1
Introduction

"Semiotic" or "semiotics" are terms not frequently found in the HCI literature. Search results can be taken as a rough indication of the relative scarcity of semiotic approaches to HCI. In December 2003 a search for *semiotic* in the HCI Bibliography (see HCI Bibliography) returned 22 records, whereas a search for *ethnographic* returned 104, one for *ergonomic* returned 344, and finally one for *cognitive* returned 1,729. Although these numbers cannot be taken as a precise measure, the difference in scale speaks for itself. Semiotics is the study of signs, signification processes, and how signs and signification take part in communication. It is neither a new discipline, nor one whose object of investigation is foreign to HCI. In its modern guise, it has been established for approximately one century. But the debate about its central themes of investigation—meaning assignment, meaning codification, and the forms, ways, and effects of meaning in communication—dates back to the Greek classics.

Most of the relatively rare allusions to semiotic principles or theories in HCI concentrate on aspects of graphical user interfaces and visual languages. However, the World Wide Web and the multicultural issues of Internet applications have raised another wave of interest in semiotics. Since sign systems are produced and perpetuated by cultures, some contemporary semioticians define semiotics as a theory of culture (Eco 1976; Danesi and Perron 1999). Their views are naturally an attractive foundation for those interested in understanding the nature and enhancing the design of computer-mediated communication in various social contexts, such as computer-supported collaborative work, online communities, computer-supported learning, and distance education.

1.1 Semiotic Theories of HCI

To date only a few authors have taken a broader and more radically committed semiotic perspective on the whole field of HCI (see, e.g., Nadin 1988; Andersen, Holmqvist, and Jensen 1993; de Souza 1993; Andersen 1997). This is mainly because, in spite of its being a resourceful discipline for HCI researchers, teachers, and practitioners, semiotics covers a vast territory of concepts and uses a wide variety of analytic methods that are neither familiar nor necessarily useful to them. Semiotic engineering is one of the few attempts to bring together semiotics and HCI in a concise and consistent way, so as to support new knowledge organization and discovery, the establishment of useful research methods for analysis and synthesis, and also the derivation of theoretically sound tools for professional training and practice.

First presented as a semiotic approach to user interface design (de Souza 1993), semiotic engineering has evolved into a theory of HCI. As its name implies, it draws on semiotics and on engineering to build a comprehensive theoretical account of HCI. Semiotics is important because HCI involves signification and meaning-related processes that take place in both computer systems and human minds. And engineering is important because the theory is expected to support the design and construction of artifacts. Moreover, semiotics and engineering become tightly coupled when we think that HCI artifacts are intellectual constructs, namely, the result of choices and decisions guided by reasoning, sense making, and technical skills, rather than predictable natural laws. Like all other intellectual products, HCI artifacts are communicated as signs, in a particular kind of discourse that we must be able to interpret, learn, use, and adapt to various contexts of need and opportunity. Thus, the semiotic engineering of HCI artifacts is about the principles, the materials, the processes, the effects, and the possibilities for producing meaningful interactive computer system discourse.

The semiotic engineering account of HCI makes explicit references to the theory's object of investigation, the interests and values involved in establishing this object as such, the epistemological conditions and methodological commitments that affect the results of investigation, and the prospect of scientific knowledge advancement in relation to its perceived conditions of use and validity. It also aims to provide the basis for useful technical knowledge for HCI professions—that is, knowledge that can be translated into professional tools (abstract or concrete), knowledge that can improve the quality of professional products or services, knowledge that can be

taught to young professionals and refined by practice, and knowledge that can be consistently compared to other practical knowledge, and be complemented, supplemented, or replaced by it.

The importance of dealing explicitly with epistemological issues in a theory of HCI, namely, with issues related to how knowledge is gained, analyzed, tested, and used or rejected lies in our need to discriminate the validity, the reach, and the applicability of HCI knowledge coming from such widely different areas as computer science, psychology, sociology, anthropology, linguistics, semiotics, design, and engineering, among others. We aim to help semiotic engineering adopters identify the need and the opportunity to use the knowledge this theory proposes, as well as the limitations and gaps that call for further investigation and other types of theories.

One of the prime advantages of a semiotic perspective on HCI is to center a researcher's attention on signs. Signs have a concrete objective stance that is produced and interpreted by individuals and groups in a variety of psychological, social, and cultural contexts. They are encoded in natural or artificial signification systems of widely diverse kinds, and they are typically used to communicate attitudes, intents, and contents in a multiplicity of media. Most semiotic approaches to HCI view computer programs as (sources of) artificial signification systems, and computer-based devices as media. For example, in figure 1.1 we see various signs in SmartFTP© v.1.0.979's interface: English words and technical terms on the main menu bar; drawings and sketches of various objects on toolbars; pull-down menus, text boxes, grids, and other widgets. Each one means something, to users and designers. Other signs appear as a result of interaction, such as the ability to drag items from a SmartFTP window into another application's window or workspace, and vice versa. These signs also mean things, such as the user's convenience or a system's requirement. SmartFTP incorporates a complex signification system that users must understand in order to take full advantage of the system's features.

The attitudes, intents, and contents communicated through interactive systems in computer media are those of the various stakeholders involved in the development of HCI artifacts—clients, owners, users, developers, designers, technical support professionals. Two of them have a distinguished status in HCI: designers and users. Designers, because they must be able to embed in the artifact they are about to produce the whole range of messages that are expected by and useful to all stakeholders (including themselves). And users, because they are the ultimate judges of whether the artifact is good or not.

