don't Work"; *The Art of Human-Computer Interface Design*; Brenda Laurel, Editor; Addison-Wesley, Reading, MA; 1990; pp. 209±219.

[RHEI90] Howard Rheingold; "An Interview with Don Norman"; *The Art of Human-Computer Interface Design*; Brenda Laurel, Editor;

Addison-Wesley, Reading, MA; 1990; pp. 5±10.

[SHNE87] Ben Shneiderman; *Designing the User Interface*; Addison-Wesley, Reading, MA; 1987.

[VERT90] Laurie Vertelney, Michael Arent, and Henry Lieberman; "Two Disciplines in Search of an Interface: Reflections on a Design Problem"; *The Art of Human-Computer Interface Design*; Brenda Laurel, Editor; Addison-Wesley, Reading, MA; 1990; pp. 45±55.

This paper represents my own work in

accordance with University regulations.