The research conducted at UCSD was supported by Contract N00014-79-C-0323, NR 667-437, and by Contract N00014-85-C-0113, NR 667-541, with the Personnel and Training Research Programs of the Office of Naval Research and by a grant from the System Development Foundation. Edwin Hutchins and James Hollan’s research was also sponsored by the Naval Personnel Research and Development Center.

Andrea diSessa’s research was supported, in part, by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contract N00014-75-C-0661.

Rachel Reiman’s work was supported by grant number IST81-200018 from the National Science Foundation.

The Consul system, discussed in William Mark’s chapter, and many of the ideas underlying this approach to user interface design, were developed by the Consul group at the University of Southern California Information Sciences Institute under DARPA funding during the years 1979–1984.

Copyright © 1986 by Lawrence Erlbaum Associates, Inc.
All rights reserved. No part of this book may be reproduced in any form, by photostat, microform, retrieval system, or any other means, without the prior written permission of the publisher.

Lawrence Erlbaum Associates, Inc., Publishers
365 Broadway
Hillsdale, New Jersey 07642

Library of Congress Cataloging-in-Publication Data
Main entry under title:
User centered system design.
Bibliography: p.
Includes index
1. Interactive computer systems—Addresses, essays, lectures. 2. System design—Addresses, essays, lectures. 3. Human engineering—Addresses, essays, lectures. I. Norman, Donald A. II. Draper, Stephen W.
QA76.9.I58U73 1986 004’.01’9 85-25207
ISBN Hardback 0-89859-781-1
ISBN Paperback 0-89859-872-9

Printed in the United States of America

Contents

I User Centered System Design 7

II The Interface Experience 63

III Users’ Understandings 153

IV User Activities 239

V Toward a Pragmatics of Human-Machine Communication 315

VI Information Flow 353

VII The Context of Computing 453
Preface
Authors and Participants
Introduction

Stephen W. Draper and Donald A. Norman
(With a postscript by Clayton Lewis)

I User Centered System Design

1 Architectural Design: An Analogy
Kristina Hooper

2 Issues in Design: Some Notes
Liam J. Bannon

3 Cognitive Engineering
Donald A. Norman

II The Interface Experience

4 Interface as Mimesis
Brenda K. Laurel

5 Direct Manipulation Interfaces
Edwin L. Hutchins, James D. Hollan, and Donald A. Norman

6 Notes on the Future of Programming: Breaking the Utility Barrier
Andrea A. diSessa

III Users' Understandings

7 User Understanding
Mary S. Riley

8 Understanding What's Happening in System Interactions
Clayton Lewis

9 Naive Theories of Computation
David Owen

10 Models of Computation
Andrea A. diSessa

11 Knowledge-Based Interface Design
William Mark

IV User Activities

12 The Structure of Users' Activities
Allen Cypher

13 Psychological Issues in Support of Multiple Activities
Yoshio Miyata and Donald A. Norman

14 Communication Paradigms for a Window System
Rachel Reichman (Adar)

V Toward a Pragmatics of Human–Machine Communication

15 There's More to Interaction Than Meets the Eye: Some Issues in Manual Input
William Buxton

16 Display Managers as the Basis for User–Machine Communication
Stephen W. Draper

VI Information Flow

17 Answers First, Then Questions
David Owen

18 Helping Users Help Themselves
Claire E. O'Malley

19 Helping Users Help Each Other
Liam J. Bannon

20 Designing for Error
Clayton Lewis and Donald A. Norman

21 Computer-Mediated Communication
Liam J. Bannon

VII The Context of Computing

22 From Cognitive to Social Ergonomics and Beyond
John Seely Brown

Glossary
References
Index
Cognitive Engineering

DONALD A. NORMAN

PROLOGUE

Cognitive Engineering, a term invented to reflect the enterprise I find myself engaged in: neither Cognitive Psychology, nor Cognitive Science, nor Human Factors. It is a type of applied Cognitive Science, trying to apply what is known from science to the design and construction of machines. It is a surprising business. On the one hand, there actually is quite a lot known in Cognitive Science that can be applied. But on the other hand, our lack of knowledge is appalling. On the one hand, computers are ridiculously difficult to use. On the other hand, many devices are difficult to use—the problem is not restricted to computers, there are fundamental difficulties in understanding and using most complex devices. So the goal of Cognitive Engineering is to come to understand the issues, to show how to make better choices when they exist, and to show what the tradeoffs are when, as is the usual case, an improvement in one domain leads to deficits in another.

In this chapter I address some of the problems of applications that have been of primary concern to me over the past few years and that have guided the selection of contributors and themes of this book. The chapter is not intended to be a coherent discourse on Cognitive Engineering. Instead, I discuss a few issues that seem central to the
way that people interact with machines. The goal is to determine what are the critical phenomena: The details can come later. Overall, I have two major goals:

1. To understand the fundamental principles behind human action and performance that are relevant for the development of engineering principles of design.

2. To devise systems that are pleasant to use—the goal is neither efficiency nor ease nor power, although these are all to be desired, but rather systems that are pleasant, even fun: to produce what Laurel calls “pleasurable engagement” (Chapter 4).

AN ANALYSIS OF TASK COMPLEXITY

Start with an elementary example: how a person performs a simple task. Suppose there are two variables to be controlled. How should we build a device to control these variables? The control question seems trivial: If there are two variables to be controlled, why not simply have two controls, one for each? What is the problem? It turns out that there is more to be considered than is obvious at first thought. Even the task of controlling a single variable by means of a single control mechanism raises a score of interesting issues.

One has only to watch a novice sailor attempt to steer a small boat to a compass course to appreciate how difficult it can be to use a single control mechanism (the tiller) to affect a single outcome (boat direction). The mapping from tiller motion to boat direction is the opposite of what novice sailors sometimes expect. And the mapping of compass movement to boat movement is similarly confusing. If the sailor attempts to control the boat by examining the compass, determining in which direction to move the boat, and only then moving the tiller, the task can be extremely difficult.

Experienced sailors will point out that this formulation puts the problem in its clumsiest, most difficult form: With the right formulation, or the right conceptual model, the task is not complex. That comment makes two points. First, the description I gave is a reasonable one for many novice sailors: The task is quite difficult for them. The point is not that there are simpler ways of viewing the task, but that even a task that has but a single mechanism to control a single variable can be difficult to understand, to learn, and to do. Second, the comment reveals the power of the proper conceptual model of the situation: The correct conceptual model can transform confusing, difficult tasks into simple, straightforward ones. This is an important point that forms the theme of a later section.

Psychological Variables Differ From Physical Variables

There is a discrepancy between the person’s psychologically expressed goals and the physical controls and variables of the task. The person starts with goals and intentions. These are psychological variables. They exist in the mind of the person and they relate directly to the needs and concerns of the person. However, the task is to be performed on a physical system, with physical mechanisms to be manipulated, resulting in changes to the physical variables and system state. Thus, the person must interpret the physical variables into terms relevant to the psychological goals and must translate the psychological intentions into physical actions upon the mechanisms. This means that there must be a stage of interpretation that relates physical and psychological variables, as well as functions that relate the manipulation of the physical variables to the resulting change in physical state.

In many situations the variables that can easily be controlled are not those that the person cares about. Consider the example of bathtub water control. The person wants to control rate of total water flow and temperature. But water arrives through two pipes: hot and cold. The easiest system to build has two faucets and two spouts. As a result, the physical mechanisms control rate of hot water and rate of cold water. Thus, the variables of interest to the user interact with the two physical variables: Rate of total flow is the sum of the two physical variables; temperature is a function of their difference (or ratio). The problems come from several sources:

1. Mapping problems. Which control is hot, which is cold? Which way should each control be turned to increase or decrease the flow? (Despite the appearance of universal standards for these mappings, there are sufficient variations in the standards, idiosyncratic layouts, and violations of expectations, that each new faucet poses potential problems.)

2. Ease of control. To make the water hotter while maintaining total rate constant requires simultaneous manipulation of both faucets.

3. Evaluation. With two spouts, it is sometimes difficult to determine if the correct outcome has been reached.
Faucet technology evolved to solve the problem. First, mixing spouts were devised that aided the evaluation problem. Then, "single control" faucets were devised that varied the psychological factors directly: One dimension of movement of the control affects rate of flow, another orthogonal dimension affects temperature. These controls are clearly superior to use. They still do have a mapping problem—knowing what kind of movement to which part of the mechanism controls which variable—and because the mechanism is no longer as visible as in the two-faucet case, they are not quite so easy to understand for the first-time user. Still, faucet design can be used as a positive example of how technology has responded to provide control over the variables of psychological interest rather than over the physical variables that are easier and more obvious.

It is surprisingly easy to find other examples of the two-variable—two-control task. The water faucets is one example. The loudness and balance controls on some audio sets is another. The temperature controls of some refrigerator-freezer units is another. Let me examine this latter example, for it illustrates a few more issues that need to be considered, including the invisibility of the control mechanisms and a long time delay between adjustment of the control and the resulting change of temperature.

<table>
<thead>
<tr>
<th>NORMAL SETTINGS</th>
<th>C AND 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLDER FRESH FOOD</td>
<td>C AND 6-7</td>
</tr>
<tr>
<td>COLDEST FRESH FOOD</td>
<td>B AND 8-9</td>
</tr>
<tr>
<td>COLDER FREEZER</td>
<td>D AND 7-8</td>
</tr>
<tr>
<td>WARMER FRESH FOOD</td>
<td>C AND 4-1</td>
</tr>
<tr>
<td>OFF (FRESH FD &amp; FRZ)</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FREEZER</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FRESH FOOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3</td>
</tr>
</tbody>
</table>

There are two variables of concern to the user: the temperature of the freezer compartment and the temperature of the regular "fresh food" compartment. At first, this seems just like the water control example, but there is a difference. Consider the refrigerator that I own. It has two compartments, a freezer and a fresh foods one, and two controls, both located in the fresh foods section. One control is labeled "freezer," the other "fresh food," and there is an associated instruction plate (see the illustration). But what does each control do? What is the mapping between their settings and my goal? The labels seem clear enough, but if you read the "instructions" confusion can rapidly set in. Experience suggests that the action is not as labeled: The two controls interact with one another. The problems introduced by this example seem to exist at almost every level:

1. Matching the psychological variables of interest to the physical variables being controlled. Although the labels on the control mechanisms indicate some relationship to the desired psychological variables, in fact, they do not control those variables directly.

2. The mapping relationships. There is clearly strong interaction between the two controls, making simple mapping between control and function or control and outcome difficult.

3. Feedback. Very slow, so that by the time one is able to determine the result of an action, so much time has passed that the action is no longer remembered, making "correction" of the action difficult.


I suspect that this problem results from the way this refrigerator's cooling mechanism is constructed. The two variables of psychological interest cannot be controlled directly. Instead, there is only one cooling mechanism and one thermostat, which therefore, must be located in either the "fresh food" section or in the freezer, but not both. A good description of this mechanism, stating which control affected which function would probably make matters workable. If one mechanism were clearly shown to control the thermostat and the other to control the relative proportion of cold air directed toward the freezer and fresh foods section, the task would be
The user would be able to get a clear conceptual model of the operation. Without a conceptual model, with a 24-hour delay between setting the controls and determining the results, it is almost impossible to determine how to operate the controls. Two variables: two controls. Who could believe that it would be so difficult?

Even Simple Tasks Involve a Large Number of Aspects

The conclusion to draw from these examples is that even with two variables, the number of aspects that must be considered is surprisingly large. Thus, suppose the person has two psychological goals, \( G_1 \) and \( G_2 \). These give rise to two intentions, \( I_1 \) and \( I_2 \), to satisfy the goals. The system has some physical state, \( S \), realized through the values of its variables: For convenience, let there be two variables of interest, \( V_1 \) and \( V_2 \). And let there be two mechanisms that control the system, \( M_1 \) and \( M_2 \). So we have the psychological goals and intentions (\( G \) and \( I \)) and the physical state, mechanisms, and variables (\( S, M, \) and \( V \)).

First, the person must examine the current system state, \( S \), and evaluate it with respect to the goals, \( G \). This requires translating the physical state of the system into a form consistent with the psychological goal. Thus, in the case of steering a boat, the goal is to reach some target, but the physical state is the numerical compass heading. In writing a paper, the goal may be a particular appearance of the manuscript, but the physical state may be the presence of formatting commands in the midst of the text. The difference between desired goal and current state gives rise to an intention, again stated in psychological terms. This must get translated into an action sequence, the specification of what physical acts will be performed upon the mechanisms of the system. To go from intention to action specification requires consideration of the mapping between physical mechanisms and system state, and between system state and the resulting psychological interpretation.

There may not be a simple mapping between the mechanisms and the resulting physical variables, nor between the physical variables and the resulting psychological states. Thus, each physical variable might be affected by an interaction of the control mechanisms: \( V_1 = f(M_1, M_2) \) and \( V_2 = g(M_1, M_2) \). In turn, the system state, \( S \) is a function of all its variables: \( S = h(V_1, V_2) \). And finally, the mapping between system state and psychological interpretation is complex. All in all, the two variable–two mechanism situation can involve a surprising number of aspects. The list of aspects is shown and defined in Table 3.1.