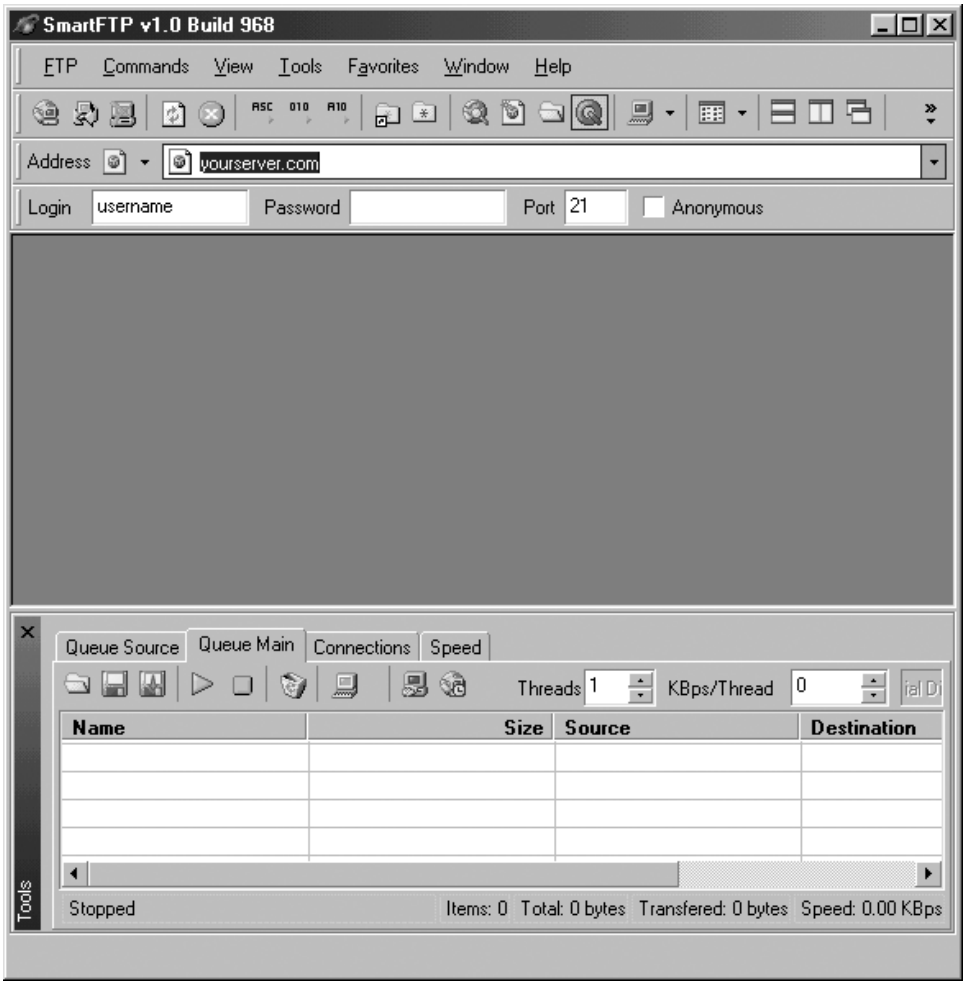

Figure 1.1
SmartFTP© v.1.0.979's interface. Screen shot reprinted by courtesy of SmartFTP© (http://www.smartftp.com).

1.2 The Semiotic Engineering Framework

Semiotic engineering starts from this general semiotic perspective and assigns to both designers and users the same role in HCI, namely, that of interlocutors in an overall communicative process. Designers must tell users what they mean by the artifact they have created, and users are expected to understand and respond to what they are being told. This kind of communication is achieved through the artifact's interface, by means of numerous messages encoded in words, graphics, behavior, online help, and explanations. Thus, by using this theory to study, analyze, and make decisions about users and their expected reactions, designers are simultaneously studying, analyzing, and making decisions about their own communicative behavior and strategies. Semiotic engineering is therefore an eminently reflective theory, which explicitly brings designers onto the stage of HCI processes and assigns them an ontological position as important as that of the users'.

This perspective is considerably different from and complementary to the user-centered perspective that has shaped our field in the last two decades following Norman and Draper's seminal publication (1986). In user-centered design (UCD), designers try to identify as precisely as possible what the users want and need. User studies and task analysis allow designers to form a design model that matches such wants and needs. The design model is projected by the system image, which users must understand and interact with to achieve their goals. Thus the system image is the ultimate key to success. If the design model is conveyed through the appropriate system image, which rests on familiar concepts and intuitive relations, users can easily grasp and remember how the system works. In semiotic engineering, however, although designers also start by trying to understand users and what they need or want to do, they are not just trying to build the system image. They are trying to communicate their design vision to users. Their design vision amounts to how users may or must interact with a particular computer system in order to achieve their goal through many communicative codes integrated in the system's interface. In figure 1.2 we see a schematic comparison of UCD and semiotic engineering. On the UCD side, the system image is the only trace of all the intellectual work that has taken place during design, and it is what the user is required to learn. On the semiotic engineering side, the designer herself is present at interaction time, telling the user about her design vision, to which the user will respond in various ways (including unexpected and creative ways). The ideal of UCD is that the user model captures the essence of the design model, projected in the system image. The ideal of

