THE HUMANE INTERFACE

New Directions for Designing Interactive Systems

JEF RASKIN

ADDISON-WESLEY

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

system changes automatically, even if the change is as small as, say, a reordered set of items on a menu, your expectations are upset and your habituation is frustrated. (Microsoft features adaptive menus in its Windows 2000 operating system.[9]) On the other hand, there is no theory that tells us that the same fixed interface cannot work well over the full span of a person's experience with it, from novice to old timer. It seems best not to have to shift paradigms during your use of a product, and no elaborate analysis is needed to reveal the advantage in having to learn only one interface to a task.

It is easy to fall into the trap of designing different interfaces for different classes of users, because by doing so, you can make sweeping assumptions that simplify the design process. Few such assumptions are likely to be true of every user in any reasonably large class of users that you specify. The antidote is to view an interface not from the perspective of a class of users but rather through the eyes of an individual. Every person who uses software over a long period goes through a relatively brief period of learning for each feature or command and a far longer period of routine (and, we hope, automatic) use. We do need to design systems that are easy to learn and understand, but it is more important that we make sure that these systems can be efficiently used in the long run. The exceptions are applications that will be used only briefly, so that every user is a novice, and habituation is not an issue. One example of such an interface is that for a computer-driven kiosk at an exhibition.

The learning phase of working with a feature involves your conscious attention. Therefore, simplicity, clarity of function, and visibility are of great importance. The expert phase is predominantly characterized by unconscious use of the feature; such use is enhanced by such qualities as aptness to the task, modelessness, and monotony. These sets of requirements are not in conflict; therefore, *a well-designed and humane interface does not have to be split into beginner and expert subsystems.*

This is not to say that an interface must not be split on these lines. However, if you find yourself designing an interface and are tempted to provide "expert" shortcuts, consider whether you should instead redesign the existing method so that it satisfies the needs of all users with one mechanism.

9. Windows 2000 was a new product as this section was being written, and I was able to interview only a few users. A typical remark was, "Adaptive menus seemed like a cool idea, but the first time a menu changed on me, I found it upsetting. I don't like the idea any more."

FOUR

Quantification

The harmony of the world is made manifest in Form and Number, and the heart and soul and all the poetry of Natural Philosophy are embodied in the concept of mathematical beauty.

—*D'Arcy Wentworth Thompson,* On Growth and Form *(1917)*

A number of methods for analyzing interface details quantitatively are available. However, explicit directions on how to use them are rare. This chapter gives an easy-to-use treatment and fully worked-out examples of Card, Moran, and Newell's keystroke-level GOMS model, Raskin's measures of efficiency, Hick's Law, and Fitts' law.

4-1 Quantitative Analyses of Interfaces

He fingered and fingered the computer—it simply amazed Melrose that the machine supposed to take the pain out of all sorts of niggling jobs took more time to perform a simple one than it would have taken Bub to do by hand ten times over.

—*Martha Grimes,* The Stargazey *(a detective novel)*

Many qualitative methods and heuristics are useful for analyzing and understanding interface design. These methods form the majority of the content of most books on the subject, including those cited in the references for Shneiderman, Norman, and Mayhew. For example, what an experienced interface designer can learn from passively observing a test of a new interface with a few subjects can be as valuable as what she can learn from any quantitative analysis. My concentration on quantitative methods is not meant to denigrate the importance of qualitative techniques but rather to help even the balance by emphasizing the numerical and empirically testable methods that are not yet widely used. Quantitative methods can often reduce argument to calculation; a further, and most important, benefit is that *understanding* why *the quantitative methods work guides us to understanding important aspects of how humans interact with machines.*

One of the best quantitative analyses of interface design is the classic model of goals, objects, methods, and selection rules (GOMS), which first gained attention in the 1980s (Card, Moran, and Newell 1983). GOMS modeling allows you to predict how long an experienced worker will take to perform a particular operation when using a given interface design. After discussing the GOMS model, I present quantitative methods for determining interface efficiency, cursor movement speed, and the time cost of decision making.

4-2 GOMS Keystroke-Level Model

The aim of exact science is to reduce the problems of nature to the determination of quantities by operations with numbers.

—*James Clerk Maxwell,* On Faraday's Lines of Force *(1856)*

I will introduce only the simplest—yet nonetheless valuable—aspect of the GOMS method: the keystroke-level model. We designers who know GOMS rarely use a detailed and formal analysis of an interface design, but that is due, in part, to our having absorbed the fundamentals of GOMS and of other quantitative methods such that our designs inherently incorporate GOMS teachings. We do bring formal analysis into play when choosing between two approaches to interface design in which small differences in speed can have significant economic or psychological effects. We can sometimes benefit from the impressive accuracy of the more complete GOMS models, such as critical-path method GOMS (CPM-GOMS) or a version called natural GOMS language (NGOMSL), which takes into account nonexpert behavior, such as learning times. We can, for example, predict how long it will take a user to execute a particular set of interface actions to within

an absolute error of less than 5 percent. In these advanced models, almost all predictions fall within 1 standard deviation of the measured times (Gray, John, and Atwood 1993, p. 278). In a field in which religious wars are waged over interface designs and in which gurus often have widely varying opinions, it is advantageous to have in your armamentarium quantitative, experimentally validated, and theoretically sound techniques. For a good overview and bibliography of the various GOMS models, including her own CPM-GOMS model, see John 1995.

4-2-1 Interface Timings

Numerical precision is the very soul of science.

—*D'Arcy Wentworth Thompson,* On Growth and Form *(1917)*

When they developed the GOMS model, its inventors observed that the time it takes the user-computer system to perform a task is the sum of the times it takes for the system to perform the serial elementary gestures that the task comprises. Although different users might have widely varying times, the researchers found that for many *comparative* analyses of tasks involving use of a keyboard and a graphical input device, you could use a set of typical times rather than measuring the times of individuals. By means of careful laboratory experiments, they developed a set of timings for different gestures. In giving the timings, I follow the original nomenclature, in which each of the times is designated by a one-letter mnemonic (Card, Moran, and Newell 1983):

$K = 0.2$ sec	Keying: The time it takes to tap a key on the keyboard
$P = 1.1$ sec	Pointing: The time it takes a user to point to a position on a display
$H = 0.4$ sec	Homing: The time it takes a user's hand to move from the keyboard to the GID or from the GID to the keyboard
$M = 1.35$ sec	Mentally preparing: The time it takes a user to prepare mentally for the next step
R	Responding: The time a user must wait for a computer to respond to input

In practice, these numbers vary widely; K can be 0.08 sec for a 135-wpm highly skilled typist, 0.2 sec for a more typical 55-wpm skilled typist, 0.28 sec for a 40-wpm average unskilled typist, or 1.2 sec for a beginning typist. Typing speed is not independent of what is being typed: It takes most people about 0.5 sec to type a random letter, given a set of randomly chosen letters to type. Typing messy codes—for example, e-mail addresses—takes most people about 0.75 sec per character. The value *K* includes time it takes the user to make corrections that he has caught immediately. Shift is counted as a separate keystroke.