### Table 3.1: Aspects of a Task

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goals and intentions.</td>
<td>A goal is the state the person wishes to achieve; an intention is the decision to act so as to achieve the goal.</td>
</tr>
<tr>
<td>Specification of the action sequence.</td>
<td>The psychological process of determining the psychological representation of the actions that are to be executed by the user on the mechanisms of the system.</td>
</tr>
<tr>
<td>Mapping from psychological goals and intentions to action sequence.</td>
<td>In order to specify the action sequence, the user must translate the psychological goals and intentions into the desired system state, then determine what settings of the control mechanisms will yield that state, and then determine what physical manipulations of the mechanisms are required. The result is the internal, mental specification of the actions that are to be executed.</td>
</tr>
<tr>
<td>Physical state of the system.</td>
<td>The physical state of the system, determined by the values of all its physical variables.</td>
</tr>
<tr>
<td>Control mechanisms.</td>
<td>The physical devices that control the physical variables.</td>
</tr>
<tr>
<td>Mapping between the physical mechanisms and system state.</td>
<td>The relationship between the settings of the mechanisms of the system and the system state.</td>
</tr>
<tr>
<td>Interpretation of system state.</td>
<td>The relationship between the physical state of the system and the psychological goals of the user can only be determined by first translating the physical state into psychological states (perception), then interpreting the perceived system state in terms of the psychological variables of interest.</td>
</tr>
<tr>
<td>Evaluating the outcome.</td>
<td>Evaluation of the system state requires comparing the interpretation of the perceived system state with the desired goals. This often leads to a new set of goals and intentions.</td>
</tr>
</tbody>
</table>

Toward a Theory of Action

It seems clear that we need to develop theoretical tools to understand what the user is doing. We need to know more about how people actually do things, which means a theory of action. There isn’t any realistic hope of getting the theory of action, at least for a long time, but
certainly we should be able to develop approximate theories. And that is what follows: an approximate theory for action which distinguishes among different stages of activities, not necessarily always used nor applied in that order, but different kinds of activities that appear to capture the critical aspects of doing things. The stages have proved to be useful in analyzing systems and in guiding design. The essential components of the theory have already been introduced in Table 3.1.

In the theory of action to be considered here, a person interacts with a system, in this case a computer. Recall that the person's goals are expressed in terms relevant to the person—in psychological terms—and the system's mechanisms and states are expressed in terms relative to it—in physical terms. The discrepancy between psychological and physical variables creates the major issues that must be addressed in the design, analysis, and use of systems. I represent the discrepancies as two gulfs that must be bridged: the Gulf of Execution and the Gulf of Evaluation, both shown in Figure 3.1.²

**The Gulfs of Execution and Evaluation**

The user of the system starts off with goals expressed in psychological terms. The system, however, presents its current state in physical terms. Goals and system state differ significantly in form and content, creating the Gulfs that need to be bridged if the system can be used (Figure 3.1). The Gulfs can be bridged by starting in either direction. The designer can bridge the Gulfs by starting at the system side and moving closer to the person by constructing the input and output characteristics of the interface so as to make better matches to the psychological needs of the user. The user can bridge the Gulfs by creating plans, action sequences, and interpretations that move the normal description of the goals and intentions closer to the description required by the physical system (Figure 3.2).

**Bridging the Gulf of Execution.** The gap from goals to physical system is bridged in four segments: intention formation, specifying the action sequence, executing the action, and, finally, making contact with the input mechanisms of the interface. The intention is the first step, and it starts to bridge the gulf, in part because the interaction language demanded by the physical system comes to color the thoughts of the person, a point expanded upon in Chapter 5 by Hutchins, Hollan, and Norman. Specifying the action sequence is a nontrivial exercise in matching the internal specification to the external (Moran, 1983). It is what Moran calls matching the internal description of the goals and intentions to the physical system (see Riley & O'Malley, 1985). It is what Moran calls matching the internal specification to the external (Moran, 1983). In the terms of the aspects listed in Table 3.1, specifying the action requires translating the psychological goals of the intention into the changes to be made to the physical variables actually under control of the system. This, in turn, requires following the mapping between the psychological intentions and the physical actions permitted on the mechanisms of the system, as well as the mapping between the physical mechanisms and the resulting physical state variables, and between the physical state of the system and the psychological goals and intentions.

After an appropriate action sequence is determined, the actions must be executed. Execution is the first physical action in this sequence: Forming the goals and intentions and specifying the action sequence were all mental events. Execution of an action means to do something, whether it is just to say something or to perform a complex motor

---

¹ There is little prior work in psychology that can act as a guide. Some of the principles come from the study of servomechanisms and cybernetics. The first study known to me in psychology—and in many ways still the most important analysis—is the book *Plans and the Structure of Behavior* by Miller, Galanter, and Pribram (1960) early in the history of information processing psychology. Powers (1973) applied concepts from control theory to cognitive concerns. In the work most relevant to the study of Human-Computer Interaction, Card, Moran, and Newell (1983), analyzed the cycle of activities from Goal through Selection: the GOMS model (Goal, Operator, Methods, Selection). Their work is closely related to the approach given here. This is an issue that has concerned me for some time, so some of my own work is relevant: the analysis of errors, of typing, and of the attentional control of actions (Norman, 1981a, 1984b, 1986; Norman & Shallice, 1985; Rumelhart & Norman, 1982).

² The emphasis on the discrepancy between the user and the system, and the suggestion that we should conceive of the discrepancy as a Gulf that must be bridged by the user and the system designer, came from Jim Hollan and Ed Hutchins during one of the many revisions of the Direct Manipulation chapter (Chapter 5).
FIGURE 3.2. Bridging the Gulfs of Execution and Evaluation. The Gulf of Execution is bridged from the psychology side by the user's formation of intentions relevant to the system and the determination of an action sequence. It is bridged from the system side when the designer of the system builds the input characteristics of the interface. The Gulf of Evaluation is bridged from the psychology side by the user's perception of the system state and the interpretation placed on that perception, which is then evaluated by comparing it with the original goals and intentions. It is bridged from the system side when the designer builds the output characteristics of the interface.

sequence. Just what physical actions are required is determined by the choice of input devices on the system, and this can make a major difference in the usability of the system. Because some physical actions are more difficult than others, the choice of input devices can affect the selection of actions, which in turn affects how well the system matches with intentions. On the whole, theorists in this business tend to ignore the input devices, but in fact, the choice of input device can often make an important impact on the usability of a system. (See Chapter 15 by Buxton for a discussion of this frequently overlooked point.)

Bridging the Gulf of Evaluation. Evaluation requires comparing the interpretation of system state with the original goals and intentions. One problem is to determine what the system state is, a task that can be assisted by appropriate output displays by the system itself. The outcomes are likely to be expressed in terms of physical variables that bear complex relationships to the psychological variables of concern to the user and in which the intentions were formulated. The gap from system to user is bridged in four segments: starting with the output displays of the interface, moving to the perceptual processing of those displays, to its interpretation, and finally, to the evaluation—the comparison of the interpretation of system state with the original goals and intention. But in doing all this, there is one more problem, one just beginning to be understood, and one not assisted by the usual forms of displays: the problem of level. There may be many levels of outcomes that must be matched with different levels of intentions (see Norman, 1981a; Rasmussen in press; Rasmussen & Lind, 1981). And, finally, if the change in system state does not occur immediately following the execution of the action sequence, the resulting delay can severely impede the process of evaluation, for the user may no longer remember the details of the intentions or the action sequence.

Stages of User Activities
A convenient summary of the analysis of tasks is that the process of performing and evaluating an action can be approximated by seven stages of user activity3 (Figure 3.3):

- Establishing the Goal
- Forming the Intention
- Specifying the Action Sequence
- Executing the Action
- Perceiving the System State
- Interpreting the State
- Evaluating the System State with respect to the Goals and Intentions

3 The last two times I spoke of an approximate theory of action (Norman, 1984a, 1985) I spoke of four stages. Now I speak of seven. An explanation seems to be in order. The answer really is simple. The full theory of action is not yet in existence, but whatever its form, it involves a continuum of stages on both the action/execu tion side and the perception/evaluation side. The notion of stages is a simplification of the underlying theory: I do not believe that there really are clean, separable stages. However, for practical application, approximating the activity into stages seems reasonable and useful. Just what division of stages should be made, however, seems less clear. In my original formulations, I suggested four stages: intention, action sequence, execution, and evaluation. In this chapter I separated goals and intentions and expanded the analysis of evaluation by adding perception and interpretation, thus making the stages of evaluation correspond better with the stages of execution: Perception is the evaluatory equivalent of execution, interpretation the equivalent of the action sequence, and evaluation the equivalent of forming the intention. The present formulation seems a richer, more satisfactory analysis.
FIGURE 3.3. Seven stages of user activities involved in the performance of a task. The primary, central stage is the establishment of the goal. Then, to carry out an action requires three stages: forming the intention, specifying the action sequence, and executing the action. To assess the effect of the action also requires three stages, each in some sense complementary to the three stages of carrying out the action: perceiving the system state, interpreting the state, and evaluating the interpreted state with respect to the original goals and intentions.

Real activity does not progress as a simple sequence of stages. Stages appear out of order, some may be skipped, some repeated. Even the analysis of relatively simple tasks demonstrates the complexities. Moreover, in some situations, the person is reactive—event or data driven—responding to events, as opposed to starting with goals and intentions. Consider the task of monitoring a complex, ongoing operation. The person's task is to respond to observations about the state of the system. Thus, when an indicator starts to move a bit out of range, or when something goes wrong and an alarm is triggered, the operator must diagnose the situation and respond appropriately. The diagnosis leads to the formation of goals and intentions; evaluation includes not only checking on whether the intended actions were executed properly and intentions satisfied, but whether the original diagnosis was appropriate. Thus, although the stage analysis is relevant, it must be used in ways appropriate to the situation.

Consider the example of someone who has written a letter on a computer word-processing system. The overall goal is to convey a message to the intended recipient. Along the way, the person prints a draft of the letter. Suppose the person decides that the draft, shown in Figure 3.4A, doesn't look right: The person, therefore, establishes the intention "Improve the appearance of the letter." Call this first intention intention₁. Note that this intention gives little hint of how the task is to be accomplished. As a result, some problem solving is required, perhaps ending with intention₂: "Change the indented paragraphs to block paragraphs." To do this requires intention₃: "Change the occurrences of .pp in the source code for the letter to .sp." This in turn requires the person to generate an action sequence appropriate for the text editor, and then, finally, to execute the actions on the computer keyboard. Now, to evaluate the results of the operation requires still further operations, including generation of a fourth intention, intention₄: "Format the file" (in order to see whether intention₂ and intention₁ were satisfied). The entire sequence of stages is shown in Figure 3.4B. The final product, the reformatted letter, is shown in Figure 3.4C. Even intentions that appear to be quite simple (e.g., intention₄: "Approve the appearance of the letter") lead to numerous subintentions. The intermediary stages may require generating some new subintentions.

Practical Implications

The existence of the two gulfs points out a critical requirement for the design of the interface: to bridge the gap between goals and system. Moreover, as we have seen, there are only two ways to do this: move the system closer to the user; move the user closer to the system. Moving from the system to the user means providing an interface that matches the user's needs, in a form that can be readily interpreted and manipulated. This confronts the designer with a large number of issues. Not only do users differ in their knowledge, skills, and needs, but for even a single user the requirements for one stage of activity can conflict with the requirements for another. Thus, menus can be thought of as information to assist in the stages of intention formation and action specification, but they frequently make execution more
Many systems can be characterized by how well they support the different stages. The argument over whether action specification should be done by command language or by pointing at menu options or icons turns out to be an argument over the relative merits of support for the stages of Execution and Action Specification.

Visual presence can aid the various stages of activity. Thus, we give support to the generation of intentions by reminding the user of what is possible. We support action selection because the visible items act as a direct translation into possible actions. We aid execution, especially if execution by pointing (throwing switches) is possible. And we aid evaluation by making it possible to provide visual reminders of what was done. Visual structure can aid in the interpretation. Thus, for some purposes, graphs, pictures, and moving images will be superior to words: In other situations words will be superior.

Moving from psychological variables to physical variables can take effort. The user must translate goals conceived in psychological terms to actions suitable for the system. Then, when the system responds, the user must interpret the output, translating the physical display of the interface back into psychological terms. The major responsibility should rest with the system designer to assist the user in understanding the system. This means providing a good, coherent design model and a consistent, relevant system image.

CONCEPTUAL MODELS AND THE SYSTEM IMAGE

There are two sides to the interface: the system side and the human side. The stages of execution and perception mediate between psychological and physical representations. And the input mechanism and output displays of the system mediate between the psychological and physical representations. We change the interface at the system side through proper design. We change the interface at the human side through training and experience. In the ideal case, no psychological effort is required to bridge the gulf. But such a situation occurs only either with simple situations or with experienced, expert users. With complex tasks or with nonexpert users, the user must engage in a planning process to go from intentions to action sequence. This planning process, often-times involving active problem solving, is aided when the
person has a good conceptual understanding of the physical system, an argument developed more fully by Riley in Chapter 7.

Think of a conceptual model of the system as providing a scaffolding upon which to build the bridges across the gulfs. The scaffoldings provided by these conceptual models are probably only important during learning and trouble-shooting. But for these situations they are essential. Expert users can usually do without them. They allow the user to derive possible courses of action and possible system responses. The problem is to design the system so that, first, it follows a consistent, coherent conceptualization—a design model—and, second, so that the user can develop a mental model of that system—a user model—consistent with the design model.

Mental models seem a pervasive property of humans. I believe that people form internal, mental models of themselves and of the things and people with whom they interact. These models provide predictive and explanatory power for understanding the interaction. Mental models evolve naturally through interaction with the world and with the particular system under consideration (see Owen’s description in Chapter 9 and the discussion by Riley, Chapter 7). These models are highly affected by the nature of the interaction, coupled with the person’s prior knowledge and understanding. The models are neither complete nor accurate (see Norman, 1983c), but nonetheless they function to guide much human behavior.

There really are three different concepts to be considered: two mental, one physical. First, there is the conceptualization of the system held by designer; second, there is the conceptual model constructed by the user; and third, there is the physical image of the system from which the users develop their conceptual models. Both of the conceptual models are what have been called “mental models,” but to separate the several different meanings of that term, I refer to these two aspects by different terms. I call the conceptual model held by the designer the Design Model, and the conceptual model formed by the user the User’s Model. The third concept is the image resulting from the physical structure that has been built (including the documentation and instructions): I call that the System Image.