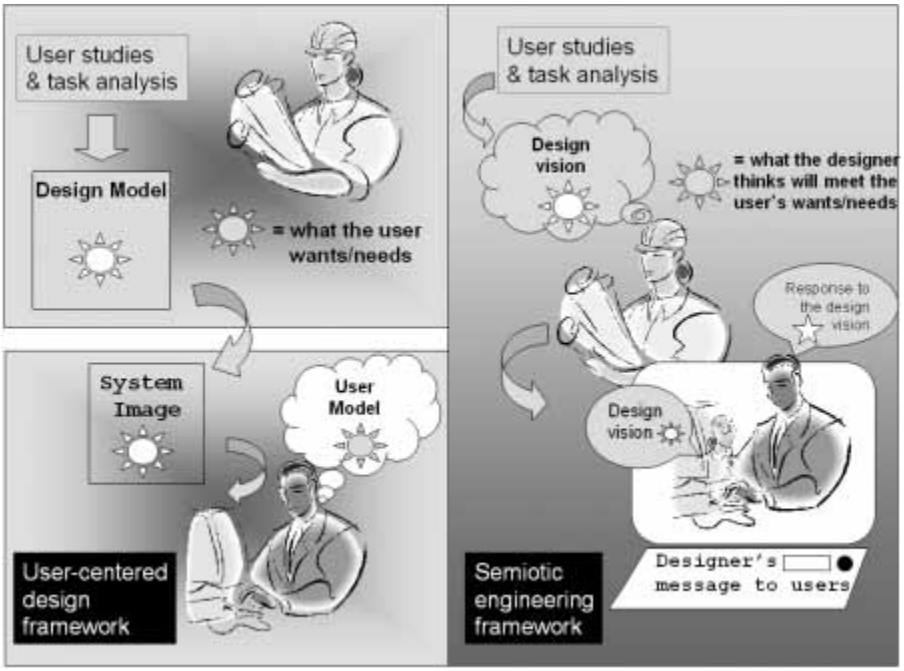

Figure 1.2
User-centered design compared to semiotic engineering.

semiotic engineering is that designer and user understand each other, and that users find the designer's vision useful and enjoyable.

The gist of user-centeredness is a radical commitment to understanding the users and asking them what they want and need. Their answers set the goals that professional HCI design practices must meet for interactive computer-based artifacts to be usable, useful, and widely adopted. Most of the technical knowledge required for successful UCD has resulted from empirical studies that attempted to associate certain features of interactive software to certain kinds of user behavior and reaction. Design features and patterns that have demonstrably met user-centeredness targets in a statistically significant quantity of situations have been compiled into legitimate technical knowledge in the form of principles, heuristic rules, and guidelines. The greatest difficulty with this kind of research practice in our field, however, has been the cost of sound predictive theories, especially in view of the speed of technological progress. The statistic validity of empirical studies requires that

numerous experiments be run with hundreds, if not thousands, of subjects before any piece of knowledge is legitimately added to an existing theory.

But two familiar obstacles have been always in the way of such extensive studies—the pressure exercised by industrial stakeholders, who can't afford the time and money involved in extensive latitudinal and/or longitudinal user studies, and the pace of technological change, which often precipitates the obsolescence of systems and techniques employed in long-term research projects. As a result, sound predictive theories of HCI have typically concentrated on specific phenomena that cut across technologies (e.g., Fitt's Law [Fitt 1954]), but are of little use when it comes to making other types design decisions (e.g., choosing between adaptive or customizable interfaces). Moreover, we have witnessed a certain glorification of guidelines and heuristic rules. In some cases, practitioners have focused on conforming to guidelines and heuristic rules instead of using them appropriately as decision-support tools. Others have overemphasized the value of checklists and forgone much of their own intellectual ability to analyze and critique the artifact they are about to build. Those who exercise this power have often been misled into thinking that because guidelines don't take them all the way to a final and unique design solution, research work from which guidelines are derived is really of little use (Bellotti et al. 1995; Rogers, Bannon, and Button 1994).

Adding other types of theories to HCI, such as explanatory theories and intervention-oriented theories (Braa and Vidgen 1997), helps us gain sound knowledge that supports decision making in design, although not on the basis of empirical data. Conceptual schemas and interpretive models can enrich our analysis of problem situations and candidate solutions. They can also help us frame problems in different ways and find opportunities for exploring design paths that would possibly not emerge in causal reasoning processes that are typical of predictive theories.

1.3 Theorizing about Software as an Intellectual Artifact

In order to illustrate the need and opportunity for such other types of theories and research practices (i.e., nonpredictive), let us explore the intellectual nature of software artifacts. What is an intellectual artifact? What is not an intellectual artifact?

All artifacts are by definition nonnatural objects created by human beings. Some of them are concrete, like forks and knives—material artifacts that were made to facilitate certain eating practices in our culture. Others are more abstract, like safety

measures—procedural artifacts that are meant to prevent accidents. Some are meant for physical purposes, like chairs. Others are meant for mental purposes, like logic truth tables. Thus, strictly speaking, all artifacts result from human ingenuity and intellectual exercise. But what we call an intellectual artifact is one that has the following features:

- it encodes a particular understanding or interpretation of a problem situation;
- it also encodes a particular set of solutions for the perceived problem situation;
- the encoding of both the problem situation and the corresponding solutions is fundamentally linguistic (i.e., based on a system of symbols—verbal, visual, aural, or other—that can be interpreted by consistent semantic rules); and
- the artifact's ultimate purpose can only be completely achieved by its users if they can formulate it within the linguistic system in which the artifact is encoded (i.e., users must be able to understand and use a particular *linguistic encoding system* in order to explore and effect the solutions enabled through the artifact).