The wide variability of each measure explains why we cannot use this simplified model to obtain absolute timings with any degree of certainty; by using the typical values, however, we usually obtain the correct *ranking* of the performance times of two interface designs. If you are evaluating complex interfaces that include overlapping time dependencies or if you must generate accurate absolute times, you should use the more complete models, which are not discussed in this book, such as CPM-GOMS.

Double Dysclicksia

The interface technique called double clicking, that is, tapping the GID button twice within a small time window and without any significant cursor movement between the taps, as an interface technique suffers from problems. You cannot always predict what objects on the display will or will not respond to a double click, and it is not always clear what will happen if there is a response. There is no indication on displayable items that double clicking is supposed to produce a response: The functionality is invisible. The way that double clicking is used in many current interfaces, the user must remember not only *which* items are double clickable but also how different classes of interface features respond to this action.

The first two burdens on the user could be at least partially alleviated by new screen conventions. The act of double clicking is, however, itself problematical. Double clicking requires operating a mouse button twice at the same location or at two locations in very close and, in most cases, within a short time, typically 500 msec. If the user clicks too slowly, the machine responds to two single clicks rather than to one double click. If the user jiggles the mouse excessively between clicks, the same error occurs. If the user taps the GID button twice in too short a time period, as when trying to select text within a word while working within certain word processors, the machine considers the two taps as a double click and selects the whole word.

A problem arises when the user is trying to select a graphical item that can be repositioned with the GID. Because the GID is likely to move when the user is pressing the GID buttons quickly, graphical applications, instead of reading a double click, may read a drag-and-drop and change the item's position. Similarly, to change the text in a text box, the user may find it necessary to reposition the accidentally moved box and to make the text edit originally intended.

Some of us are unaffected by dysclicksia: These lucky people never miss with the mouse; they single and double click with insouciance and panache, do not suffer from side effects of clicking, always remember what will and what will not respond to double clicking, and can shoot a flying bird with a .357-caliber revolver while driving along a twisty mountain road. But we can't assume that all users are so lucky. We must design for the dysclicksic user and remain aware of the problems inherent in using double clicks in an interface.[1]

The duration of the machine response time, *R*, can have an unexpected effect on user actions. If a user operates a control and nothing appears on the display for more than approximately 250 msec, she is likely to become uneasy, to try again, or to begin to wonder whether the system is failing.

We cannot build products that can complete any operation within human reaction time, but our interfaces can always, within that time, give feedback that the input has been received and recognized. Otherwise, user actions—often flailing at the keyboard, trying to get a response—during a delay can start the system off on unintended activities, causing further delay or damaging the user's content. For example, if you try to download a file while accessing America Online from a browser, such as Netscape's, there is often a long delay. No feedback lets you know that progress is being made; a small, static message far from the locus of attention says only that the computer is awaiting reply. After a few seconds, inexperienced users start clicking at buttons on the display, which stops the download—again without feedback.

It is important that interfaces provide feedback if delays are unavoidable; display a progress bar (Figure 4.1) that accurately reflects the time remaining. If you cannot predict how much time an operation will take, say so! Do not lie to or misinform users.

[1] The term *dysclicksia,* a disease for which the only permanent cure is good design, was coined by Pam Martin (personal communication 1997).

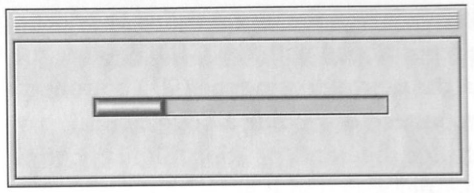


Figure 4.1. A progress bar. It is important that it represent time linearly. A textual statement of time remaining, if accurate, is also a humane feature when delays are unavoidable.

4-2-2 GOMS Calculations

We begin the calculation of the time it takes to perform a method, such as "move your hand from the graphical input device to the keyboard and type a letter," by listing the operations from the GOMS list of gestures (see Section 4-2-1) used in this method, in this case *H K*. Listing the gestures (*K*, *P*, and *H*) is the easy part of creating an instance of GOMS models. The more difficult part of developing an instance of a keystroke-level GOMS model is figuring out at what points the user will stop to perform an unconscious mental operation: the mental preparation (*M*) times. The basic rules—following the methods of Card, Moran, and Newell 1983, p. 265—for deciding where mental operations occur in a method are presented in Table 4.1. In Section 4-2-3, we look at how these rules are applied in practice.

In these rules, a **string** is a sequence of characters. A **delimiter** is a character that marks the beginning or the end of a meaningful string of text, such as a natural-language word or a telephone number. For example, spaces are the delimiters for most words; a period is the most common delimiter at the end of a sentence; parentheses delimit parenthetical remarks; and so on. The operators are *K*, *P*, and *H*. When a command needs information, such as when you use the command that sets the time for an alarm to go off and have to supply the time, the information you supply is an **argument** for that command.

4-2-3 GOMS Calculation Examples

An interface design usually begins with a task or a set of tasks that need to be accomplished. A statement of the task and the means available for implementing a solution are often formulated as a requirement or specification. In this example, the user is personified as Hal, a laboratory assistant.

TABLE 4.1. HEURISTICS FOR PLACING MENTAL OPERATORS

Rule 0 Initial insertion of candidate *Ms*
Insert *Ms* in front of all *Ks* (keystrokes). Place *Ms* in front of all *Ps* (acts of pointing with the GID) that select commands, but do not place *Ms* in front of any *Ps* that point to arguments of those commands.

Rule 1 Deletion of anticipated *Ms*
If an operator following an *M* is fully anticipated in an operator just previous to that *M*, then delete that *M*. For example, if you move the GID with the intent of tapping the GID button when you reach the target of your GID move, then you delete, by this rule, the *M* you inserted as a consequence of rule 0. In this case, *P M K* becomes *P K*.

Rule 2 Deletion of *Ms* within cognitive units
If a string of *M Ks* belongs to a cognitive unit, then delete all the *Ms* but the first. A cognitive unit is a contiguous sequence of typed characters that form a command name or that is required as an argument to a command. For example, *Y*, *move*, *Helen of Troy*, or *4564.23* can be examples of cognitive units.