The Design Model is the conceptual model of the system to be built. Ideally, this conceptualization is based on the user’s task, requirements, and capabilities. The conceptualization must also consider the user’s background, experience, and the powers and limitations of the user’s information processing mechanisms, most especially processing resources and short-term memory limits.

The user develops a mental model of the system—the User’s Model. Note that the user model is not formed from the Design Model: It results from the way the user interprets the System Image. Thus, in many ways, the primary task of the designer is to construct an appropriate System Image, realizing that everything the user interacts with helps to form that image: the physical knobs, dials, keyboards, and displays, and the documentation, including instruction manuals, help facilities, text input and output, and error messages. The designer should want the User’s Model to be compatible with the underlying conceptual model, the Design Model. And this can only happen through interaction with the System Image. These comments place a severe burden on the designer. If one hopes for the user to understand a system, to use it properly, and to enjoy using it, then it is up to the designer to make the System Image explicit, intelligible, consistent. And this goes for everything associated with the system. Remember too that people do not always read documentation, and so the major (perhaps entire) burden is placed on the image that the system projects.4

4 The story is actually more complex. The “user’s model” can refer to two distinctive things: the individual user’s own personal, idiosyncratic model (which is the meaning I intended); or the generalized “typical user” model that is what the designer develops to help in the formulation of the “Design Model.” I jumped between these two different meanings in this paragraph. Finally, there is yet another model to worry about: the model that an intelligent program might construct of the person with which it is interacting. This too has been called a user model and is discussed by Mark in Chapter 11.
There do exist good examples of systems that present a System Image to the user in a clear, consistent fashion, following a carefully chosen conceptual model in such a way that the User’s Model matches the Design Model. One example is the spreadsheet programs (starting with VISICALC), systems that match the conceptualizations of the targeted user, the accountant or budget planner. Another good example is the stack calculator, especially the early designs from Hewlett Packard. And a third example is the "office desk" metaphor followed in the Xerox Star, Apple Lisa and Macintosh workstations.

It is easier to design consistent Design Models for some things than for others. In general, the more specialized the tool, the higher the level at which a system operates, the easier the task. Spreadsheets are relatively straightforward. General purpose operating systems or programming languages are not. Whenever there is one single task and one set of users, the task of developing the conceptual model is much simplified. When the system is general purpose, with a relatively unlimited set of users and power, then the task becomes complex, perhaps undoable. In this case, it may be necessary to have conceptualizations that depend on the use to which the system is being put.

This discussion is meant to introduce the importance and the difficulties of conceptual models.5 Further discussion of these issues occurs throughout this book, but most especially in the chapters by diSessa (Chapter 10), Mark (Chapter 11), Owen (Chapter 9), and Riley (Chapter 7).

ON THE QUALITY OF HUMAN-COMPUTER INTERACTION

The theme of quality of the interaction and "conviviality" of the interface is important, a theme worth speaking of with force. So for the moment, let me move from a discussion of theories of action and conceptual models and speak of the qualitative nature of human-computer interaction. The details of the interaction matter, ease of use matters, but I want more than correct details, more than a system that is easy to learn or to use: I want a system that is enjoyable to use.

This is an important, dominating design philosophy, easier to say than to do. It implies developing systems that provide a strong sense of understanding and control. This means tools that reveal their underlying conceptual model and allow for interaction, tools that emphasize comfort, ease, and pleasure of use: for what Illich (1973) has called convivial tools. A major factor in this debate is the feeling of control that the user has over the operations that are being performed. A "powerful," "intelligent" system can lead to the well documented problems of "overautomation," causing the user to be a passive observer of operations, no longer in control of either what operations take place, or of how they are done. On the other hand, systems that are not sufficiently powerful or intelligent can leave too large a gap in the mappings from intention to action execution and from system state to psychological interpretation. The result is that operation and interpretation are complex and difficult, and the user again feels out of control, distanced from the system.

Laurel approaches this issue of control over one’s activities from the perspective of drama in her chapter, Interface as Mimesis (Chapter 4). To Laurel, the critical aspect is "pleasurable engagement," by which she means the complete and full engagement of the person in pursuit of the "end cause" of the activity. The Computer should be invisible to the user, acting as the means by which the person enters into the engagement, but avoiding intrusion into the ongoing thoughts and activities.

The Power of Tools

When I look around at instances of good system design—systems that I think have had profound influence upon the users, I find that what seems more important than anything else is that they are viewed as tools. That is, the system is deemed useful because it offers powerful tools that the user is able to apply constructively and creatively, with understanding. Here is a partial list of system innovations that follow these principles:

- Smalltalk. This language—and more importantly, the design philosophy used in getting there—emphasize the development of tools at an appropriate conceptual level, with object-oriented, message-passing software, where new instances or procedures
are derived from old instances, with derived (inherited) conditions and values, and with the operations visible as graphic objects, if you so want them to be (Goldberg, 1984; Tesler, 1981).

- The Xerox Star computer. A carefully done, psychologically motivated approach to the user interface, emphasizing a consistent, well-thought-through user model (Smith, Irby, Kimball, Verplank, & Harslem, 1982). The implementation has changed how we think of interfaces. The Star was heavily influenced by Smalltalk and it, in turn, led to the Apple Lisa and Macintosh.

- UNIX. The underlying philosophy is to provide a number of small, carefully crafted operations that can be combined in a flexible manner under the control of the user to do the task at hand. It is something like a construction set of computational procedures. The mechanisms that make this possible are a consistent data structure and the ability to concatenate programs (via "pipes" and input-output redirection). The interface suffers multiple flaws and is easily made the subject of much ridicule. But the interface has good ideas: aliases, shell scripts, pipes, terminal independence, and an emphasis on shared files and learning by browsing. Elsewhere I have scolded it for its shortcomings (Compton, 1984; Norman, 1981b), but we should not overlook its strengths.


- Spreadsheets. Merging the computational power of the computer with a clean, useful conceptual model, allowing the interface to drive the entire system, providing just the right tools for a surprising variety of applications.

- Steamer. A teaching system based on the concept of intelligent graphics that make visible to the student the operations of an otherwise abstract and complex steam generator system for large ships. (Hollan, Hutchins, & Weizman, 1984).

- Bill Budge's Pinball Construction Set (Budge, 1983). A game, but one that illustrates the toolkit notion of interface, for the user can manipulate the structures at will to create the game of choice. It is easy to learn, easy to use, yet powerful. There is no such thing as an illegal operation, there are no error messages—and no need for any. Errors are simply situations where the operation is not what is desired. No new concepts are in this game over those illustrated by the other items on this list, but the other examples require powerful computers, whereas this works on home machines such as the Apple II, thus bringing the concept to the home.

This list is idiosyncratic. It leaves out some important examples in favor of ones of lesser importance. Nonetheless, these are the items that have affected me the most. The major thing all these systems offer is a set of powerful tools to the user.

The Problem With Tools

The Pinball Construction Set illustrates some of the conflicts that tools present, especially conflict over how much intelligence should be present. Much as I enjoy manipulating the parts of the pinball sets, much as my 4-year-old son could learn to work it with almost no training or bother, neither of us are any good at constructing pinball sets. I can't quite get the parts in the right places: When I stretch a part to change its shape, I usually end up with an unworkable part. Balls get stuck in weird corners. The action is either too fast or too slow. Yes, it is easy to change each problem as it is discovered, but the number seems endless. I wish the tools were more intelligent—do as I am intending, not as I am doing. (This point is examined in more detail in Chapter 5 by Hutchins, Hollan, and Norman.)

Simple tools have problems because they can require too much skill from the user. Intelligent tools can have problems if they fail to give any indication of how they operate and of what they are doing. The user can feel like a bystander, watching while unexplained operations take place. The result is a feeling of lack of control over events. This is a serious problem, one that is well known to students of social psychology. It is a problem whether it occurs to the individual while interacting with colleagues, while a passenger in a runaway vehicle, or while using a computer. If we take the notion of "conviviality" seriously, we will develop tools that make visible their operations and assumptions. The argument really comes down to presenting an appropriate system image to the user, to assist the user's understanding
of what is going on: to keep the user in control. These are topics discussed in Mark’s chapter (Chapter 11). They require, among other things, developing a good model of the user. In addition, the user must have a good user’s model of the system.

When systems take too much control of the environment, they can cause serious social problems. Many observers have commented on the dehumanizing results of automation in the workplace. In part, this automatically results from the systems that take control away from the users. As Ehn and Kyng (1984) put it, such a result follows naturally when the office or workplace is thought of as a system, so that the computer reduces "the jobs of the workers to algorithmic procedures" minimizing the need for skill or control, and thereby the attractiveness of the workplace. The alternative view, that of tools, offers more control to the worker. For Eng and Kyng, tools "are under complete and continuous manual control of the worker, are fashioned for the use of the skilled worker to create products of good use quality, and are extensions of the accumulated knowledge of tools and materials of a given labour process." The problem arises over and over again as various workplaces become automated, whether it is the factory, the office, or the aviation cockpit. I believe the difficulties arise from the tension between the natural desire to want intelligent systems that can compensate for our inadequacies and the desire to feel in control of the outcome. Proponents of automatic systems do not wish to make the workplace less pleasant. On the contrary, they wish to improve it. And proponents of tools often wish for the power of the automated systems. (See Chapters 2, 19, and 21 by Bannon for further discussion of these issues.)

The Gulfs of Execution and Evaluation, Revisited

The stages of action play important roles in the analysis of the interface, for they define the psychological stages that need support from the interface. Moreover, the quality of the interaction probably depends heavily upon the "directness" of the relationship between the psychological and physical variables: just how the Gulfs of Figure 3.1 are bridged. The theory suggests that two of the mappings of Table 3.1 play critical roles: (a) the mapping from the psychological variables in which the goals are stated to the physical variables upon which the control is actually exerted; (b) the mapping from the physical variables of the system to psychological variables. The easier and more direct these two mappings, the easier and more pleasant the learning and use of the interface, at least so goes the theory. In many ways, the design efforts must focus upon the mappings much more than the stages. This issue forms the focus of much of the discussion in the chapter by Hutchins, Hollan, and Norman (Chapter 5), where it is the mappings that are discussed explicitly as helping bridge the gulf between the demands of the machine and the thought processes and actions of the user. In that chapter the discussion soon turns to the qualitative feeling of control that can develop when one perceives that manipulation is directly operating upon the objects of concern to the user: The actions and the results occur instantaneously upon the same object. That chapter provides a start toward a more formal analysis of these qualitative feelings of "conviviality" or what Hutchins, Hollan, and Norman call "direct engagement" with the task.

The problem of level. A major issue in the development of tools is to determine the proper level. Tools that are too primitive, no matter how much their power, are difficult to work with. The primitive commands of a Turing machine are of sufficient power to do any task doable on a computer, but who would ever want to program any real task with them? This is the "Turing tarpi" discussed in Chapter 5 by Hutchins, Hollan, and Norman. When I program a computer, I want a language that matches my level of thought or action. A programming language is precisely in the spirit of a tool: It is a set of operations and construction procedures that allows a machine to do anything doable, unrestricted by conventions or preconceived notions. The power of computers comes about in part because their languages do follow the tool formulation. But not everyone should do this kind of programming. Most people need higher-level tools, tools where the components are already closely matched to the task. On the other hand, tools that are at too high a level are too specialized. An apple-peeler is well matched to its purpose, but it has a restricted set of uses. Spelling checkers are powerful tools, but of little aid outside their domain. Specialized tools are invaluable when

6 Streitz (1985) has expressed a similar view, stating that "An interactive computer system (ICS) is the more user-oriented the less discrepancies do exist between the relevant knowledge representations on the user’s side and on the side of the ICS."
they match the level and intentions of the user, frustrating when they do not.

How do we determine the proper level of a tool? That is a topic that needs more study. There are strong and legitimate arguments against systems that are too specialized. Equally, there are strong arguments against tools that are too primitive, that operate at too low a level. We want higher-level tools that are crafted to the task. We need lower-level tools in order to create and modify higher-level ones. The level of the tool has to match the level of the intention. Again, easier to say than to do.

DESIGN ISSUES

Designing computer systems for people is especially difficult for a number of reasons. First, the number of variables and potential actions is large, possibly in the thousands. Second, the technology available today is limited: limited in the nature of what kinds of input mechanisms exist; limited in the form and variety of output; limited in the amount of affordable memory and computational power. This means that the various mappings (see Table 3.1) are particularly arbitrary. On the other hand, the computer has the potential to make visible much more of the operation of the system and, more importantly, to translate the system’s operations into psychologically meaningful variables and displays than any other machine. But, as the opening sections of this chapter attempted to demonstrate, the problem is intrinsically difficult: It isn’t just computers that are difficult to use, interaction with any complex device is difficult.

Any real system is the result of a series of tradeoffs that balance one design decision against another, that take into account time, effort, and expense. Almost always the benefits of a design decision along one dimension lead to deficits along some other dimension. The designer must consider the wide class of users, the physical limitations, the constraints caused by time and economics, and the limitations of the technology. Moreover, the science and engineering disciplines necessary for a proper design of the interface do not yet exist. So what is the designer to do? What do those of us who are developing the design principles need to do? In this section I review some of the issues, starting with a discussion of the need for approximate theory, moving to a discussion of the general nature of tradeoffs, and then to an exhortation to attend first to the first-order issues. In all of this, the goal is a User-Centered Interface, which means providing intelligent, understandable, tools that bridge the gap between people and systems: convivial tools.

What Is It We Want in Computer Design?

Approximate science. In part we need a combined science and engineering discipline that guides the design, construction, and use of systems. An important point to realize is that approximate methods suffice, at least for most applications. This is true of most applied disciplines, from the linear model of transistor circuits to the stress analysis of bridges and buildings: The engineering models are only approximations to reality, but the answers are precise enough for the purpose. Note, of course, that the designer must know both the approximate model and its limits.