According to this definition, knives and forks are not intellectual artifacts, nor are safety measures and chairs. But logic truth tables are intellectual artifacts, and so are books (if we think of their content), although books can also be viewed as physical artifacts (if we think of the physical media where ideas and information are stored). All intellectual artifacts require that producer and consumer use the same language. And language is not a metaphor in this case; it is a genuine system of symbols with a defined vocabulary, grammar, and set of semantic rules. Natural language descriptions have an additional set of pragmatic rules, which refine the way in which semantic rules are applied in communicative contexts. But artificial languages usually don't include such pragmatic rules in their description.

The advent of graphical user interfaces (GUIs) popularized the idea that software was some kind of tool. Almost every interface began to include tools, toolboxes, and toolbars, with explicit reference to concrete physical artifacts. Gibson's theorizing on affordances (Gibson 1979) inspired HCI researchers (Norman 1988), and the idea of direct manipulation (Shneiderman 1983) turned the tool metaphor into one of the pillars of HCI design. But soon the metaphor started to show some of its limitations. For example, the Gibsonian notion of affordance could not be directly applied to the HCI. Designers intuitively began to speak of "putting affordances" in the interfaces (Norman 1999); evaluation methods spoke of users "missing and declining affordances" (de Souza, Prates, and Carey 2000), a sign that affordances were themselves being used as a metaphor and not as a technical concept. The invariant feature of all metaphorical uses of the term was the presence

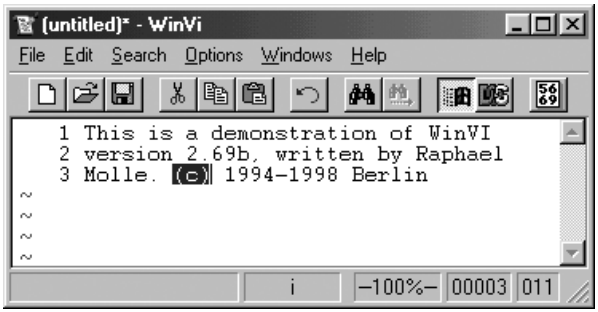

Figure 1.3
WinVi©, a graphical user interface for VI®. Screen shot reprinted by courtesy of Raphael Molle.

of an intended meaning, of what designers meant to do, or what they meant to say with specific interface elements. And discussions turned around the occasional mismatches between what was meant by designers and users when they referred to certain signs in the interface. Relevant issues having to do with how the designers' intent was encoded in the interface, how users decoded (interpreted) them, and how they used them to express their own intent during interaction could not fit into the category of typical concrete artifacts. They referred essentially to linguistic processes (though not necessarily involving only natural language signs). And the kinds of tools needed to support these processes are not material (like hammers, screwdrivers, paintbrushes, which all apply to concrete artifacts that we can manipulate), but rather immaterial (like illustrations, demonstrations, explanatory conversations, clarification dialogs, which all refer to intellectual artifacts that we can learn). They are epistemic tools, or tools that can leverage our use of intellectual artifacts.

Epistemic tools can benefit both designers and users, since they are both involved in communication with or about the same intellectual artifacts. It may be difficult to see why we need epistemic tools to use a basic text editor like WinVi©, shown in figure 1.3. The concrete tools in this GUI must all be familiar to computer-literate users, except perhaps for the last three that mean, respectively, ANSI character set, DOS® character set, and hexadecimal mode. So one might expect any computer-literate user to be able to write the text seen in figure 1.3, to edit it with the familiar cut, copy, and paste tools, and to open and save files, just as easily as he or she would use pencils, pens, paper, classifiers, and folders. However, unless the user is introduced to the idea behind WinVi, there is a chance that the power of this intellectual artifact will never be understood. Unless the user knows the Vi

Editor and its command language, the string operations that can be performed with WinVi, and the situations in which such operations are desirable and needed, he or she may end up complaining about the barrenness of this piece of software (e.g., compared to user-friendly ones that may not have half the string processing power of WinVi). Epistemic tools, like a useful online help system, with clever examples, explanations, and suggestions for how to use WinVi, should be provided with the artifact in order to enable users to take full advantage of this product.

Treemap© 3.2 (a visualization tool for hierarchical structures in which color and size may be used to highlight the attributes of leaf nodes) is another example of why epistemic tools are needed, just as theories that can help us build them. Visualizations allow for comparisons among substructures and facilitate the discovery of patterns. All interactions are based on direct manipulation of interface elements, either pertaining to the visualization itself or to specific controls applicable to it. In figure 1.4 we see data referring to the 2000 presidential elections in the United States. Lighter-colored rectangles correspond to states with population ranging from over 10 million to nearly 30 million people. The size of rectangles is proportionate to Gore's votes, whereas the brightness of the color is proportionate to Bush's votes. The pop-up rectangle provides details about the number of electoral votes and each of the candidates' votes in each state (or rectangle). In the "slice and dice" visualization mode shown in figure 1.4, the layout represents the states grouped in a fashion that mimics the geographical layout. The user may choose which label to see on the rectangles. On the upper right part of the screen is more information about Florida, the selected state on the left side. On the lower part, in the Filters tab, are various sliders that can help the user narrow in on specific ranges of values for the various attributes of the data.