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

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

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

Requirement
Hal works at a computer, typing reports; he is occasionally interrupted by one or another of the researchers in the room, and is asked to convert a temperature reading from degrees Fahrenheit (F) or Celsius (C) to degrees C or F, respectively. For example, Hal might be asked, "Please convert 302.25 degrees from Fahrenheit to

Celsius." Hal must use the keyboard or GID to enter the temperature provided; voice or other input means are not available. Conversions from C to F and from F to C are approximately equally likely to be required. About 25 percent of the temperatures called out are negative, although the digits are unpredictable and equally distributed, and only 10 percent of the temperatures have integer values, such as 37 degrees. The numerical result must appear on the display; no other output means are available. Hal reads to the researcher the converted value from the screen. The input and the output must allow for at least ten digits on each side of the decimal point.

In designing an interface for a system that allows Hal to do his job, your goal is to minimize the time it takes Hal to do the conversion. Speed and accuracy must be maximized; screen real estate is not limited. The window, or area of the display in which the temperature conversion takes place, is already active and waiting for Hal's input via GID or keyboard. The way Hal interacts with the interface to return to his typing on the computer is not your concern; your job is finished as soon as the result is displayed.

In estimating the time it takes Hal to use the interface, assume an average of four typed characters in an entered temperature, including any decimal point and sign. Also assume—unrealistically, but for simplicity's sake—that Hal's typing is perfect; error detection and notification are not needed.

Now, I would like you to stop reading so that you can design an interface for this simple example. It will not take long to write down your proposed solution, along with sketches of the display that Hal will see; do not just think about this problem but rather write about it as well. (You will be tempted to read on without honoring my request. Please reconsider. The next few sections will make much more interesting reading if you have already tried to solve the problem yourself.) After designing your interface, read the two GOMS analyses that follow. Then you will be ready to analyze your own interface.

4-2-3-1 Hal's Interface: Solution 1, Dialog Box

The instructions in Figure 4.2 are reasonably clear; from them we can write down the method that Hal must use in terms of the gestures of the GOMS model. The GOMS representation is shown growing incrementally as each new gesture is added to the method.

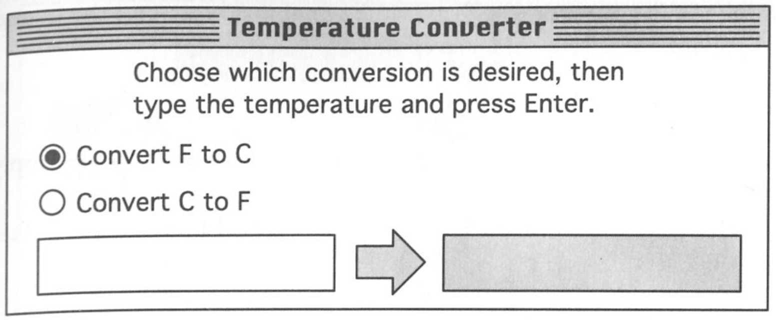

Temperature Converter
Choose which conversion is desired, then type the temperature and press Enter.
Convert F to C
Convert C to F

Figure 4.2. A dialog box solution with radio buttons.

- Move hand to the graphical input device:

  H
- Point to the desired radio button:

  HP
- Click on the radio button:

  HPK

Half of the time, the interface will already have the correct conversion chosen, and Hal will not need to click on the radio button. We consider first the case in which it is not the one already chosen.

- Move hands back to the keyboard:

  HPKH
- Type the four characters:

  HPKHKKKK
- Tap *Enter:*

  HPKHKKKKK

The keystroke for the tap of the Enter key completes the method portion of the analysis. Using rule 0, we add *M*s in front of all of the *K*s and *P*s except those *P*s that point to arguments, of which there are none in this example:

HMPMKHMKMKMKMKMK

Rule 1 tells us to change *P M K* to *P K* and to eliminate any other fully anticipated *M*s, of which there are none in this example. Rule 2 eliminates

Ms in the middle of strings, such as in the string that represents the temperature. Applying these two rules leaves

HMPKHMKKKKMK

The *M* before the final *K* is required by rule 4. Rules 3 and 5 do not apply in this example.

The next step is to add the times represented by the letters. (Recall that $K = 0.2$, $P = 1.1$, $H = 0.4$, and $M = 1.35$):

$$H + M + P + K + H + M + K + K + K + K + M + K =$$

$$0.4 + 1.35 + 1.1 + 0.2 + 0.4 + 1.35 + 4 \star (0.2) + 1.35 + 0.2 = 7.15$$ seconds

In the case in which the correct conversion is already selected, the method is

MKKKKMK

$$M + K + K + K + K + M + K = 3.7 \text{ sec}$$

By the requirements document, these two cases are equally likely. Thus, the average time it will take Hal to use this interface for one conversion task will be $(7.15 + 3.7) / 2 \approx 5.4$ seconds. But, because the two methods that Hal has to use are different, it will be difficult for him to operate this interface automatically. One of the open problems in the quantitative analysis of interfaces is how to estimate error rates from a given interface design.

Next, we explore a graphical interface that makes extensive use of a familiar metaphor.

4-2-3-2 Hal's Interface: Solution 2, GUI

The interface shown in Figure 4.3 uses realistic representations of thermometers to indicate temperature. Hal can lower or raise the pointer on each thermometer in Figure 4.3 by using the drag method with the GID. Hal indicates which conversion he wants by moving the arrow on either the Celsius or the Fahrenheit thermometer. He does not type any characters; he simply selects the temperature on the input thermometer. As he moves one of the pointers, the pointer on the other thermometer moves to the corresponding temperature. To set the required precision, Hal expands and contracts the scales; he can also change the range. When Hal changes the scale or the range on one thermometer, those on the other thermometer change automatically to cover approximately the same set of temperatures. Numerical readouts are provided on the movable arrow. The temperature is indicated both numerically and with a bar, so Hal can use either the graphical or the character-based representations of the data to accommodate his learning style or personal preferences. The Auto-Med feature changes the ranges such that they are centered on 37 degrees Celsius and 98.6 degrees Fahrenheit, in case someone in the lab is working with human body temperatures; this feature is designed to save time.