Consider an example from Psychology: the nature of short-term memory (STM). Even though there is still not an agreed upon theory of memory, and even though the exact nature of STM is still in doubt, quite a bit is known about the phenomena of STM. The following approximation captures a large portion of the phenomena of STM and is, therefore, a valuable tool for many purposes:

The five-slot approximate model of STM. Short-term memory consists of 5 slots, each capable of holding one item (which might be a pointer to a complex memory structure). Each item decays with a half-life of 15 seconds. Most information is lost from STM as a result of interference, new information that takes up the available slots.

Although the approximate model is clearly wrong in all its details, in most practical applications the details of STM do not matter: This approximate model can be very valuable. Other approximate models are easy to find. The time to find something can be approximated by assuming that one object can be examined within the fovea at any one time, and that saccades take place at approximately 5 per second. Reaction and decision times can be approximated by cycles of 100 milliseconds. The book by Card, Moran, and Newell (1983) provides sophisticated examples of the power of approximate models of human cognition. All these models can be criticized at the theoretical level. But they all provide numerical assessment of behavior that will be accurate enough for almost all applications.
Tradeoffs

Design is a series of tradeoffs: Assistance for one stage is apt to interfere with another. Any single design technique is apt to have its virtues along one dimension compensated by deficiencies along another. Each technique provides a set of tradeoffs. The lesson applies to almost any aspect of design. Add extra help for the unskilled user and you run the risk of frustrating the experienced user. Make the display screen larger and some tasks get better, but others get more confused. Display more information, and the time to paint the display goes up, the memory requirement goes up, programs become larger, bulkier, slower. It is well known that different tasks and classes of users have different needs and requirements.

The design choices depend on the technology being used, the class of users, and the goals of the design. The designers must decide which aspects of the interface should gain, which can be left wanting. This focus on the tradeoffs emphasizes that the design problem must be looked at as a whole, not in isolated pieces; for the optimal choice for one part of the problem will probably not be optimal for another. According to this view, there are no correct answers; only tradeoffs among alternatives.

It might be useful to point out that although there may not be any best solution to a problem in which the needs of different parts conflict, there is a worst solution. And even if no design is “best” along all dimensions, some designs are clearly better than others—along all dimensions. It clearly is possible to design a bad system. Equally, it is possible to avoid bad design.

The prototypical tradeoff: information versus time. One basic tradeoff pervades many design issues: Factors that increase informativeness tend to decrease the amount of available workspace and system responsiveness. On the one hand, the more informative and complete the display, the more useful when the user has doubts or lacks understanding. On the other hand, the more complete the display, the longer it takes to be displayed and the more space it must occupy physically. This tradeoff of amount of information versus space and time appears in many guises and is one of the major interface issues that must be handled (Norman, 1983a). To appreciate its importance, one has only to examine a few recent commercial offerings, highly touted for their innovative (and impressive) human factors design that were intended to make the system easy and pleasurable to use, but which so degraded system response time that serious user complaints resulted. The term “user friendly” has taken on a negative meaning as a result of badly engineered tradeoffs, sacrificing utility, efficiency, and ease of use for the benefit of some hypothetical, ill-informed, first-time user.

It is often stated that current computer systems do not provide beginning users with sufficient information. However, the long, informative displays or sequence of questions, options, or menus that may make a system usable by the beginner are disruptive to the experienced user who knows exactly what action is to be specified and wishes to minimize the time and mental effort required to do the specification. The tradeoff here is not only between different needs, but between different stages of activity. After all, the extra information required by the beginner would not bother the experienced users if they could ignore it. However, this information usually cannot be ignored. It is apt to take excess time to be displayed or to use up valuable space on the display, in either case impeding the experienced users in executing and evaluating their actions. We pit the experienced user’s requirement for ease of specification against the beginner’s requirement for knowledge.

First- and second-order issues. One major tradeoff concerns just which aspects of the system will be worked on. With limited time and people, the design team has to make decisions: Some parts of the system will receive careful attention, others will not. Each different aspect of the design takes time, energy, and resources, none of which is apt to be readily available. Therefore, it is important to be able to distinguish the first order effects from the secondary effects—the big issues from the little issues.

I argue that it is the conceptual models that are of primary importance: the design model, the system image, the user’s model. If you don’t have the right design model, then all else fades to insignificance. Get the major issue right first—the Design Model and the System Image. Then, and only then, worry about the second order issues.

Example: VISICALC. At the time VISICALC was introduced, it represented a significant breakthrough in design. Bookkeepers and accountants were often wary of computers, especially those who were involved in small and medium size enterprises where they had to work alone, without the assistance of corps of programmers and computer specialists. VISICALC changed all this. It let the users work on their own terms, putting together a "spreadsheet" of figures, readily changing the numbers and watching the implications appear in the relevant spots.
It would be useful to explore the various design issues involved in the construction of VISICALC. The designers not only were faced with the creation of a conceptualization unlike anything else that existed, but they chose to do it on a relatively small and limited machine, one in which the two major languages available were BASIC and Assembler code, which could only display 24 rows of 40 columns worth of upper-case letters and digits. Yet, spreadsheets require matrices with hundreds of rows and columns of numerals. The success of VISICALC was due both to the power of the original conceptualization and the clever use of design techniques to overcome the limitations of the machine. Probably an important key to its success was that the design team consisted of just two people, one a user (at the time, he was a student in the Harvard Business School who needed a tool to do business analyses and projections), the other a programmer.

But look at the command structure used in VISICALC: cryptic, obscure, and unmeaningful. It is easy to make errors, difficult to remember the appropriate operations. The choice of command names could be used as an exercise in how not to do things, for they appear to be the typical conventions chosen by computer programmers, for computer programmers. The point of this is to note that VISICALC was a success story, despite the poor choice of command structure. Yes, VISICALC would have been much improved had the commands been better. People would have liked it better, users would have been happier. But the commands were a second-order issue. The designers of VISICALC were working with limited time, manpower, and budget: They were wise in concentrating on the important conceptualizations and letting the problems of command names go for later. I certainly do not wish to advocate the use of poor commands, but the names are second-order issues.

Why was the command structure less important than the overall conceptual structure? Two factors helped:

- The system was self-contained.
- The typical user was a frequent user.

First, VISICALC was a self-contained system. That is, many users of VISICALC, especially the first wave of users, used only VISICALC. They put the floppy disk containing VISICALC into the computer, turned it on, did their work, and then turned off the computer. Therefore, there were no conflicts between the command choices used by VISICALC and other programs. This eliminated one major source of difficulty. Second, most users of VISICALC were practiced, experienced users of the system. The prime audience of the system was the professional who worked with spreadsheet computations on a regular basis. Therefore, the commands would be expected to be used frequently. And whenever there is much experience and practice, lack of meaning and consistency is not so important. Yes, the learning time might be long, but it only need take place once and then, once the commands have been learned well, they become automatic, causing no further difficulty. Choices of command names are especially critical when many different systems are to be used, each with its own cryptic, idiosyncratic choice of names. Problems arise when different systems are involved, oftentimes with similar functions that have different names and conventions, and with similar names that have different meanings. When a system is heavily used by beginners or casual users, then command names take on added significance.

### Prescriptions for Design Principles

What is it that we need to do? What should we accomplish? What is the function of *Cognitive Engineering*? The list of things is long, for here we speak of creating an entirely new discipline, one moreover that combines two already complex fields: psychology and computer science. Moreover, it requires breaking new ground, for our knowledge of what fosters good interactions among people and between people and devices is young, without a well-developed foundation. We are going to need a good, solid technical grounding in the principles of human processing. In addition, we need to understand the more global issues that determine the essence of interaction. We need to understand the way that hardware affects the interaction: As Chapter 15 by Buxton points out, even subtle changes in hardware can make large changes in the usability of a system. And we need to explore the technology into far richer and more expressive domains than has so far been done.

On the one hand, we do need to go deeper into the details of the design. On the other hand, we need to determine some of the higher, overriding principles. The analysis of the stages of interaction moves us in the former direction, into the details of interaction. In this chapter I have raised a number of the issues relevant to the second issue: the higher, more global concerns of human-machine interaction. The general ideas and the global framework lead to a set of overriding design guidelines, not for guiding specific details of the design, but for structuring how the design process might proceed. Here are some prescriptions for design:

- *Create a science of user-centered design.* For this, we need principles that can be applied at the time of the design, principles that get the design to a pretty good state the first time around.
This requires sufficient design principles and simulation tools for establishing the design of an interface before constructing it. There will still have to be continual iterations, testing, and refinement of the interface—all areas of design need that—but the first pass ought to be close.

- Take interface design seriously as an independent and important problem. It takes at least three kinds of special knowledge to design an interface: first, knowledge of design, of programming and of the technology; second, knowledge of people, of the principles of mental computation, of communication, and of interaction; and third, expert knowledge of the task that is to be accomplished. Most programmers and designers of computer systems have the first kind of knowledge, but not the second or third. Most psychologists have the second, but not the first or third. And the potential user is apt to have the third, but not the first or second. As a result, if a computer system is to be constructed with a truly user-centered design, it will have to be done in collaboration with people trained in all these areas. We need either especially trained interface specialists or teams of designers, some members expert in the topic domain of the device, some expert in the mechanics of the device, and some expert about people. (This procedure is already in use by a number of companies: often those with the best interfaces, I might add.)

- Separate the design of the interface from the design of the system. This is the principle of modularization in design. It allows the previous point to work. Today, in most systems, everyone has access to control of the screen or mouse. This means that even the deepest, darkest, most technical systems programmer can send a message to the user when trouble arises: Hence arises my favorite mystical error message: "longjmp botch, core dump" or du Boulay's favorite compiler error message: "Fatal error in pass zero" (Draper & Norman, 1984; du Boulay & Matthew, 1984). It is only the interface module that should be in communication with the user, for it is only this module that can know which messages to give, which to defer, to know where on the screen messages should go without interfering with the main task, or to know the associated information that should be provided. Messages are interruptions (and sometimes reminders), in the sense described in the chapters by Cypher (Chapter 12) and Miyata and Norman (Chapter 13).

Because they affect the ongoing task, they have to be presented at the right time, at the right level of specification.

Modularity also allows for change: The system can change without affecting the interface; the interface can change without affecting the system. Different users may need different interfaces, even for the same task and the same system. Evaluations of the usability of the interface may lead to changes—the principle of iterative, interactive design—and this should be possible without disruption to the rest of the system. This is not possible if user interaction is scattered throughout the system: It is possible if the interface is a separate, independent module.

- Do user-centered system design: Start with the needs of the user. From the point of view of the user, the interface is the system. Concern for the nature of the interaction and for the user—these are the things that should force the design. Let the requirements for the interaction drive the design of the interface, let ideas about the interface drive the technology. The final design is a collaborative effort among many different disciplines, trading off the virtues and deficits of many different design approaches. But user-centered design emphasizes that the purpose of the system is to serve the user, not to use a specific technology, not to be an elegant piece of programming. The needs of the users should dominate the design of the interface, and the needs of the interface should dominate the design of the rest of the system.

ACKNOWLEDGMENTS

The chapter has been much aided by the comments of numerous people. I thank Eileen Conway for her aid with the illustrations. Julie Norman and Sondra Buffett provided extensive editorial comments for each of the numerous revisions. Liam Bannon, Steve Draper, and Dave Owen provided a number of useful comments and suggestions. Jonathan Grudin was most savage of the lot, and therefore the most helpful. And the Asilomar Workshop group provided a thorough reading, followed by two hours of intensive commentary. All this effort on the part of the critics led to major revision and reorganization. For all this assistance, I am grateful.
DIRECT MANIPULATION: ITS NATURE, FORM, AND HISTORY

The best way to describe a Direct Manipulation interface is by example. Suppose we have a set of data to be analyzed with the numbers stored in matrix form. Their source and meaning are not important for this example: The numbers could be the output of a spreadsheet, a matrix of numerical values from the computations of a conventional programming language, or the results of an experiment. Our goal is to analyze the numbers, to see what relations exist among the rows and columns of the matrix. The matrix of numbers is represented on a computer display screen by an icon. To plot one column against another, we simply get a copy of a graph icon, then draw one line from the output of one column to the x-axis of the graph icon and another line from the output of the second column to the y-axis input of the graph icon: See Figure 5.1A. Now, what was wanted? Erase the lines and reconnect them. Want to see other graphs? Make more copies of the graph icons and connect them. Need a logarithmic transformation of one of the axes? Move up a function icon, type in the algebraic function that is desired \( y = \log x \), in this case) and connect it in the desired data stream. Want the analysis of variance of the logarithm of the data?
FIGURE 5.1. An elementary example of doing simple statistical computations by Direct Manipulation. (A) The basic components: The data are contained in the matrix, represented by the icon in the upper left corner of the screen. At the bottom of the screen are basic icons that represent possible functions. To use one, a copy of the desired icon is moved to the screen and connected up, much as is shown for the graph. (B) More complex interconnections, including the use of a logarithmic transformation of the data, a basic statistical package (for means and standard deviations), and an Analysis of Variance Package (ANOVA).

Connect the matrix to the appropriate statistical icons. These examples are illustrated in Figure 5.1B.

Now consider how we could partition the data. Suppose one result of our analysis was the scatter diagram shown in Figure 5.2A. The straight line that has been fitted through the points is clearly inappropriate. The data fall into two quite different clusters and it would be best to analyze those clusters separately. In the actual data matrix, the points that form the two clusters might be scattered randomly throughout the data set. The regularities are apparent only when we plot them. How do we pull out the clusters? Suppose we could simply circle the points of interest and use each circled set as if it were a new matrix of values, each of which could be analyzed in standard ways, as shown in Figure 5.2B.