As was the case with WinVi, Treemap may go underestimated and underutilized if users are not introduced to the power of this data visualization tool. Although the interactive patterns are not difficult to learn and support manipulations that can help the user find aspects of information that might be lost in tabular presentations, they are not obvious for average users. The intellectual value added to the tool by its designers is not clear from the signs in the interface. There is no reference to patterns of data, and no hint of when or why one kind of visualization is better than the other. But of course the designers included alternative visualizations because some are better than others in certain situations. However, unless the users are told about the relative advantages of each or get at least some sort of cue, they may miss the whole idea of the tool.

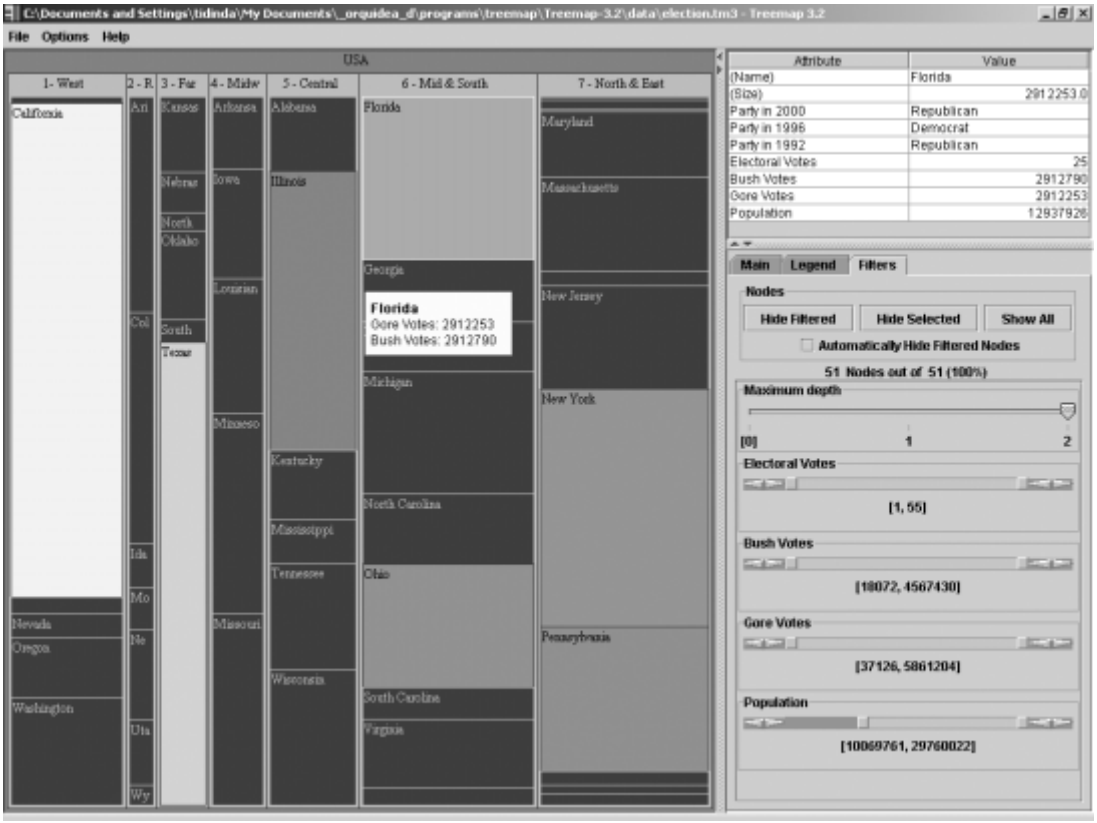

Figure 1.4
Direct manipulation and data visualization possibilities in Treemap© 3.2. Screen shot reprinted by courtesy of HCIL, University of Maryland, College Park (http://www.cs.umd.edu/hcil/Treemap/).

Intellectual tools deserve an appropriate presentation or introduction. Not only in operational terms (which is the most popular kind of online help users get from software manufacturers), but also (and perhaps more interestingly) about the problem-solving strategies that the designers had in mind when they produced the software. A detailed example of how users would benefit from a more careful introduction to knowledge associated with the product will help us draw some conclusions about the intellectual aspects of software production and use that nonpredictive theories such as semiotic engineering can bring to light and help explore. To this end I will employ a use scenario from Adobe® Acrobat® 5.0.5.