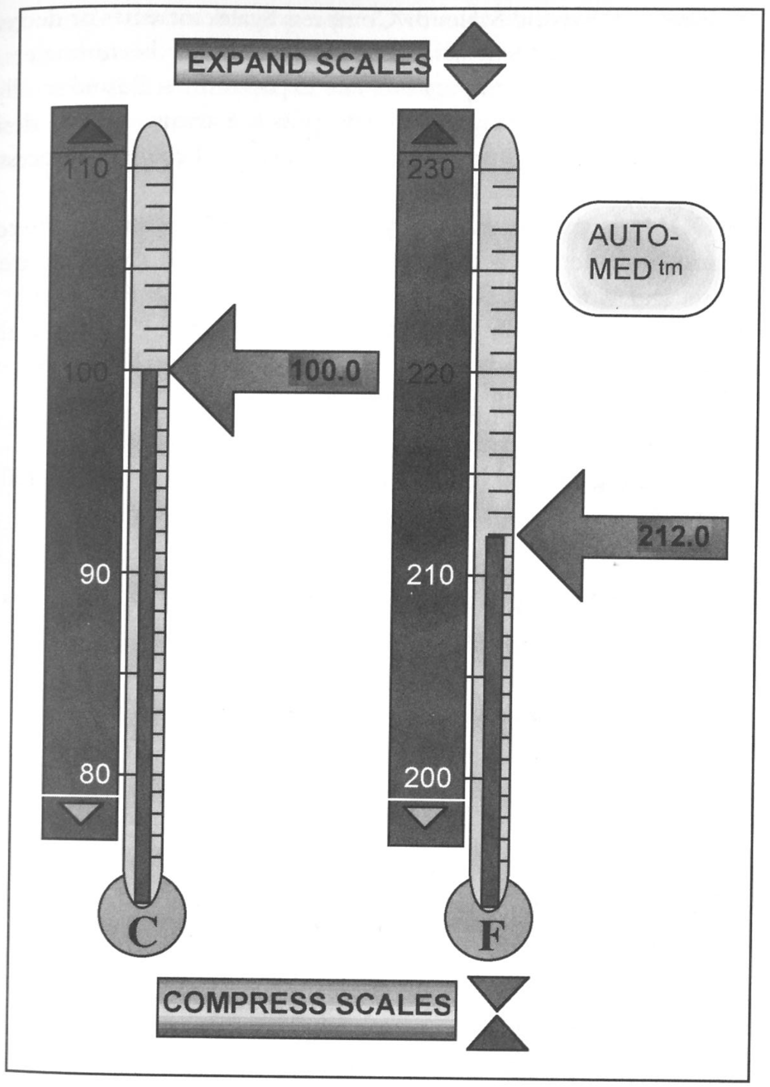


Figure 4.3. A GUI for Hal's interface. (See color insert.)

Clicking on Expand Scales or Compress Scales increases or decreases by a factor of 10 the values at tick marks on the vertical thermometers. To get quickly to a far-distant temperature, Hal expands the scale and scrolls up or down until the desired range is in view, puts the arrow near the desired temperature, and then compresses the scale, adjusting the arrow if necessary, until the desired precision is attained.

A GOMS keystroke-level analysis of this graphical interface is complex because the method Hal uses depends on where the converter is presently set and what range and precision Hal needs. We look first at the fastest case, in which the range and the precision of the C or the F thermometer happen to be already set as Hal wants them to be. This analysis will give us the minimum time needed to use this interface.

- Write down the gestures Hal uses as he moves his hand to the GID and clicks and holds down the GID button on the desired arrow:

  HPK

- Continue listing gestures as Hal moves the arrow until it points to the correct value and then releases the GID button:

  HPKPK

- Place *M*s according to rule 0:

  HMPMKMK

- Eliminate two *M*s according to rule 1:

  HMPKK

There are no cognitive units, no consecutive terminators, and no other reasons to apply rules 2 through 5. We find the total time by adding the times for each gesture:

$H + M + P + K + K$

0.4 + 1.35 + 1.1 + 0.2 + 0.2 = 3.25 seconds

This calculation applies to the lucky case in which the input thermometer was preset to the appropriate range and resolution. Now consider the case in which Hal wants to expand the scale factor so that he can see the desired temperature, change the range, compress the scale factor to get adequate resolution, and then move the arrow. I will write down the method Hal uses, without going through a step-by-step derivation. (I assume that Hal is a perfect user and does not have to juggle back and forth to find the right places on the thermometer.) Hal has to use the arrows to scroll several times. Each scrolling operation may require several gestures; the computer then animates the scrolling operation, which takes time. To estimate scrolling times for the analysis, I built a similar interface and measured scrolling times, which were all 3 seconds or longer. Using *S* to represent the scrolling times, we can write the sequence of gestures that Hal uses as follows:

HPKSKPKSKPKSKPKK

Using the rules to place *M*s, we get

H + 3(M + P + K + S + K) + M + P + K + K

0.4 + 3 * (1.35 + 0.2 + 3.0 + 0.2) + 1.35 + 0.4 + 0.2 + 0.2 = 16.8 seconds

Except for the rare case in which the thermometer scales are correctly set at the beginning of the problem, a perfect user will need more than 16 seconds to accomplish a temperature conversion using this method. A real—imperfect—user would jog the scales and the arrows back and forth and thus take even longer.

4-3 Measurement of Interface Efficiency

Every tool carries with it the spirit by which it has been created.

—*Werner Karl Heisenberg*

We have looked at two interfaces, one of which will take about 5 seconds to operate and the other of which will take more than 15 seconds to operate. It is clear which of the two better satisfies the requirement. The next question that we ask is how fast an interface that satisfies the requirement can be.

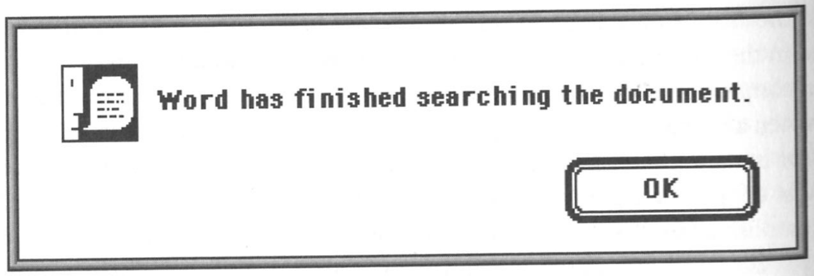
Given a design for an interface, you can use GOMS and its extensions to calculate how long a user will take to accomplish any well-defined task with that interface. But analysis models do not answer the question of just how fast you should expect an interface to be. To answer this question, we can use a measure from information theory. In the following discussion, *information* is used in the technical sense of a quantification of the amount of data conveyed by a communication, such as when two people have a telephone conversation or when a human sends a message, such as a click of the GID button when the cursor is at a certain location, to a machine. Before dealing with the technical details of measuring the amount of information a user must provide to accomplish a task, we establish the need for such a measurement.