The examples of Figures 5.1 and 5.2 are partially implemented,
EDWIN L. HUTCINS, JAMES D. HOLLAN, and DONALD A. NORMAN

Partially hypothetical: But they illustrate a powerful manipulation medium for computation. The promise of Direct Manipulation is that instead of an abstract computational medium, all "programming" is done graphically, in a form that matches the way one thinks about the problem. The desired operations are done simply by moving the appropriate icons onto the screen and connecting them together. Connecting the icons is the equivalent of writing a program or calling on a set of statistical subroutines, but with the advantage of being able to directly manipulate and interact with the data and the connections. There are no hidden operations, no syntax or command names to learn. What you see is what you get. Some classes of syntax errors are eliminated. For example, you can't point at a nonexistent object. The system requires expertise in the task domain, but only minimal knowledge of the computer or of computing.

A number of direct manipulation systems already exist. One was described briefly in Norman's chapter on "Cognitive Engineering": Bill Budge's Pinball Construction Set (Budge, 1983). It allows for the construction of game environments simply by moving the desired pieces onto the game, then stretching, coloring, and otherwise manipulating the environment.

Direct manipulation interfaces seem remarkably powerful. Shneiderman has suggested that direct manipulation systems have the following virtues:

1. Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.
2. Experts can work extremely rapidly to carry out a wide range of tasks, even defining new functions and features.
3. Knowledgeable intermittent users can retain operational concepts.
4. Error messages are rarely needed.
5. Users can see immediately if their actions are furthering their goals, and if not, they can simply change the direction of their activity.
6. Users have reduced anxiety because the system is comprehensible and because actions are so easily reversible. (Shneiderman, 1982, p. 251)

Can this really be true? Certainly there must be problems as well as benefits. It turns out that the concept of Direct Manipulation is complex. A checklist of surface features is unlikely to capture the real sources of power in direct manipulation interfaces. Moreover, although there are important benefits, there are also deficits. There are limitations: Like everything else, direct manipulation systems trade off one set of virtues and vices against another. It is important that we understand all the tradeoffs.

A Brief History of Direct Manipulation

The term "Direct Manipulation" was coined by Shneiderman (see Shneiderman, 1974, 1982, 1983; the 1983 review provides the best treatment of the topic), to refer to interfaces having the following properties:

1. Continuous representation of the object of interest.
2. Physical actions or labeled button presses instead of complex syntax.
3. Rapid incremental reversible operations whose impact on the object of interest is immediately visible. (Shneiderman, 1982, p. 251)

Hints of direct manipulation programming environments have been around for quite some time. The first major landmark is Sutherland's Sketchpad, a graphical design program (Sutherland, 1963). Sutherland's goal was to devise a program that would make it possible for a person and a computer "to converse rapidly through the medium of line drawings." Sutherland's work is a landmark not only because of historical priority but because of the ideas that he helped develop: He was one of the first to discuss the power of graphical interfaces, the conception of a display as "sheets of paper," the use of pointing devices, the virtues of constraint representations, and the importance of depicting abstractions graphically.

Sutherland's ideas took 20 years to have widespread impact. The lag is perhaps due more to hardware limitations than anything else. Highly interactive, graphical programming requires the ready availability of considerable computational power and it is only recently that machines capable of supporting this type of computational environment have become inexpensive enough to be generally available. But now we see these ideas in many of the computer-aided design and computer-aided manufacturing systems; many of which can trace their heritage directly to Sutherland's work. Borning's ThingLab program (1979) explored a general programming environment, building upon many of Sutherland's
ideas within the Smalltalk programming environment. More recently, Direct Manipulation systems have been appearing with reasonable frequency. We have already discussed Bill Budge's Pinball Construction Set (Budge, 1983). Other examples exist in the area of intelligent training systems (e.g., Steamer: Hollan, Hutchins, & Weitzman, 1984, Hollan, Stevens, & Williams, 1980). Steamer makes use of similar techniques and also provide tools for the construction of interactive graphical interfaces. Finally, the spreadsheet program, first available on the Apple II computer and now commonplace, incorporates many of the essential features of Direct Manipulation. In the lead article of Scientific American's special issue on computer software, Kay (1984) claims that the development of dynamic spreadsheet systems gives strong hints that programming styles are in the offing that will make programming as it has been done for the past 40 years obsolete.

The Goal: A Cognitive Account of Direct Manipulation

We see promise in the notion of direct manipulation, but as yet we see no explanation of it. There are systems with attractive features, and claims for the benefits of systems that give the user a certain sort of feeling, and even lists of properties that seem to be shared by systems that provide that feeling, but no account of how particular properties might produce the feeling of directness. The purpose of this chapter is to examine the underlying basis for Direct Manipulation systems. On the one hand, what is it that provides the feeling of "directness"? Why do the examples feel so natural? What is so compelling about the notion? On the other hand, why does it seem so painful at times? Certainly you wouldn't want to do basic programming this way, or would you? What about Norman's complaint (in Chapter 3):

Much as I enjoy manipulating the parts of the pinball sets, much as my 4-year-old son could learn to work it with almost no training or bother, neither of us are any good at constructing pinball sets. I can't quite get the parts in the right places: When I stretch a part to change its shape, I usually end up with an unworkable part. Balls get stuck in weird corners. The action is either too fast or too slow. Yes, it is easy to change each problem as it is discovered, but the number seems endless. I wish the tools were more intelligent—do as I am intending, not as I am doing.

For us, the notion of Direct Manipulation is not a unitary concept nor even something that can be quantified in itself. It is an orienting notion. "Directness" is an impression or a feeling about an interface. What we seek to do here is to characterize the space of interfaces and see where within that picture the range of phenomena that contribute to the feeling of directness might reside. The goal is to give cognitive accounts of these phenomena. At the root of our approach is the assumption that the feeling of directness results from the commitment of fewer cognitive resources. Or put the other way round, the need to commit additional cognitive resources in the use of an interface leads to the feeling of indirectness. As we shall see, some of the production of the feeling of directness is due to adaptation by the user, so that the designer can neither completely control the process, nor take full credit for the feeling of directness that may be experienced by the user.

There is a need to provide a characterization of how interfaces work, both in the computer science and the psychological senses, that will allow us to examine their strengths, weaknesses, and limitations. The computer science sense of "how they do it" concerns the computer-based technologies that contribute to interface power: object-oriented programming, constraint-based systems, logic programming, pointing devices, window systems, etc. In this chapter we focus primarily on the psychological sense of "how they do it" because we feel that this is where the power lies. We believe that while many technological advances may be helpful (new types of input devices, for example), even the existing technology is not used to full advantage because of a lack of understanding of the psychological side of interface design.

We will not attempt to set down hard and fast criteria under which an interface can be classified as direct or not direct. The sensation of directness is always relative. It is often due to the interaction of a number of factors. There are costs associated with every factor that increases the sensation of directness. At present we know of no way to measure the tradeoff values, but we will attempt to provide a framework within which one can say what is being traded off against what.

TWO ASPECTS OF DIRECTNESS: DISTANCE AND ENGAGEMENT

There are two separate and distinct aspects of the feeling of directness. One involves a notion of the distance between one's thoughts and the physical requirements of the system under use. A short distance means that the translation is simple and straightforward, that thoughts are readily translated into the physical actions required by the system and that the system output is in a form readily interpreted in terms of the goals of interest to the user. We call this aspect "distance" to emphasize the fact that directness is never a property of the interface alone, but
involves a relationship between the task the user has in mind and the way that task can be accomplished via the interface. Here the critical issues involve minimizing the effort required to bridge the gulf between the user's goals and the way they must be specified to the system.

The second aspect of directness concerns the qualitative feeling of engagement, the feeling that one is directly manipulating the objects of interest. There are two major metaphors for the nature of human-computer interaction, a conversation metaphor and a model world metaphor. In a system built on the conversation metaphor, the interface is a language medium in which the user and system have a conversation about an assumed, but not explicitly represented world. In this case, the interface is an implied intermediary between the user and the world about which things are said. In a system built on the model world metaphor, the interface is itself a world where the user can act, and that changes state in response to user actions. The world of interest is explicitly represented and there is no intermediary between user and world. Appropriate use of the model world metaphor can create the sensation in the user of acting upon the objects of the task domain themselves. We call this aspect "direct engagement."

Distance

An interface introduces distance to the extent there are gulfs between a person's goals and knowledge and the level of description provided by the systems with which the person must deal. These are the Gulf of Execution and the Gulf of Evaluation (Figure 5.3) that were incorporated by Norman in his chapter on Cognitive Engineering (Chapter 3). The Gulf of Execution is bridged by making the commands and mechanisms of the system match the thoughts and goals of the user as much as possible. The Gulf of Evaluation is bridged by making the output displays present a good Conceptual Model of the system that is readily perceived, interpreted, and evaluated. The goal in both cases is to minimize cognitive effort.

We suggest that the feeling of directness is inversely proportional to the amount of cognitive effort it takes to manipulate and evaluate a system and, moreover, that cognitive effort is a direct result of the Gulfs of Execution and Evaluation. The better the interfaces to the system help bridge the gulfs, the less cognitive effort needed, the more direct the resulting feeling of interaction.

In Chapter 3, Norman suggests that it is useful to consider an approximate theory of action, and in particular, that it is useful to consider separately the seven different stages of activity shown in Figure 5.4 that a person might go through in the performance of a task. For the current discussion, the important point is that the interface must try to match the goals of the person. The two gulfs are bridged in two directions; from the system side by the system interface and from the user side by mental structures and interpretation. The bridges from the user's side are aided through the development of appropriate conceptual models—the design model and the user model discussed in Chapter 3, which are in turn aided by the system image presented by the interface. But a major burden should be carried by the interface itself. The more of the gulf spanned by the interface, the less distance need be bridged by the efforts of the user.

Direct Engagement

The description of the nature of interaction to this point begins to suggest how to make a system less difficult to use, but it misses an important point, a point that is the essence of Direct Manipulation. The analysis of the execution and evaluation process explains why there is difficulty in using a system and it says something about what must be done to minimize the mental effort required to use a system. But there is more to it than that. The systems that best exemplify Direct Manipulation all give us the qualitative feeling that we are directly engaged with control of the objects—not with the programs, not with the computer, but with the semantic objects of our goals and intentions. This is the feeling that Laurel discusses in Chapter 4: a feeling of first-personness, of direct engagement with the objects that concern us. Are we analyzing data? Then we should be manipulating the data
themselves or if we are designing an analysis of data, we should be manipulating the analytic structures themselves. Are we playing a game? Then we should be manipulating directly the game world, touching and controlling the objects in that world, with the output of the system responding directly to our actions, and in a form compatible with them.

Historically, most interfaces have been built on the conversational metaphor. There is power in the abstractions that language provides (we discuss some of this later), but the implicit role of interface as an intermediary to a hidden world denies the user direct engagement with the objects of interest. Instead, the user is in direct contact with linguistic structures, structures that can be interpreted as referring to the objects of interest, but that are not those objects themselves.

Making the central metaphor of the interface that of the model world supports the sensation of directness: Instead of describing the actions of interest, the user performs those actions. In the conventional interface, the system describes the results of the actions: In the model world the system would present directly the actions taken upon the objects. This change in central metaphor is made possible by relatively recent advances in technology. One of the exciting prospects for the study of Direct Manipulation is the exploration of the properties of systems that provide for direct engagement.

Building interfaces on the model world metaphor requires a special sort of relationship between the input interface language and the output interface language. In particular, the output language must represent its subject of discourse in a way that natural language does not normally do. The expressions of a Direct Manipulation output language must behave in such a way that the user can assume that they, in some sense, are the things they refer to. Furthermore, the nature of the relationship between input and output language must be such that an output expression can serve as a component of an input expression. When these conditions are met, it is as if we are directly manipulating the things that the system represents.

Thus, consider a system in which a File is represented by an image on the screen and actions are done by pointing to and manipulating the image. In this case, if we can specify a file by pointing at the screen representation, we have met the goal that an expression in the output "language" (in this case an image) be allowed as a component of the input expression. If we ask for a listing of files, we would want the result to be a physical display that can, in turn, be used directly to specify the further operations to be done. Notice that this is not how a conversation works. In conversation, one may refer to what has previously been said, but one cannot operate upon what has been said. This kind of requirement does not require an interface of pictures, diagrams, or icons. It can be done with words and descriptions. The key properties are that the objects, whatever their form, have behaviors, can be referred to by other objects, and that referring to an object causes it to behave. In the file listing example, we must be able to use the output expression that represents the file in question as a part of the input expression calling for whatever operation we desire upon that file, and the output expression that represents the file must change as a result of being referred to in this way. The goal is to permit the user to act as if the representation is the thing itself.

These conditions are met in many screen editors when the task is the arrangement of strings of characters. The characters appear as they
are typed. They are then available for further operations. We treat them (but not their meanings!) as though they are the things we are manipulating. These conditions are also met in the statistics example with which we opened this chapter (Figure 5.1), in Steamer, and by the Pin-Ball Construction Set. The special conditions are not met in file listing commands on most systems, the commands that allow one to display the names and attributes of file structure. The issue is that the outputs of these commands are "names" of the objects. Operating on the names does nothing to the objects to which they refer. In a Direct Manipulation situation, we would feel that we had the files in front of us, that the program that "listed" the files actually placed the files before us: Any further operation on the files would take place upon the very objects delivered by the directory listing command. This would provide the feeling of directly manipulating the objects that were returned.

When we deal with the conceptually complex objects of computation, we have to remember that each object typically has a number of different attributes. An object can be viewed from different "perspectives," behaving quite differently with each viewpoint.

Consider the file example. If the objects are the names of the files (or icons containing the names), this depiction will work well as long as the operations to be performed are things like removing the files, renaming them, or positioning them as sources of input or receivers of output. But it is not a good representation for all possible operations. Changing the contents of the file, for instance, requires a representation of the contents as opposed to its "surface" features: The objects to be manipulated are the words or images within the file itself. This requires a different representation. Or consider what is required to change the properties of the file—its protection status, perhaps. Here we require a representation that makes visible and manipulable these properties of the file. Thus, at least three different representations are required. It is unlikely and probably inappropriate to try to develop a single representation of an object that can serve all purposes.