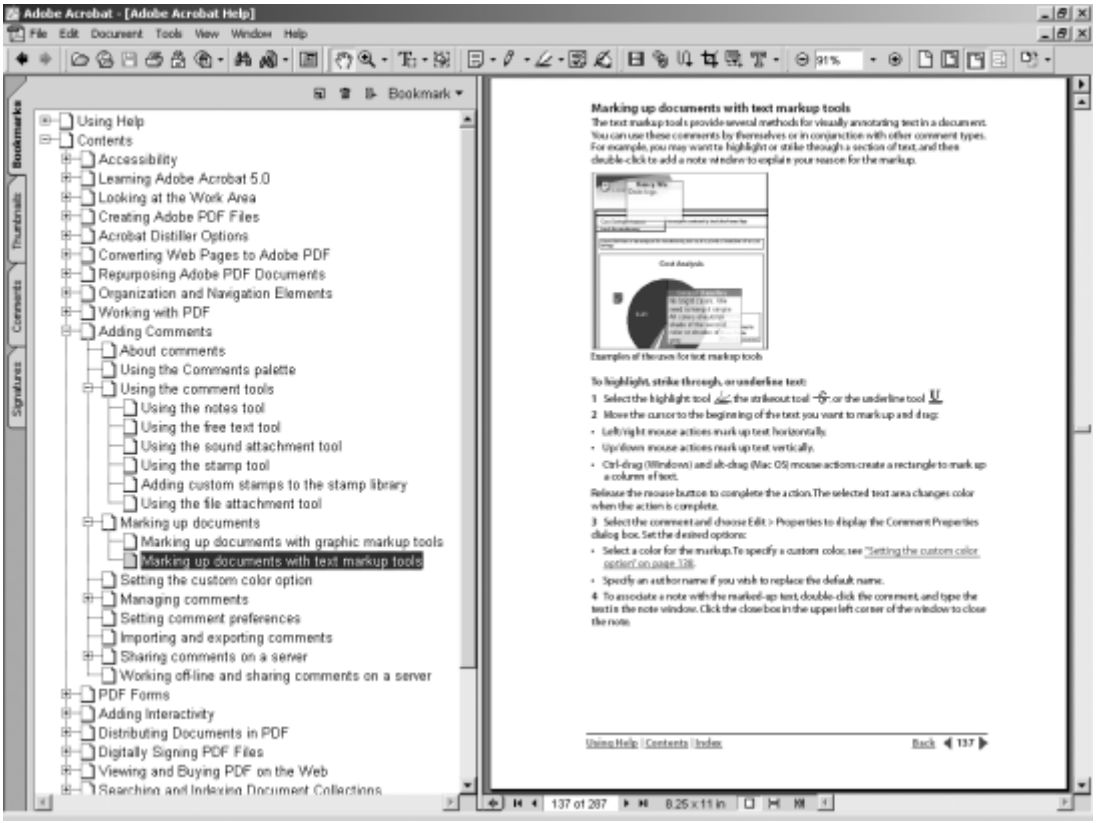

Figure 1.5
Acrobat® 5.0.5 standard interface. Adobe® product screen shot reprinted with permission from Adobe Systems Incorporated.

Acrobat offers to users a set of powerful tools for reviewing documents. Among the most useful are "commenting tools" and "markup tools." In figure 1.5 we see a screen shot of Acrobat showing its own help documentation as a PDF file that can be commented and marked up. On the top of the screen is the usual menu bar and tool bars. On the left-hand side are the bookmarks, which in this case contain a detailed summary of the document. And on the right-hand side is a document page linked to the highlighted entry of the summary ("marking up documents with text markup tools"). The "note tool" is an example of a commenting tool, and the "highlight tool" is an example of a markup tool. The analogy with concrete artifacts such as notes and highlight markers is emphasized by the visual signs appear-

ing on the toolbars on top of the screen. Thus, users can write notes and highlight portions of text while they are reviewing documents.

Following the UCD cannons, the system image directly projects the idea that users can call a document's reader's attention to portions of the text and write comments about them. So, for instance, a reasonable reviewing strategy that most users are likely to adopt is to highlight parts of the text that need to be reviewed and then write a note with suggestions and justifications or questions. The steps to achieve this goal are as follows:

1. to select the highlight tool
2. to press down the mouse button at the beginning of the text portion to be highlighted
3. to drag the mouse to the end of the text portion to be highlighted
4. to release the mouse button
5. to select the note tool
6. to click on the document page close to the highlighted portion
7. to double-click on the note icon
8. to type in comments and suggestions
9. to close the note (optional)

The visual effect of these steps can be seen in figure 1.6. The first two lines of the top paragraph on the page are highlighted, and next to them is a note where the user comments that she never knew highlights were comments. This user has been using Acrobat for over a year now, and her reviewing strategy has always been what she calls the "highlight-and-add-note" strategy. But she learned a few things about this strategy. For example, the spatial contiguity of the note and the highlighted text is of course critical. Otherwise, sentences like "This is new to me" cannot be correctly interpreted. *This* must always refer to an object that is spatially contiguous to it. Another important lesson is about producing a summary of her comments, which she often does using the "summarize tool" (see figure 1.7). This tool generates a listing of comments made by the user. The comments can be filtered and sorted in various ways, but the default situation is to list comments and markups in sequential order of creation. Therefore, when this user generates a summary of her document, the content of a note she wrote following steps 6 to 8 comes right next to the text she has highlighted in steps 2 to 4. As a result, she can still interpret *this* in the listing as referring to the contents of the two highlighted sentences. The referent of *this* is shown in the previous entry of the summary.