To make a reasonable estimate of the time that the fastest possible interface for a task would take, we can proceed by first determining a lower bound on the amount of information a user has to provide to complete that task; this minimal amount is independent of the design of the interface. If the methods of a proposed interface require an input of information that exceeds the calculated lower bound, the user is doing unnecessary work, and the proposed interface can be improved. On the other hand, if the proposed interface requires the user to supply exactly the amount of information that the task requires, you cannot make a more information-efficient interface for this task. In this latter case, there may yet be ways of improving—and there are certainly many ways of ruining—the interface, but at least this one efficiency goal will have been met.

Information-theoretic efficiency is defined similarly to the way efficiency is defined in thermodynamics; in thermodynamics we calculate efficiency by dividing the power coming out of a process by the power going into that process. If, during a certain time interval, an electrical generator is producing 820 watts while it is driven by an engine that has an output of 1,000 watts, it has an efficiency of 820 / 1,000, or 0.82. Efficiency is also often expressed as a percentage; in this case, the generator has an efficiency of 82 percent. A perfect generator—which by the second law of thermodynamics cannot exist—would have an efficiency of 100 percent.

The **information efficiency** ***E*** of an interface is defined as the minimum amount of information necessary to do a task, divided by the amount of information that has to be supplied by the user. As is true of physical efficiency, E is at least 0 and is at most 1. Where no work is required for a task and no work is done, the efficiency is defined as 1. (This formality is necessary to avoid the case of 0 divided by 0, as in responding to a transparent error message. See Section 5-5.)

E can be 0 when the user is required to provide information that is totally unnecessary (Figure 4.4). Surprisingly, a number of interface details achieve the dubious honor of having $E = 0$. A dialog box that allows the user only one possible action, such as clicking the box's OK button, is such an example. (JavaScript has a command, *Alert*, solely for creating such unnecessary boxes: The designers were wise enough to remove *goto* from the JavaScript language to force structured code, but they failed to provide similar guidance on the interface side.)


Word has finished searching the document.

OK

Figure 4.4. A dialog box with an information theoretic efficiency of 0.

E takes into account only the information required by the task and that supplied by the user. Two or more methods may have the same E, yet have different total times. It is even possible that a first method has a higher E yet is slower than a second method—for example, $MKMK$ versus $MKKK$. In this example, only two characters have to be entered when the first method is used. In the second method, three characters are required, yet it takes less time to perform the task. It is difficult to construct many real-life situations that exhibit this inversion of speed and information efficiency.[2] For the most part, the more efficient interface is also the more productive, more humane interface.

Information is measured in bits; a single bit, which represents a choice between two alternatives—such as 0 and 1, on and off, or yes and no—is the unit of information.[3] For example, a choice made among four objects would require 2 bits of information: If the objects are A, B, C, and D, the first bit could choose either A and B, or C and D; once that choice was made—say C and D—the second bit would choose either C or D. Two binary choices, or 2 bits, suffice to separate one item from a set of four. To choose among eight alternatives, you need 3 bits; to choose among sixteen items, you need 4 bits; and so on. In general, given n equally likely alternatives, the amount of information communicated by all of them taken together is the power of 2 equal to n:

$$\log_2 n$$

And the amount of information in any one of them is

$$(1/n) \log_2 n \tag{1}$$

If the probabilities among the alternatives are not necessarily equal and the ith alternative has probability $p(i)$, the information associated with that alternative is

$$p(i) \log_2 (1 / p(i)) \tag{2}$$

The amount of information is the sum (over all alternatives) of expression (2), which reduces to expression (1) in the equiprobable case. It

2. It is possible to design more sophisticated measures of efficiency; for example, the M operator does not enter into our calculation. However, the simple measure defined here suffices for the purposes of this book.

3. *Bit* is mathematician John W. Tukey's contraction of the words BInary digitT (Shannon and Weaver 1963, p. 9).

follows that the information content of an interface that allows only the tap of a single button is 0 bits; not tapping the button is not permitted:

$$1 \log_2 (1) = 0 \tag{3}$$

It would seem, however, that the required tap of a single button can, for example, cause the ignition of dynamite used to demolish a building. Would this tap of the button then convey information? It would not, because not tapping the button was not an alternative; the interface "allows only the tap of a single button." If, however, the button was not tapped during, say, a five-minute time window in which the demolition was permitted, the building would not be demolished, and the tap or nontap would convey up to 1 bit of information because there were, in this case, two possible messages. From expression (2), we know that the calculation involves the probability, p, that the building will be exploded. The probability that it will not be exploded is therefore $1 - p$. From expression (2), we can calculate the information content of this interface:

$$p \log_2 (1 / p) + (1 - p) \log_2 (1 / (1 - p)) \tag{4}$$

When $p = ½$, expression 4 evaluates to

$$½ \times 1 + ½ \times 1 = ½ + ½ = 1$$

Expression (4) evaluates to less than 1 if $p \neq ½$. In particular, it evaluates to 0 when $p = 0$ or $p = 1$, as in expression (3).

This example illustrates an important point: We can measure the information embodied in a message only in the context of the set of possible messages that might have been received. To calculate the amount of information that has been conveyed by the reception of a message, we must know, in particular, the probability of that message having been sent. The amount of information in any message is independent of other messages past or future, is without reference to time or duration, and does not depend on any other events; similarly, the outcome of the flip of a fair coin is unaffected by previous tosses or by what time of day it is tossed.

As explained in Shannon and Weaver (1963), it is also important to keep in mind that

> information should not be confused with meaning . . . information is a measure of one's freedom of choice when one selects a message. . . . Note that it is misleading (although often convenient) to say that one or the other message [when just two are possible] conveys [1 bit of] information. The concept of information applies not to the individual messages (as the concept of meaning would), but rather to the situation as a whole, the unit information indicating that in this situation one has an amount of freedom of choice in selecting a message, which it is convenient to regard as a standard or unit amount. (p. 9)

However, a user's actions in performing a task could be modeled with greater accuracy as a Markoff process, whereby the probability of a later action depends on earlier actions taken by the user, but the single-event probabilities discussed are sufficient for the purposes of this book; messages are assumed to be independent and equiprobable.