We should recognize that more than one view of an object is often necessary and provide a separate representation for each one. Even a system that gives us the illusion that we are manipulating the objects directly needs different representations when the objects are viewed from different perspectives.

The point is that when an interface presents a world of action rather than a language of description, manipulating a representation can have the same effects and the same feel as manipulating the thing being represented. The members of the audience of a well-staged play willfully suspend their beliefs that the players are actors and become directly engaged in the content of the drama. In a similar way, the user of a well designed model world interface can willfully suspend belief that the objects depicted are artifacts of some program and can thereby directly engage the world of the objects. This is the essence of the "first-personness" feeling of direct engagement. Let us now return to the issue of distance and explore the ways that an interface can be direct or indirect with respect to a particular task.

**TWO FORMS OF DISTANCE: SEMANTIC AND ARTICULATORY**

Whenever we interact with a device, we are using an interface language. That is, we must use a language to describe to the device the nature of the actions we wish to have performed. This is true regardless of whether we are dealing with an interface based on the conversation metaphor or on the model world metaphor, although the properties of the language in the two cases are different. A description of desired actions is an expression in the interface language.

The notion of an interface language is not confined to the everyday meaning of language. Setting a switch or turning a steering wheel can be expressions in an interface languages if switch-setting or wheel-turning are how one specifies the operations that are to be done. After an action has been performed, evaluation of the outcome requires that the device make available some indication of what has happened: That output is an expression in the output interface language. Output interface languages are often impoverished. Frequently the output interface language does not share vocabulary with the input interface language. Two forms of interface language—two dialects, if you will—must exist to span the gulfs between user and device: the input interface language and the output interface language.

Both the languages people speak and computer programming languages are almost entirely symbolic in the sense that there is an arbitrary relationship between the form of a vocabulary item and its meaning. The reference relationship is established by convention and must be learned. There is no way to infer meaning from form for most vocabulary items. Because of the relative independence of meaning and form we describe separately two properties of the interface language: Semantic Directness and Articulatory Directness.
Figure 5.5 summarizes the relationship between semantic and articulatory distance. We now examine semantic and articulatory directness separately for the Gulfs of Execution and Evaluation.

Semantic Directness
Semantic directness concerns the relation of the meaning of an expression in the interface language to what the user wants to say. Two important questions about semantic directness are:

- **Is it possible to say what one wants to say in this language?** That is, does the language support the user's conception of the task domain? Does it encode the concepts and distinctions in the domain in the same way that the user thinks about them?

- **Can the things of interest be said concisely?** Can the user say what is wanted in a straightforward fashion, or must the user construct a complicated expression to do what appears in the user's thoughts as a conceptually simple piece of work?

Semantic directness is an issue with all languages. Natural languages generally evolve such that they have rich vocabularies for domains that are of importance to their speakers. When a person learns a new language—especially when the language is from a novel culture—the new language may seem indirect, requiring complicated constructs to describe things the learner thinks should be easy to say. But the differences in apparent directness reflect differences in what things are thought important in the two cultures. Natural languages can and do change as the need arises. This occurs through the introduction of new vocabulary or by changing the meaning of existing terms. The result is to make the language semantically more direct with respect to the topic of interest.

Semantic Distance in the Gulfs of Execution and Evaluation

_Beware the Turing tar-pit in which everything is possible but nothing of interest is easy_ (Alan Perlis, 1982).

The Gulf of Execution. At the highest level of description, a task may be described by the user's intention: "compose this piece" or "design this building." At the lowest level of description, the performance of the task consists of the shuffling of bits around inside the machine. Between the interface and the low-level operations of the machine is the system-provided task-support structure that implements the expressions in the interface language. The situation that Perlis called the "Turing tar-pit" is one in which the interface language lies near or at the level of bit shuffling of a very simple abstract machine. In this case, the entire burden of spanning the gulf from user intention to bit manipulation is carried by the user. The relationship between the user's intention and the organization of the instructions given the machine is distant, complicated, and hard to follow. Where the machine is of minimal complexity, as is the case with the Turing machine example, the wide gulf between user intention and machine instructions must be filled by the user's extensive planning and translation activities. These activities are difficult and rife with opportunities for error.

Semantic directness requires matching the level of description required by the interface language to the level at which the person thinks of the task. It is always the case that the user must generate some information-processing structure to span the gulf. Semantic distance in the gulf of execution reflects how much of the required structure is provided by the system and how much by the user. The more that the user must provide, the greater the distance to be bridged.
The Gulf of Evaluation. On the evaluation side, semantic distance refers to the amount of processing structure that is required for the user to determine whether the goal has been achieved. If the terms of the output are not those of the user's intention, the user will be required to translate the output into terms that are compatible with the intention in order to make the evaluation. For example, suppose a user's intent is to control how fast the water level in a tank rises. The user performs some controlling action and observes the output. But if the output only shows the current value, the user must observe the value over time and mentally compare the values at different times to see what the rate of change is. See Figure 5.6A. The information needed for the evaluation is in the output, but it is not there in a form that directly fits the terms of the evaluation. The burden is on the user to perform the required transformations and that requires effort. Suppose the rate of change were directly displayed, as in Figure 5.6B. This indication reduces the mental workload, making the semantic distance between intentions and output language much shorter.

Reducing the Semantic Distance That Must Be Spanned

Figure 5.6 provides one illustration of how semantic distance can be changed. In general, there are only two basic ways to reduce the distance, one from the system side (requiring effort on the part of the system designer), the other from the user side (requiring effort on the part of the user). Each direction of bridge building has several components. Here let us consider the following possibilities:

- The designer can construct higher-order and specialized languages that move toward the user, making the semantics of the input and output languages match that of the user;
- The user can develop competence by building new mental structures to bridge the gulf. In particular, this requires the user to automate the response sequence and to learn to think in the same language as that required by the system.

Higher-level languages. One way to bridge the gulf between the intentions of the user and the specifications required by the computer is well-known: Provide the user with a higher-order language, one that directly expresses frequently encountered structures of problem decomposition. Instead of requiring the complete decomposition of the task all the way to low-level operations, let the task be described in the same language used within the task domain itself. Although the computer still requires low-level specification, the job of translating from the domain language to the programming language can be taken over by the machine itself.

This implies that designers of higher-level languages should consider how to develop interface languages for which it will be easy for the user to create the mediating structure between intentions and expressions in the language. One way to facilitate this process is to provide consistency across the interface surface. That is, if the user builds a structure to make contact with some part of the interface surface, a savings in effort can be realized if it is possible to use all or part of that same structure to make contact with other areas.

Beware the over-specialized system where operations are easy, but little of interest is possible (the converse of the Turing tar-pit).

The result of matching a language to the task domain brings both good news and bad news. The good news is that tasks are easier to specify. Even if considerable planning is still required to express a task in a high-level language (Riley & O'Malley 1984; Riley, Chapter 7), the amount of planning and translation that can be avoided by the user and passed off to the machine is enormous. The bad news is that the language has lost generality. Tasks that do not easily decompose into
the terms of the language may be difficult or impossible to represent. In the extreme case, what can be done is easy to do, but outside that specialized domain, nothing can be done. The power of a specialized language system derives from carefully specified primitive operations, selected to match the predicted needs of the user, thus capturing frequently occurring structures of problem decomposition.

The trouble is that there is a conflict between generality and matching to any specific problem domain. Some high-level languages and operating systems have striven to close the gap between user intention and the interaction language while preserving freedom and ease of general expression by allowing for extensibility of the language or operating system. Such systems allow the users to move the interface closer to their conception of the task.

The Lisp language and the UNIX operating system serve as examples of this phenomenon. Lisp is a general purpose language, but one that has extended itself to match a number of special, high-level domains. As a result, Lisp can be thought of as a system rather than a language, with numerous levels layered on top of the underlying language kernel. There is a cost to this method. As more and more specialized domain levels get added, the language system gets larger and larger, becoming more clumsy to use, more expensive to support, and more difficult to learn. Just look at any of the manuals for the large Lisp systems (Interlisp, Zetalisp) to get a feel for the complexity involved. The same is true for UNIX operating system which started out with a number of low-level, general primitive operations. Users were allowed (and encouraged) to add their own, more specialized operations, or to package together the primitives into higher-level operations. The results in all these cases are massive systems that are hard to learn and that require a large amount of support facilities. The documentation becomes huge, and not even system experts know all that is present. Moreover, the difficulty of maintaining such a large system increases the burden on everyone, and the possibility of having standard interfaces to each specialized function has long been given up.

The point is that as the interface approaches the user’s intention end of the gulf, functions become more complicated and more specialized in purpose. Because of the incredible variety of human intentions, the lexicon of a language that aspires to both generality of coverage and domain specific functions can grow very large. In any of the modern dialects of Lisp one sees a microcosm of the argument about high-level languages in general. The fundamentals of the language are simple, but a great deal of effort is required to do anything useful at the low level of the language itself. Higher-level functions written in terms of lower-level ones make the system easier to use when the functions match intentions, but in doing so they may restrict possibilities, proliferate vocabulary, and require that a user know an increasing amount about the language of interaction rather than the domain of action.

**Make the output show semantic concepts directly.** An example of reducing semantic distance on the output side is provided by the scenario of controlling the rate of filling a water tank, described in Figure 5.6. In that situation, the output display was modified to show rate of flow directly, something normally not displayed but instead left to the user to compute mentally.

In similar fashion, the change from line-oriented text editors to screen-oriented text editors, where the effects of editing commands could be seen instantly, is another example of matching the display to the user’s semantics. In general, the development of WYSIWYG systems ("What you see is what you get") provides other examples. And finally, the spreadsheet program has been so important, in part because its output format continually and elegantly shows the state of the system in the very terms the accountant likes to use, at least so some believe. This format is actually not so good for many users, which explains the rapid proliferation of graphical packages that transform spreadsheet data into a form more readily interpretable by a wider class of user.

The attempt to develop good semantic matches at the system output confronts the same conflict between generality and power faced in the design of input languages. If the system is too specific and specialized, the output displays lack generality. If the system is too rich, the user has trouble learning and selecting among the possibilities. One solution for both the output and input problem is to abandon hope of maintaining general computing and output ability and to develop special purpose systems for particular domains or tasks. In such a world, the location of the interface in semantic space is pushed closer to the domain language description. Here, things of interest are made simple because the lexicon of the interface language maps well into the lexicon of domain description. Considerable planning may still go on in the conception of the domain itself, but little or no planning or translation is required to get from the language of domain description to the language of the interface. The price paid for these advantages is a loss of generality: Many things are unnatural or even impossible.
Automated behavior does not reduce semantic distance. Cognitive effort is required to plan a sequence of actions to satisfy some intent. Generally, the more structure required of the user, the more effort use of the system will entail. However, this gap can be overcome if the users become familiar enough with the system. Structures that are used frequently need not be rebuilt every time they are needed if they have been remembered. Thus, a user may remember how to do something rather than having to rederive how to do it. It is well known that when tasks are practiced sufficiently often, they become automated, requiring little or no conscious activity. As a result, over time, the use of the interface to solve a particular set of problems will feel less difficult and more direct. Experienced users will sometimes argue that the interface they use directly satisfies their intentions, even when less skilled users complain of the complexity of the structures. To the skilled user, the interface feels direct because the invocation of mediating structure has been automated. They have learned how to transform frequently arising intentions into action specifications. The result is a feeling of directness as compelling as that which results from semantic directness. Automatization is useful, for it improves the interaction of the user with the system, but the feeling of directness it produces depends only on how much practice the particular user has. It gives the system credit for the work the user has done. Although we need to remember that this happens, that users may adjust themselves to the interface and, with sufficient practice, may view it as directly supporting their intentions, we need to distinguish between the cases in which the feeling of directness originates from a close semantic coupling between intentions and the interface language and that which originates from practice. The resultant feeling might be the same in the two cases, but there are crucial differences between how the feeling is acquired and what one needs to do as an interface designer to generate it.

The user can adapt to the system representation. One way to span the gulf is for the users to change their own conceptualization of the problem so that they come to think of it in the same terms as the system. In some sense, this means that the gulf is bridged by moving the user closer to the system. Because of their experience with the system, the users change both their understanding of the task and the language with which they think of the issues. This is related to the notion of linguistic determinism. If it is true that the way we think about something is shaped by the vocabulary we have for talking about it, then it is important for the designer of a system to provide the user with a good representation of the task domain in question. The interface language should provide a powerful, productive way of thinking about the domain.

The point was made forcibly to us in a class in which these ideas about Direct Manipulation were being described. Erasing a word by using an eraser was defined as direct action. Erasing by appropriate marking with a mouse pointing device in a text editor was also called direct, although slightly less. But using a standard, command-language-based text editor was not direct. "But why not?" queried a student. "I am an expert user of vi, and when I wish to delete a word, all I do is think 'delete that word,' my fingers automatically type 'dw,' and the word disappears from the screen. How could anything be more direct?"

The frequent use of even a poorly designed interface can sometimes result in a feeling of directness like that produced by a semantically direct interface. A user can compensate for the deficiencies of the interface through continual use and practice so that the ability to use it becomes automated, requiring little conscious activity. While automation is one factor which can contribute to a feeling of directness, it is essential for an interface designer to distinguish it from semantic directness. Automatization does not reduce the semantic distance that must be spanned. The gulfs between a user's intentions and the interface must still be bridged by the user. Although practice and the resulting expertise can make the crossing less difficult, it does not reduce the magnitude of the gulfs. Planning activity may be replaced by a single memory retrieval so that instead of figuring out what to do, the user remembers what to do. Automatization may feel like direct control, but it comes about for completely different reasons than semantic directness. Automatization is useful, for it improves the interaction of the user with the system, but the feeling of directness it produces depends only on how much practice the particular user has. It gives the system credit for the work the user has done. Although we need to remember that this happens, that users may adjust themselves to the interface and, with sufficient practice, may view it as directly supporting their intentions, we need to distinguish between the cases in which the feeling of directness originates from a close semantic coupling between intentions and the interface language and that which originates from practice. The resultant feeling might be the same in the two cases, but there are crucial differences between how the feeling is acquired and what one needs to do as an interface designer to generate it.