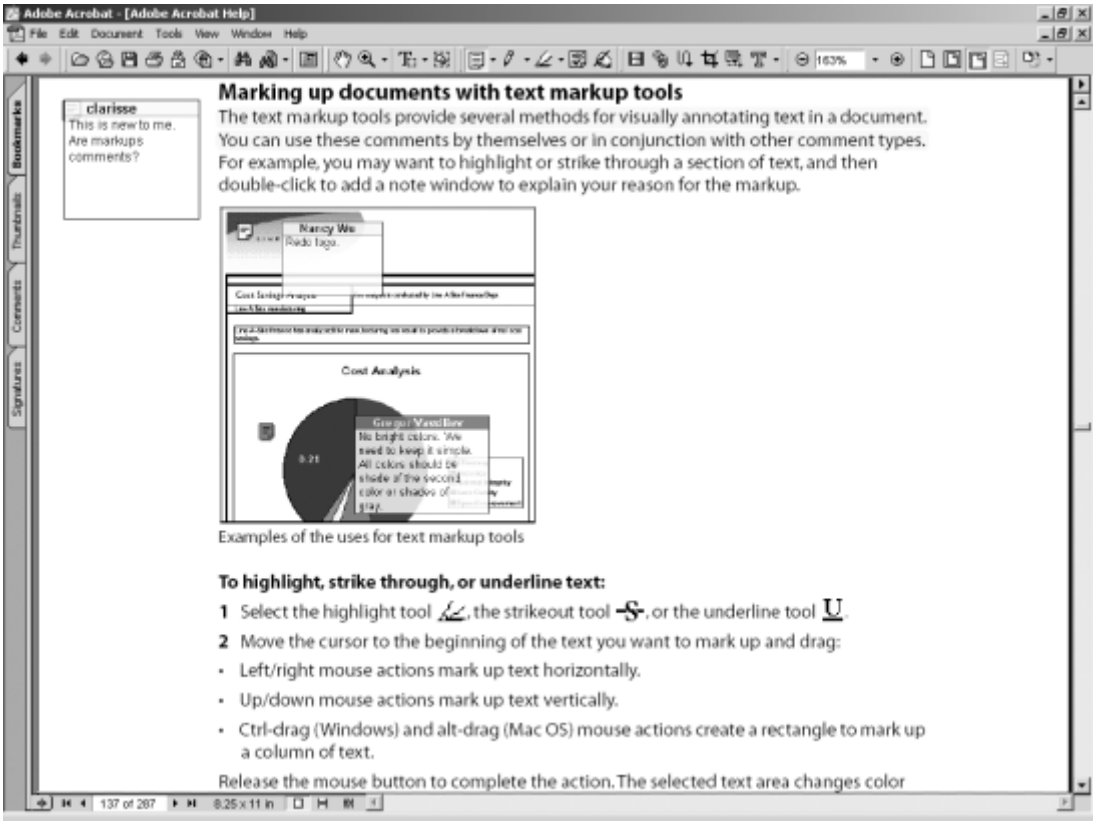

Figure 1.6
Notes and highlights in Acrobat® documents. Adobe® product screen shot reprinted with permission from Adobe Systems Incorporated.

At times, when checking all of her comments, she has tried to extend the highlighted portion of text to which comments refer. For instance, suppose that after doing a lot of reviewing on Acrobat's help document, the user highlights the other two lines of the first paragraph. Assuming that no other comments and markups have been made on this page, this becomes the third summary entry, and is listed after the note contents. As a result, although visually the note contents correctly refer to the four lines in the first paragraph on the page seen in figures 1.6 and 1.7, textually the reference is lost in the summary (see figure 1.8). The note comment is inserted between the first and the second two lines of the paragraph, and *this* seems to refer only to the part of the text, although it really refers to all of it.