The amount of information conveyed by nonkeyboard devices can also be calculated. If your display is divided into two regions—one labeled Yes and the other labeled No—a single click in one or the other region would supply 1 bit of information. If there are n equally likely targets, with one click, you supply $\log_2 n$ bits of information. If the targets are of unequal size, the amount of information given by each does not change, but it does take longer to move the GID to smaller targets—by an amount that we shall show how to calculate presently. If the targets have unequal probability, the formula is the same as that already given for keyboard inputs with unequal probabilities. There is a difference in that a user can operate a keyboard key in 0.2 sec, whereas it will take 1.3 sec to operate an on-screen button, on average, ignoring homing time.

For our purposes, we can calculate the information content of voice input by treating speech as a sequence of input symbols, rather than as a continuous phenomenon with a certain bandwidth and duration.

This treatment of information theory and its relationship to interface design is a simplified account. Yet even in this rudimentary form, information theory—used in a manner analogous to our use of the simplified GOMS keystroke-level model—can give us first-order guidance in evaluating the quality of our interface designs.

4-3-1 Efficiency of Hal's Interfaces

Men loven of propre kynde newefangelnesse.

—*Chaucer, "The Squire's Tale"*

It is useful to go through a detailed example of a calculation of the average amount of information required for an interface technique. I will again use the temperature-conversion example. According to the requirement, the input needed by the converter consists of an average of four typed characters; a decimal point occurs once in 90 percent of the inputs and not at all in the other 10 percent, and the negative sign occurs once in 25 percent of the inputs and not at all in the other 75 percent. For simplicity, and because there is no need for 1 percent precision in the answer, I will assume that all of the other digits occur with equal frequency, and I will ignore the 10 percent of the inputs that have no decimal point.

We need to determine the set of possible messages and the probability of each. Five forms are possible, where *d* denotes a digit:

1. –.*dd*
2. –*d*.*d*
3. .ddd
4. d.dd *and*
5. *dd*.*d*.

The first two each occur 12.5 percent of the time, and there are 100 of each of them; the final three each occur 25 percent of the time, and there are nearly 1,000 of each.[4] The probability for either of the first two types of messages is (0.125 / 100) = 0.00125; the probability for any one of the final three types of messages is (0.75 / 3000) = 0.00025. The sum of the probabilities of the messages is, as it must be, 1.

The amount of information of each message, in bits, is given by expression (2)[5]:

$$p(i) \log_2 (1 / p(i))$$

This expression evaluates to approximately 0.012 for the negative values and to 0.003 for the positive values. Calculating 200 × 0.0067 + 3000 × 0.003 gives a total of 11.4 bits for each message.

Taking the probabilities into account can be important. If we took a simple-minded approach and assumed that all of the 12 symbols (minus, decimal point, and the 10 digits) were equally likely, the probability of each would be $\frac{1}{12}$, and the information contained in a four-character message would be approximately

$$4 \log_2 (12) \approx 14 \text{ bits}$$

4. The "nearly" comes from the fact that the temperature of 0 degrees will not be entered as 0.00 or 00.0.

5. To get logs to the base 2 on a calculator or a computer that has only natural logs (ln), use: $\log_2 (x) = \ln (x) / \ln (2)$.

It is a theorem of information theory that the information is at a maximum when all symbols are equally likely. Therefore, making the assumption of equiprobable messages will give you a value that is equal to or greater than the amount of information in each message. Obviously, this assumption also makes estimating the information content of a message easier to compute. If the resultant value of the approximation is smaller than the amount of information your interface requires the user to supply, you do not yet need to bother with the more refined calculation.

We have just calculated that the task requires that Hal supply an average of about 11 bits of information each time he has to convert a temperature. We can—and will, presently—divide this quantity by the amount of information the interface requires him to supply. The result will be the efficiency of the interface.

Another simplification for quick analysis is to find the amount of information in a keystroke or a GID operation and then to count the various gestures. When a keystroke delivers information to a computer, the amount of information delivered depends on the total number of keys available—for example, the number of keys on the keyboard—and the relative frequency with which each key is used. Thus, keystrokes can be used as a rough measure of information. If a keyboard had 128 keys, each of which had the same frequency of use, each key would represent 7 bits of information. In practice, the frequency of use varies tremendously—for example, space and *e* are common, whereas *j* and \ are rare), and the information per keystroke is closer to 5 bits in most applications. The requirement stated that the average length of the input that specifies the temperature was four keystrokes.

For this analysis, it is easier to use a measure simpler than information-theoretic efficiency but that often achieves the same practical effect. **Character efficiency** is defined as the minimum number of characters required for a task, divided by the number of characters the interface makes the user enter.

Achieving an interface that required four keystrokes, on average, would give us a character efficiency of 100 percent. If we add a keystroke to decide which conversion is desired and then another to delimit the answer, our average length of input will grow to six keystrokes, and our keystroke efficiency will drop to 67 percent. If Hal has as his input device only a 16-key numeric keypad, the information provided by a single keystroke would be 4 bits, and the interface would be more efficient. (The requirements, however, do not permit us to use this option.)

Because any task in a GOMS analysis requires at least one mental operator, the most keystroke-efficient interface for the temperature-conversion problem will have, in theory, an average time of

$$M + K + K + K + K = 2.15 \text{ sec}$$

Thus, it will be considerably faster than either of the two interfaces already discussed. However, typing four characters on a standard keyboard supplies at least 20 bits of information, whereas only 10 bits are required—an information-theoretic efficiency of 50 percent—so we know that there is room for improvement. As we have seen, using a standard numeric keypad instead of a full keyboard drops the input information per four keystrokes to 16 bits, raising the efficiency to 62 percent. A dedicated numeric keypad—one that has only the digits, the minus sign, and a decimal point—will permit a slightly higher score, of about 70 percent efficiency. We raise the score again by using special encodings of temperature information and novel input devices, but training difficulties and excessive costs begin to loom with these extreme approaches, so I will stop here and accept 70 percent information-theoretic efficiency. Theoretical limits may or may not be reached by a practical interface, but they do give us a star by which to steer.

4-3-2 Other Solutions for Hal's Interface

In Section 4-3-1, we stopped trying to improve information-theoretic efficiency when we reached 70 percent. We achieved that efficiency with an unspecified, theoretical interface that somehow managed to have 100 percent keystroke efficiency. Let us see how close we can come to this ideal with a standard keyboard and a GID.

Consider an all-keyboard interface. In this interface, a note appears on the display:

> **To convert temperatures, indicate the desired scale by typing C for Celsius or F for Fahrenheit. Type the numeric temperature; then press the Enter key. The converted temperature value will be displayed.**

A GOMS analysis finds that the user must make six keystrokes. Following the rules for placements of *M*s gives us

$$MKKKKKKMK$$

The average time is 3.9 seconds.