This is the Whorfian hypothesis of language, transformed to the computer setting. The basic idea is that dedicated users change the very language and manner with which they think about problems, so that the initial goals and intentions are formed in the same language required by the system, thus bridging the gulf from the user side. One has only to listen to devoted system users to be convinced of the validity of the concept: Indeed, the actions available on the computer system have been so thoroughly introduced into the culture of the
user, that the concepts are used in non-computer applications as well, where they are no longer directly applicable:

“What movies are playing in town? Grep the newspaper, would you?”

“Both those ideas are good: Let’s cons them up.”

Linguistic determinism takes place at a more fundamental level than the other ways of reducing the semantic distance. While moving the interface closer to the user’s intentions may make it difficult to realize some intentions, changing the user’s conception of the domain may prevent some intentions from arising at all. So while a well designed special purpose language may give the user a powerful way of thinking about the domain, it may also restrict the user’s flexibility to think about the domain in different ways.

That the user may change conceptual structure to match the interface language follows from the notion that every interface language implies a representation of the tasks it is applied to. The representation implied by an interface is not always a coherent one. Some interfaces provide a collection of partially overlapping views of a task domain. If the user is to move toward the model implied by the interface, and thus reduce the semantic distance, that model should be coherent and consistent over some conception of the domain. There is, of course, a tradeoff here between the costs to the user of learning a new way to think about a domain and the potential added power of thinking about it in the new way.

Virtuosity and semantic distance. Sometimes users have a conception of a task and of a system that is broader and more powerful than that provided by an interface. The structures they build to make contact with the interface go beyond it. This is how we characterize virtuoso performances in which the user may “misuse” limited interface tools to satisfy intentions that even the system designer never anticipated. In such cases of virtuosity the notion of semantic distance becomes more complicated and we need to look very carefully at the task that is being accomplished. Semantic directness always involves the relationship between the task one wishes to accomplish and the ways the interfaces provides for accomplishing it. If the task changes, then the semantic directness of the interface may also change.

Consider a musical example: Take the task of producing a middle-C note on two musical instruments, a piano and a violin. For this simple task, the piano provides the more direct interface because one must place the bow on the G string, place a choice of fingers in precisely the right location on that string, and draw the bow. A piano’s keyboard is more semantically direct than the violin’s strings and bow for the task of producing notes. The piano has a single well-defined vocabulary item for each of the notes within its range, while the violin has an infinity of vocabulary items many of which do not produce proper notes at all. However, when the task is playing a musical piece well rather than simply producing notes, the directness of the interfaces can change. In this case, one can complain that a piano has a very indirect interface because it is a machine with which the performer “throws hammers at strings.” The performer has no direct contact with the components that actually produce the sound and so, the production of desired nuances in sound is more difficult. Here, as musical virtuosity develops, the task that is to be accomplished also changes from just the production of notes to concern for how to control more subtle characteristics of the sounds like vibrato, the slight changes in pitch used to add expressiveness. For this task the violin provides a semantically more direct interface than the piano. Thus, as we have argued earlier, an analysis of the nature of the task being performed is essential in determining the semantic directness of an interface.

We are reminded of a quote from Minsky:

“A computer is like a violin. You can imagine a novice trying first a phonograph and then a violin. The latter, he says, sounds terrible. That is the argument heard from our humanists and most of our computer scientists. Computer programs are good they say, for particular purposes, but they aren’t flexible. Neither is a violin, or a typewriter, until you learn how to use it.” (Minsky, 1967)

Articulatory Directness

In addition to its meaning, every vocabulary item in every language has a physical form and that form has an internal structure. Words in natural languages, for example, have phonetic structure when spoken and typographic structure when printed. Similarly, the vocabulary items that constitute an interface language have a physical structure. Where semantic directness has to do with the relationships between user’s intentions and meanings of expressions, articulatory directness has to do with the relationships between the meanings of expressions and their
physical form. On the input side, the form may be a sequence of character-selecting key presses for a command language interface, the movement of a mouse and the associated "mouse clicks" in a pointing device interface, or a phonetic string in a speech interface. On the output side, the form might be a string of characters, a change in an iconic shape, an auditory signal, or a graph, diagram, or animation.

There are ways to design languages such that the relationships between the forms of the vocabulary items and their meanings are not arbitrary. One technique is to make the physical form of the vocabulary items structurally similar to their meanings. In spoken language this relationship is called onomatopoeia. Onomatopoetic words in spoken language refer to their meanings by imitating the sound they refer to. Thus we talk about the "boom" of explosions or the "cock-a-doodle-doo" of roosters. There is an economy here in that the user's knowledge of the structure of the surface acoustical form has a non-arbitrary relation to meaning. There is a directness of reference in this imitation. An intervening level of arbitrary symbolic relations is eliminated. Other uses of language exploit this effect partially. Thus, while the word "long" is arbitrarily associated with its meaning, sentences like "She stayed a looooooooooong time" exploit a structural similarity between the surface form of "long" (whether written or spoken) and the intended meaning. The same sorts of things can be done in the design of interface languages.

In many ways, the interface languages should have an easier time of exploiting articulatory similarity than do natural languages because of the rich technological base available to them. Thus, if the intent is to draw a diagram, the interface might accept as input the drawing motions. In turn, it could present as output diagrams, graphs, and images. If one is talking about sound patterns to the input interface language, the output could be the sounds themselves. The computer has the potential to exploit articulatory similarities through technological innovation in the varieties of dimensions upon which it can operate. This potential has not been exploited, in part because of economic constraints. The restriction to simple keyboard input limits the form and structure of the input languages and the restriction to simple, alphanumeric terminals with small, low resolution screens, limits the form and structure of the output languages.

Articulatory Distance in the Gulfs of Execution and Evaluation

The relationships among semantic distance, articulatory distance, and the gulfs of execution and evaluation are shown in Figure 5.7.

Take the simple, commonplace activity of moving a cursor on the screen. If we do this by moving a mouse, pointing with the finger or a light pen at the screen, or otherwise mimicking the desired motion, then at the level of action execution, these are all articulatorily direct interactions. The meaning of the intention is cursor movement and the action is specified by means of a similar movement. One way to achieve articulatory directness at the input side is to provide an interface that permits specification of an action by mimicking it, thus supporting a articulatorily similarity between the vocabulary item and its meaning. Any nonarbitrary relationship between the form of an item and its meaning can be a basis for articulatory directness. Where it is not possible to create a nonarbitrary relationship between the form of the item and its meaning, we recognize that not all arbitrary
relationships are equally easy to learn. It may be possible to exploit previous user knowledge in creating this relationship. Much of the work on command names in command language interfaces is an instance of trying to develop memorable and discriminable arbitrary relationships between the forms and the meanings of command names.

Articulatory directness at the output side is similar. If the user is following the changes in some variable, a moving graphical display can provide articulatory directness. A table of numbers, although containing the same semantic information, is not articulatorily direct. Thus, the graphical display and the table of numbers might both be equal in semantic directness, but unequal in articulatory directness. Articulatorily direct interaction can couple the interaction between action and meaning so naturally that relationships between intentions and actions and between actions and output seem straightforward and obvious.

Our students have pointed out that although it is indeed mimetically and articulatorily direct to cause cursor movement by an analogous movement of a pointing device, in many instances the truly direct interface will do away with cursors. That is, cursor movements reflect current technological limitations.

If we expand our imaginations, we can develop interface technologies that are even more direct than pointing movements. Thus, to delete a word, why move a cursor to the spot: Simply look at the word and speak the appropriate command.

In general, articulatory directness is highly dependent upon I/O technology. To make actions and displays articulatorily direct requires a much richer set of technological input/output devices than most systems currently have. We will, of course, need to have keyboards, pointing devices, and high-resolution, bit-mapped screens (which not all machines or terminals have today). The mouse is a spatio-mimetic device. That means that it can provide articulatorily direct input for tasks that can be represented spatially. The mouse is useful for a wide variety of tasks not because of any properties inherent in itself, but because we map so many kinds of relationships (even ones that are not intrinsically spatial) on to a spatial metaphor. In addition, we need sound and speech, certainly as outputs, and possibly as inputs. Precise control of timing will be necessary for those applications where the domain of interest is time-sensitive. In general, we will want to open our imaginations to the set of relevant technologies. Perhaps it is suggested and carried out a set of experiments on doing arithmetic by sense of smell (Galton, 1894). Less fancifully conceived, input might be sensitive not only to touch, place, and timing, but also to pressure or to torque. (See Chapter 15 by Buxton.)

Articulatory distance can also be manipulated by the mental model adopted by the user. An example of this comes from the following story, told us by Yutaka Sayeki. His motorcycle had a switch on the left handlebar for controlling the turn signals: Moving the switch forward signaled a right turn, backward a left turn. The switch control was semantically direct in that it had a single unambiguous item for each relevant intention regarding the turn signals, but Sayeki's understandings of the forms of the actions encouraged him to map the action "push switch forward" onto the intention "turn left," which is wrong. As a result, it was difficult to remember which switch direction was associated with which direction of turn. A mimetically direct switch would have moved to the right for the right turn, to the left for the left turn. Sayeki solved the problem through the creation of an appropriate mental model, in part by realizing that the motorcycle switch motion was analogous to the required movement of the turn signal lever in an automobile: The required direction is parallel to the direction of movement of the handlebar (or steering wheel).

Consider the way in which the handlebars of the motorcycle turn. When making a left turn, the left handlebar moves backwards. For a right turn, the left handlebar moves forward. The turn switch was located on the left handlebar, and the required switch movements exactly paralleled the handlebar movements. By reconceptualizing the task as signaling the direction of motion of the handlebars, the switch suddenly became mimetically, articulatorily direct: The switch movement mimics the desired movement. With one mental model, the motion of the switch seems arbitrary, indirect, and difficult to remember. With a different mental model, the switch motion is articulatorily direct and, therefore, easy to learn and to use.
Iconographic languages. Pictographic and iconic languages are examples of articulatory representation in which the form of the expression is related to its meaning. By definition, an icon is a representation that stands for its object by virtue of a resemblance to it. However, even when the form of the icon is very like its intended meaning, the mapping is not complete. Instead, certain features of the referent are abstracted and preserved in the form of the icon, while others are discarded. And even those features that are preserved may be established by convention. For example, while the international iconic symbols for male and female (see illustrations) may seem imitative of a fundamental distinction, they are scarcely interpretable to societies in which trousers and skirts are not worn. Even the interpretation of widely recognized icons requires background knowledge of conventions. Most icons of necessity are abstractions of the thing they depict. Thus, the articulatory directness of even iconic or pictographic representations is not complete: Their interpretation requires knowledge or explanation.

DIRECT ENGAGEMENT

Direct Engagement occurs when a user experiences direct interaction with the objects in a domain. Here, there is a feeling of involvement directly with a world of objects rather than of communicating with an intermediary. The interactions are much like interacting with objects in the physical world. Actions apply to the objects, observations are made directly upon those objects, and the interface and the computer become invisible. Although we believe this feeling of direct engagement to be of critical importance, in fact, we know little about the actual requirements for producing it. Laurel (Chapter 4), discusses some of the requirements. At a minimum, to produce a feeling of direct engagement the system needs:

- Execution and evaluation to be direct in the senses discussed in this chapter.
- Input and output languages of the interface to be interreferential, allowing an input expression to incorporate or make use of a previous output expression. This is crucial for creating the illusion that one is directly manipulating the objects of concern.
- The system to be responsive, with no delays between execution and the results, except where those delays are appropriate for the knowledge domain itself.
- The interface to be unobtrusive, not interfering or intruding. If the interface itself is noticed, then it stands in a third-person relationship to the objects of interest, and detracts from the directness of the engagement.

In order to have a feeling of direct engagement, the interface must provide the user with a world in which to interact. The objects of that world must feel like they are the objects of interest, that one is doing things with them and watching how they react. In order for this to be the case, the output language must present representations of objects in forms that behave in the way that the user thinks of the objects behaving. Whatever changes are caused in the objects by the set of operations must be depicted in the representation of the objects. This use of the same object as both an input and output entity is essential to providing objects that behave as if they are the real thing. It is because an input expression can contain a previous output expression that the user feels the output expression is the thing itself and that the operation is applied directly to the thing itself. This is exactly the concept of "interreferential I/O" discussed by Draper (Chapter 16).

In addition, all of the discussion of semantic and articulatory directness apply here too, because the designer of the interface must be concerned with what is to be done and how one articulates that in the languages of interaction. But the designer must also be concerned with creating and supporting an illusion. The specification of what needs to be done and evidence that it has been done must not violate the illusion, else the feeling of direct engagement will be lost.
One factor that seems especially relevant to maintaining this illusion is the form and speed of feedback. Rapid feedback in terms of changes in the behavior of objects not only allows for the modification of actions even as they are being executed, but also supports the feeling of acting directly on the objects themselves. It removes the perception of the computer as an intermediary by providing continual representation of system state. In addition, rapidity of feedback and continuous representation of state allows one to make use of perceptual faculties in evaluating the outcome of actions. We can watch the actions take place, monitoring them much like we monitor our interactions with the physical world. The reduction in the cognitive load of mentally maintaining relevant information and the form of the interaction contribute to the feeling of engagement.