We can decrease this time if we can use the C or the F itself as a delimiter. That is, consider an interface in which the following instructions appear:

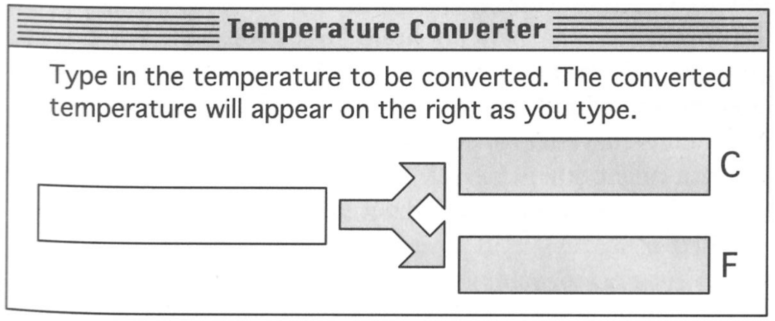
> **To convert temperatures, type the numeric temperature, followed by C if it is in degrees Celsius or F if it is in degrees Fahrenheit. The converted temperature will be displayed.**

In this example, the Enter key is not used. Some primitive interface-building tools demand that the user tap Enter and will not permit us to use C or F as a delimiter; such tools are inadequate for building humane interfaces.

The GOMS analysis of the C/F-delimiter interface yields

$$MKKKKMK$$

The average time is 3.7 seconds. If we did not have an analysis that showed that the theoretical minimum time is 2.15 sec, this solution might strike us as satisfactory. It is considerably more efficient than the ones that we discussed previously, so we might stop here. Tempted by that theoretical minimum, however, we ask whether there is an even faster approach. Consider the interface depicted in Figure 4.5; we might describe it as *bifurcated:* One input will give us two outputs.


Temperature Converter

Type in the temperature to be converted. The converted temperature will appear on the right as you type.

Figure 4.5. An interface that does not require a delimiter. A more efficient interface is made possible by taking advantage of character-at-a-time interaction, and by performing both conversions at once.

Under the bifurcated interface, no delimiter is required. Furthermore, the user does not have to specify which conversion is desired. The GOMS analysis for the average input of four characters is

$$MKKKK$$

The bifurcated interface achieves the minimum 2.15 seconds and has 100 percent character efficiency.

If, as in our example, the output sometimes changes when a character is typed, the flickering of the output does not distract you, because your locus of attention is the input. The continually changing output is often beneficial: The user will notice it only peripherally after the first few times that he uses the feature, at which point it will provide him feedback that the system is responding to his input. For single-character interaction to be effective, the system must respond quickly; in particular, the interaction must keep up with the user's typing speed. Only a slow network connection should exhibit this problem.

Although not part of the requirement, you might ask how this converter is "cleared" for the next operation. Does the clear operation add a keystroke? Not necessarily. For example, we could design the interface such that, whenever the operator returns to his background task or goes on to another task, the values in the converter are automatically grayed and the converter becomes inactive. The values shown are not cleared at this time, so that they can be referred to again if necessary. The next input to the converter does clear the old values.

Just because it has optimal speed of operation and is highly efficient, the bifurcated converter is not necessarily the best interface of those discussed or of those possible. Parameters other than speed also are of importance, such as error rate, user learning time, and long-term user retention of the way to use the interface. We should be especially concerned about the error rate of the bifurcated converter, due to Hal's possibly reading the wrong output box, especially because he may have just heard, for example, the word *Celsius* and thus be required to read out the Fahrenheit line. Nonetheless, the bifurcated converter would definitely be on the short list of interfaces to be tested for the temperature-converter application, and a few others that we have seen—solutions that might otherwise have seemed worth a try had we not learned how to do a GOMS analysis—would not make the cut.

Whether we use it in a simple keystroke-timing analysis or in a detailed information-theoretic extravaganza, a quantification of the theoretical minimum-time, minimum-character, or minimum-information interface can be a useful guide for our designs. *Without a quantitative guide, we are only guessing at how well we are doing and at how much room there is for improvement.*

4-4 Fitts' Law and Hick's Law

It behooves us to place the foundations of knowledge in mathematics.

—*Roger Bacon,* Opus Majus *(13th century)*

Various quantitative laws relating to interface design have sound cognetic underpinnings and have been validated repeatedly. These laws often give you additional data on which you can base interface-design decisions. Fitts' law quantifies the fact that the farther a target is from your current cursor position or the smaller the target is, the longer it will take you to move the cursor to the target. Hick's law quantifies the observation that the more choices of a given kind you have, the longer it takes you to come to a decision.

4-4-1 Fitts' Law

Consider that you are moving a cursor toward an on-screen button. The button is the *target* of the move. The length of a straight line from the position at which the cursor started to the closest point on the target is the *distance* used in the statement of Fitts' law. Given the size of the target and the distance to be moved, **Fitts' law** gives you the average time it takes a user to succeed in getting the cursor to the button.

In the one-dimensional case, in which the target's size, measured along the line of motion, is S and the target is at a distance D from the starting position (Figure 4.6), Fitts' law states that

$$\text{Time (in msec)} = a + b \log_2 (D / S + 1)$$

(The constants a and b are determined experimentally or are derived from human performance parameters.)[6] The time that you calculate begins when

6. Mathematics, supposedly a paragon of clarity, clings yet to that old-fashioned style whereby undefined variables appears in a formula before you know what they stand for. For example, you will see such statements as

$$A = \pi r^2,$$

where r is the radius of a circle and A is its area.

This can be confusing, forcing you to read ahead and then go back, especially if the equation is a long one with lots of as-yet-unexplained variables. Far better, from a reader's viewpoint, is to follow the obvious dictum to define terms *before* you use them:

A circle with radius r has an area A, given by:

$$A = \pi r^2$$

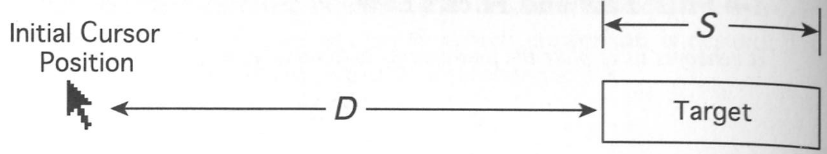


Figure 4.6. Distances used in Fitts' law to determine the time to move a cursor to a target.

the cursor is at the starting point and after the user has chosen the target. The logarithm to the base 2 gives a measure of the difficulty of the task in terms of the number of bits of information it takes to describe the (one-dimensional) path of the cursor.