A SPACE OF INTERFACES

Distance and engagement are depicted in Figure 5.8 as two major dimensions in a space of interface designs. The dimension of engagement has two landmark values: One is the metaphor of interface as conversation; the other the metaphor of interface as model world. The dimension of distance actually contains two distances to be spanned: semantic and articulatory distances, the two kinds of gulf that lie between the user's conception of the task and the interface language.

The least direct interface is often one that provides a low-level language interface, for this is apt to provide the weakest semantic match between intentions and the language of the interface. In this case, the interface is an intermediary between the user and the task. Worse, it is an intermediary that does not understand actions at the level of description in which the user likes to think of them. Here the user must translate intentions into complex or lengthy expressions in the language that the interface intermediary can understand.

A more direct situation arises when the central metaphor of the interface is a world. Then the user can be directly engaged with the objects in a world, but still, if the actions in that world do not match those that the user wishes to perform within the task domain, getting the task done may be a difficult process. The user may believe that things are getting done and may even experience a sense of engagement with the world, yet still be doing things at too low a level. This is the state of some of the recently introduced direct manipulation systems: They produce an immediate sense of engagement, but as the user develops experience with the system, the interface appears clumsy, to interfere too much, and to demand too many actions and decisions at the wrong level of specification. These interfaces appear on the surface to be Direct Manipulation interfaces, but they fail to produce the proper feelings of Direct Engagement with the task world. Beginners tend to like them: Opinions among experts vary.

Closing the distance between the user's intentions and the level of specification of the interface language allows the user to make efficient specifications of intentions. Where this is done with a high level language, quite efficient interfaces can be designed. This is the situation in most modern integrated programming environments. For some classes of tasks, such interfaces may be superior to direct manipulation interfaces.

Finally, the most direct of the interfaces will lie where engagement is maximized, where just the correct semantic and articulatory matches are provided, and where all distances are minimized.
PROBLEMS WITH DIRECT MANIPULATION

Direct Manipulation systems have both virtues and vices. The immediacy of feedback and the natural translation of intentions to actions make some tasks easy. The matching of levels of thought to the interface language—semantic directness—increases the ease and power of performing some activities at a potential cost of generality and flexibility. Not all things should be done directly. Thus, a repetitive operation is probably best done via a script, that is, through a symbolic description of the tasks that are to be accomplished. Direct Manipulation interfaces have difficulty handling variables, or distinguishing the depiction of an individual element from a representation of a set or class of elements. Direct Manipulation interfaces have problems with accuracy, for the notion of mimetic action puts the responsibility on the user to control actions with precision, a responsibility that is often best handled through the intelligence of the system, and sometimes best communicated symbolically.

A more fundamental problem with direct manipulation interfaces arises from the fact that much of the appeal and power of this form of interface comes from their ability to directly support the way we normally think about a domain. They amplify our knowledge of the domain and allow us to think in the familiar terms of the application domain rather than those of the medium of computation. But if we restrict ourselves to only building interfaces that allow us to do things we can already do and to think in ways we already think, we will miss the most exciting potential of new technology: to provide new ways to think of and to interact with a domain. Providing these new ways and creating conditions that will make them feel direct and natural is an important challenge to the interface designer.

Direct manipulation interfaces are not a panacea. Although with sufficient practice many interfaces can come to feel direct, a properly designed interface, one which exploits semantic and articulatory directness, should decrease the amount of learning required and provide a natural mapping to the task. But interface design is subject to many tradeoffs. There are surely instances when one might wisely trade off directness for generality, or for more facile ways of saying abstract things. The articulatory directness involved in pointing at objects might need to be traded off against the difficulties of moving the hands between input devices or of problems in pointing with great precision.

It is important not to equate directness with ease of use. Indeed, if the interface is really invisible, then the difficulties within the task domain get transferred directly into difficulties for the user. Suppose the user struggles to formulate an intention because of lack of knowledge of the task domain. The user may complain that the system is difficult to use. But the difficulty is in the task domain, not in the interface language. Direct Manipulation interfaces do not pretend to assist in overcoming problems that result from poor understanding of the task domain.

Reassessing the Claims for Direct Manipulation

What about the claims for direct manipulation? Do these systems really meet the hopes? Alas, it is too early to tell. We believe that direct manipulation systems carry gains in ease of learning and ease of use. If the mapping is done correctly, then both the form and the meaning of commands should be easier to acquire and retain. Interpretation of the output should be immediate and straightforward. If the interface is a model of the task domain, then one could have the feeling of directly engaging the problem of interest itself. It is sometimes said that in such situations the interface disappears. It is probably more revealing to say that the interface is no longer recognized as an interface. Instead, it is taken to be the task domain itself.

But are these desirable features? Are the tradeoffs too costly? As always, we are sure that the answer will depend on the tasks to be accomplished. Certain kinds of abstraction that are easy to deal with in language seem difficult in a concrete model of a task domain. When we give up the conversation metaphor, we also give up dealing in descriptions, and in some contexts, there is great power in descriptions. As an interface to a programming task, direct manipulation interfaces are problematic. We know of no really useful direct manipulation programming environments. Issues such as controlling the scope of variable bindings promise to be quite tricky in the direct manipulation environments. Will Direct Manipulation systems live up to their promise? Yes and no. Basically, the systems will be good and powerful for some purposes, poor and weak for others. In the end, many things done today will be replaced by Direct Manipulation systems. But we will still have conventional programming languages.

On the surface, the fundamental idea of a Direct Manipulation interface to a task flies in the face of two thousand years of development of abstract formalisms as a means of understanding and controlling the world. Until very recently, the use of computers has been an activity squarely in that tradition. So the exterior of Direct Manipulation, providing as it does for the direct control of a specific task world, seems somehow atavistic, a return to concrete thinking. On the inside, of course, the implementation of direct manipulation systems is yet another step in that long formal tradition. The illusion of the
absolutely manipulable concrete world is made possible by the technology of abstraction.

Earlier in this chapter we reprinted two sets of descriptions of the virtues of direct manipulation systems, claims made in the early enthusiasm of their first discovery. Now that we have examined the many aspects of directness, it is time to go back and re-evaluate those claims. Let us see what we can say about them now. First, Shneiderman (1982) pointed at three "essential features" of Direct Manipulation. Let us examine each of the three features: Shneiderman's comments in italics, our assessment in regular font.

1. Continuous representation of the object of interest.

This relates to the directness of evaluation, and as our analyses show, seems an essential aspect of a direct engagement system.

2. Physical actions or labeled button presses instead of complex syntax.

The feature of "physical actions" we interpret to refer to articulatory directness, and more particularly, to what one might call "mimetic directness," actions that mimic the desired changes on the objects of interest, especially movement or size changes. "Labeled button presses" must refer to semantic directness. The former clearly is important in establishing direct engagement. The latter is not. The latter operation really seems irrelevant to arguments about this form of directness, whatever the virtues for ease of learning or ease of use.

3. Rapid incremental reversible operations whose impact on the object of interest is immediately visible.

This is perhaps the essence of direct engagement: It reflects what we and Draper (Chapter 16) call the importance of "interreferential I/O." The important point to us, implicitly stated in this assumption, is that the objects upon which the actions are taken are exactly the same as those upon which evaluation is made. The reversibility of the operations is desirable, but not necessary. Not all operations can have this feature in a natural way. So too with immediacy of the result: Where immediacy is a natural part of the domain, then this is essential. Otherwise, it is desirable, but may not always be necessary—as we have discussed at length within this chapter.

If we reinterpret Shneiderman's claims to be about Direct Engagement, then his three features fare well: They do seem to help define what is necessary to develop a direct engagement system. Now let us look at Shneiderman's six claims for the results of such a system:

1. Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.

This may or may not be true. We think it really derives from the fact that a good Direct Manipulation interface is invisible—the user feels as if operations are actually done directly on the task domain. And if the computer novice is already knowledgeable in the task domain, then much of what is needed to use the interface is already known.

Why might training be possible through demonstration? Well, two reasons. Lewis (Chapter 8) argues that demonstration is, in general, a superior method of instruction. Think of a demonstration as a dynamic example. In this case, then, the superiority of demonstration has nothing to do with direct manipulation. But there is a second reason. Owen (Chapter 17) points out that demonstrations with normal, command language systems, are often puzzling because the actions are not visible. That is, the expert waves hands over the keyboard and mysterious wonderful results appear on the screen. The learner often has little notion of what operation was performed. But with a typical direct manipulation (read direct engagement) system, the actions themselves are visible, and their results are both visible and also direct reflections of the operations done upon them. Whenever these cases apply, we believe the claim will be valid.

2. Experts can work extremely rapidly to carry out a wide range of tasks, even defining new functions and features.

We are suspicious of this claim. In fact, we would not be surprised if experts are slower with Direct Manipulation systems than with command language systems. We suspect that the virtues of Direct Manipulation lie elsewhere: Speed at execution is not likely to be a relevant factor. Real experts can probably type a few lines of obscure code much faster than they could...
move objects around a screen, position them properly, and do the necessary pointing operations.

3. **Knowledgeable intermittent users can retain operational concepts.**

   This could be true, but if so it probably reflects two things: The expertise at usage really reflects expertise in the subject matter, which is probably well established and apt to fade slowly from memory, if at all; and a good semantic mapping of actions leads to slower forgetting and also easier rederivation of the operations that are forgotten. We suspect that these claims are true, but derive from aspects that would be true for many well-designed systems, not just Direct Manipulation systems.

4. **Error messages are rarely needed.**

   Yes and No. Error messages are often not needed because the results are immediately visible, and because some classes of errors may not even be possible. But Direct Manipulation systems have their own problems. It is possible to make new classes of errors, some of them potentially serious. Worse, because these are apt to be errors in the task domain (but legal operations as far as the interface is concerned) they are hard to detect. Finally, Direct Manipulation systems sometimes simply don’t bother with error messages, assuming that the ease of evaluation will make it obvious to the user that the desired operation was not done. It’s not that the message wasn’t needed, it’s just that the system didn’t bother to present one. This what Lewis and Norman call the “do nothing” strategy. This strategy is not always desirable (see Lewis & Norman, Chapter 20).

5. **Users can see immediately if their actions are furthering their goals, and if not, they can simply change the direction of their activity.**

   The first part of the claim results from the properties of Direct Evaluation. The second part has already been discussed in our response to feature 3 of the first list. But basically, the ability to "change the direction of their activity" results from the natural reversibility of many actions. For those actions that are not so naturally reversible, the systems do not fare any differently than more conventional systems. Immediate feedback as to the outcome of an operation and "undo" commands are valuable for any system, not just Direct Manipulation systems.

6. **Users have reduced anxiety because the system is comprehensible and because actions are so easily reversible.**

   This is outside the domain of our analyses and difficult to assess. A fair comparison would require two systems, both equally well-matched in capability and in semantic directness.

   In conclusion, some of the claims seem too strong, some represent features of many well-designed systems, and some seem to be correct. All in all, the early claims seem to fare reasonably well, although today we can bring more sophistication and depth of analysis to bear upon them then was possible at the time they were made.

**CONCLUDING REMARKS**

Direct Manipulation implies a directness of action, a directness of the translation of intentions to actions, and a directness in the feedback and knowledge of the system. The feeling of directness also implies a feeling of control over the objects in the task domain—a Direct Engagement which results, in part, from the feeling that one is operating directly upon the objects, where the objects upon which actions are performed are the very same as those from which output is received.

In this chapter we have concentrated upon the various aspects of directness with respect to the performance of a task, with little mention of the problems of learning. Nonetheless, a system that is properly designed should also be one that is easy to learn. Our discussions of the nature of the Gulfs separating a person’s intentions and evaluations from the system’s inputs and outputs provide a foundation for making the learning task easier. If the Gulfs of Execution and Evaluation are properly bridged from the system side, then the system should be relatively easy to learn: Minimum processing is required to get from intention to action and from outcome to interpretation. The learning required should be concentrated in the task domain. The hope is that people will be able to spend their time learning the task domain, not learning the computer system. Indeed, a properly developed Direct Manipulation interface should appear to the user as if it is the task that is being executed directly: The computer system and its interface will be more or less invisible. Much empirical work remains to be done to substantiate these claims.

The understanding of Direct Manipulation interfaces is complex. There are a host of virtues and vices. Do not be dismayed by the complexity of the arguments. To our mind, direct manipulation interfaces
provide exciting new perspectives on possible modes of interactions with computers. It is our intention to explore these modes further, through the building of working systems, expanding the theoretical analysis, and performing empirical evaluations. We believe that this direction of work has the potential to deliver important new conceptualizations and a new philosophy of interaction.

ACKNOWLEDGMENTS
We thank Ben Shneiderman for his helpful comments on an earlier draft of the chapter, Eileen Conway for her aid with the illustrations, and Julie Norman and Sondra Buffett for extensive editorial comments.

NOTES ON THE FUTURE OF PROGRAMMING: BREAKING THE UTILITY BARRIER

ANDREA A. diSESSA

In preparing this chapter I felt obliged to look at a number of other discussions of the future of programming. Few of them mention anything resembling the issues discussed here. Instead, probably the most prominent theme is the "complexity barrier"—the tremendous difficulty and cost involved in creating and maintaining huge programs. This is, no doubt, a serious and enduring problem. But it is the most serious one for professional programmers, not nonprofessionals who, at least in terms of numbers, will dominate future use of computers. More than numbers, I believe the ultimate social and cultural impact of computation will be determined to a great extent by what we can cause to happen when technologically unsophisticated users sit down at a machine. The hope I share with many others is that computation can significantly enhance intellectual development and productivity for most, if not all, people. (Two exemplary references are Winograd, 1984a, 1984b.)

What role will programming play in this? Some of my favorite antagonists in this regard feel that computers will totally disappear into the woodwork as far as ordinary people are concerned, the way electrical relays and motors and control micro-processors have already disappeared. Even if they don't disappear, certainly, it is said, the ordinary person will need to know as little about programming as about repairing