The units of distance do not affect the calculated time, because D / S is the ratio of two distances and is therefore dimensionless. It follows that, even though we might move the pointing device a distance smaller or larger than the distance the cursor moves on the display, the law still works when the distances are measured on the display, assuming a linear relationship between GID and cursor motion. Fitts' law applies only to the kinds of motions we make when we are using most human-machine interfaces: motions that are small relative to human body size and that are uninterrupted, that is, movements that can be made in one continuous motion. For back-of-the-envelope approximations, I use $a = 50$ and $b = 150$ in the Fitts' law equation.

An extension of Fitts' law to more complex constraints, such as tracking a cursor between straight or curved walls, has been developed and tested empirically (Accot and Zhai 1997). For a two-dimensional target, you can usually obtain a reasonable approximation of the time needed to move the cursor to the target, using the smaller of the horizontal and vertical dimensions of the target for the value of S (Mackenzie 1995).

Fitts' law explains, for example, why it is much faster to move the cursor to an Apple Macintosh–style menu (Figure 4.7) that is on the edge of a display than to a Microsoft Windows–style menu (Figure 4.8) that floats away from an edge. The size S of the Windows menu on my display is 5 mm. The effective size of the Macintosh target is large because you do not have to stop within the confines of the menu bar but rather can continue to move the GID any comfortable distance beyond that needed to put the cursor in the menu: The cursor stops at the edge of the display.

A series of tests I performed determined that users typically stop within about 50 mm of the edge of the display on the Macintosh, so we can use 50 mm as S for the Macintosh. On a 14-inch flat panel display, the aver-

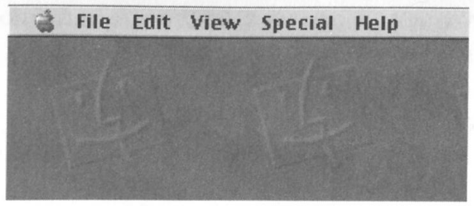


Figure 4.7. The Macintosh menu, at the top edge of the screen, effectively increases its size compared to a menu that floats away from the edge. (See color insert.)

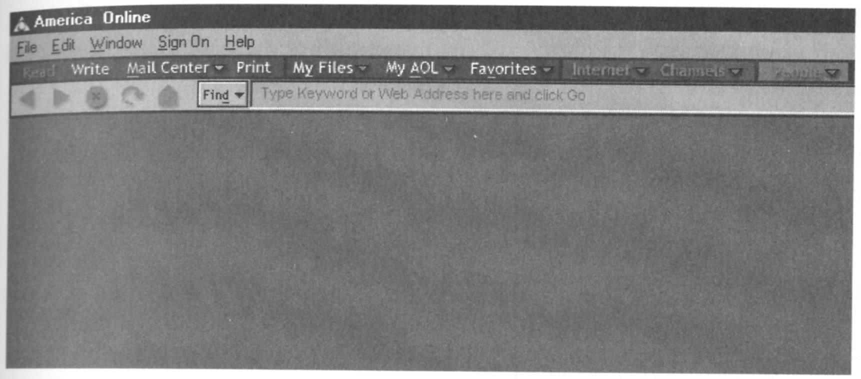


Figure 4.8. The Windows menu is below the top edge of the screen; you have to place the cursor more carefully to pull down a submenu. (See color insert.)

age distance the cursor must be moved to reach the menu bars is 80 mm; thus, the calculated time to move the cursor to a menu item on the Macintosh is

$$50 + 150 \log_2 (80 / 50 + 1) = 256 \text{ msec.}$$

This result is far less than the calculated time it takes to move the cursor to a corresponding menu item on a Windows-style menu:

$$50 + 150 \log_2 (80 / 5 + 1) = 663 \text{ msec.}$$

These calculations apply only to the time it takes the user to move the cursor. Clicking on the target to indicate that you believe that the cursor has reached its goal adds another 0.1 sec on average. ($K = 0.2$ in the GOMS model includes both the downstroke and the release of the button, whereas this timing is stopped by the downstroke.) In a typical experimental situation, you have to add the human reaction time of about 0.25 sec at the start of the cursor movement. When we take these factors into account, we get times that agree with what I have observed: It takes about 0.6 sec, on average, for a user to open an Apple menu, whereas it takes a user more than 1 sec to open a Windows menu. This analysis makes it clear why menus were deliberately placed at the edge of the display when the Macintosh interface was developed.

4-4-2 Hick's Law

Before you move the cursor to a target or take any one action with a multiplicity of choices, you must first choose the target or action. **Hick's law** says that when you have to choose to take one among *n* alternative actions and when the probabilities of taking each alternative are equal, the time to choose one of them is proportional to the logarithm to the base 2 of the number of choices, plus 1. When put this way, Hick's law looks just like Fitts' law:

$$\text{Time (in msec)} = a + b \log_2 (n + 1)$$

If the probability of the *i*th choice is $p(i)$, then, instead of the logarithmic factor in the equation, you use

$$\sum_i p(i) \log_2 (1 / p(i) + 1)$$

The coefficients in Hick's law are strongly dependent on many conditions, including how the choices are presented and how habituated to the system the user has become. (If the choices are presented in a confusing manner, both *a* and *b* can increase; habituation decreases *b*.) These dependencies will not be discussed here; all we need to consider is that making decisions takes time, that making complex decisions takes more time than making simple ones, and that the relationship is logarithmic. In the absence of better information, we can use the same coefficients *a* and *b* as for Fitts' law to make off-the-cuff or relative estimates.

Whatever positive, nonzero coefficients we use for *a* and *b*, it follows from Hick's law that giving a user many choices simultaneously is usually faster than is organizing the same choices into hierarchical groups. Making choices from one menu of eight items is faster than is making choices from two menus of four items each. Assuming that the items are equally likely to be chosen—and ignoring the time it takes to open the second menu, which, if taken into account, would make the time taken for the two-menu interface even longer—we compare the time to select one item of eight, $a + b \log_2 8$, with the time to select one item of four twice, $2 (a + b \log_2 4)$. Because $\log_2 8 = 3$ and $\log_2 4 = 2$, and because both $a < 2a$ and $3b < 4b$, we see that

$$a + 3b < 2 (a + 2b)$$

This accords with experiments on menu structures (see, for example, Norman and Chin 1988).

Our discussion of Fitts' and Hick's laws is incomplete. For example, it is no accident that they have the same form as the Shannon-Hartley theorem. Nonetheless, this brief treatment is sufficient to alert you to these useful guides to interface design. They can help you even if, as in the example, you do not know the empirical coefficients *a* and *b.* (For more detail, see Card, Moran, and Newell 1983, pp. 72–74.)