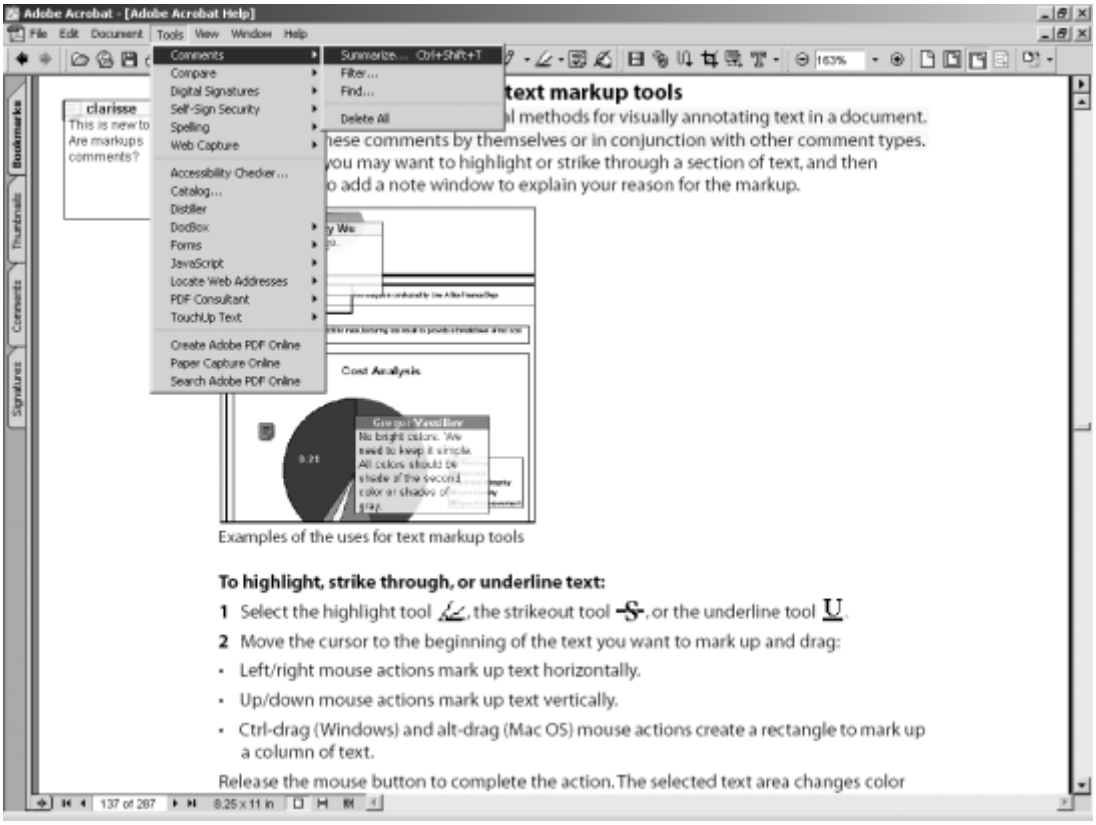

Figure 1.7
Menu choices for Comments in Acrobat®. Adobe® product screen shot reprinted with permission from Adobe Systems Incorporated.

The trick in Acrobat's interface is that, given how the system image is projected, the highlight-and-add-note strategy is the most obvious to users. And this may lead to problems in generating summaries. However, as the Acrobat's help documentation is saying, the highlight-and-add-note strategy is not exactly the design vision for efficient commenting and reviewing. As can be read in figure 1.5, designers do believe that users "may want to highlight or strike through a section of text, and . . . add a note . . . to explain [the] reason for the markup." However, the way to do this, in their view, is not to use the note tool but to "double-click [on the highlight or strike-through] to add a note window." But the system image does not suggest that this is a possibility, or that "highlights" and "notes" are both "comments"

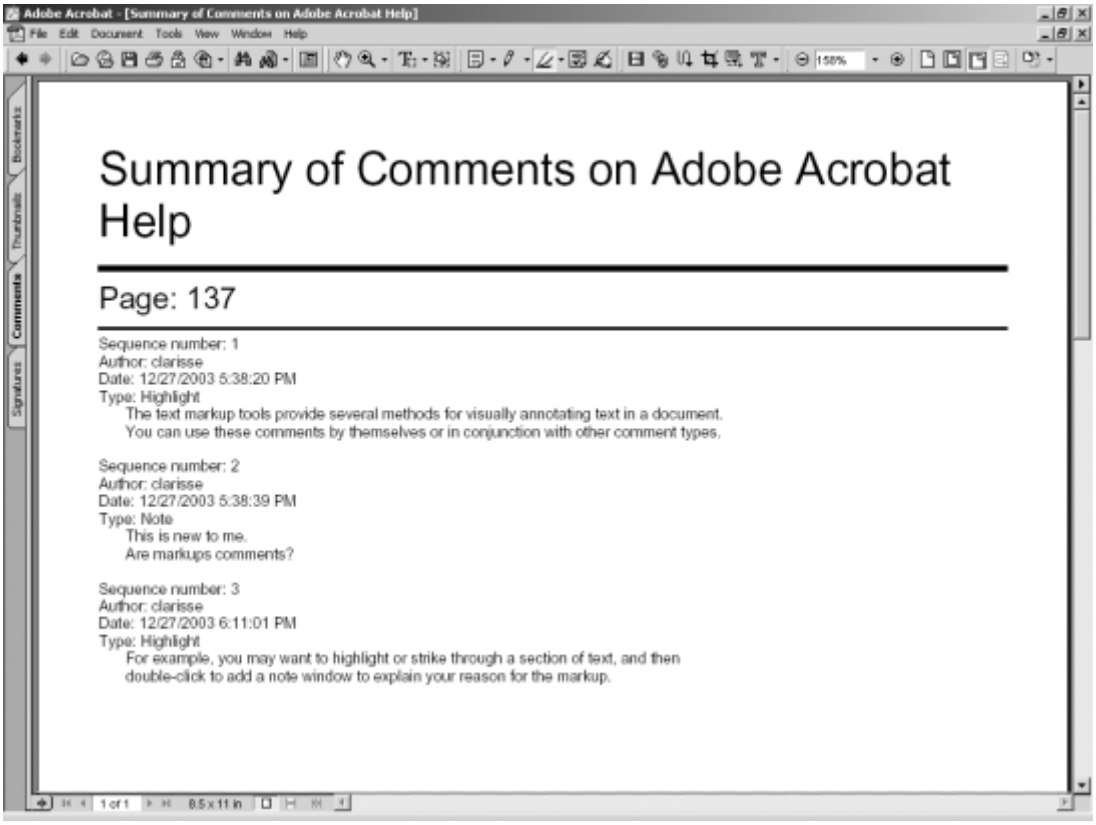

Figure 1.8
Textual discontinuity of visually continuous highlighted text in comment summary. Adobe® product screen shot reprinted with permission from Adobe Systems Incorporated.

(which is precisely what the user is surprised to learn in the documentation). In the physical world, these are different things.

If Acrobat's default configurations are changed, however, the system image gives the user a hint that maybe notes and highlights have indeed something in common. The user can change her preferences and choose to see the sequence number of every comment. The visual effect of this change can be seen in figure 1.9. Note that both notes and highlights have "comment sequence numbers" (this is how this configuration parameter is named in the preference dialog). So, they are both comments. And, since we can double-click on note icons to open small windows and type in

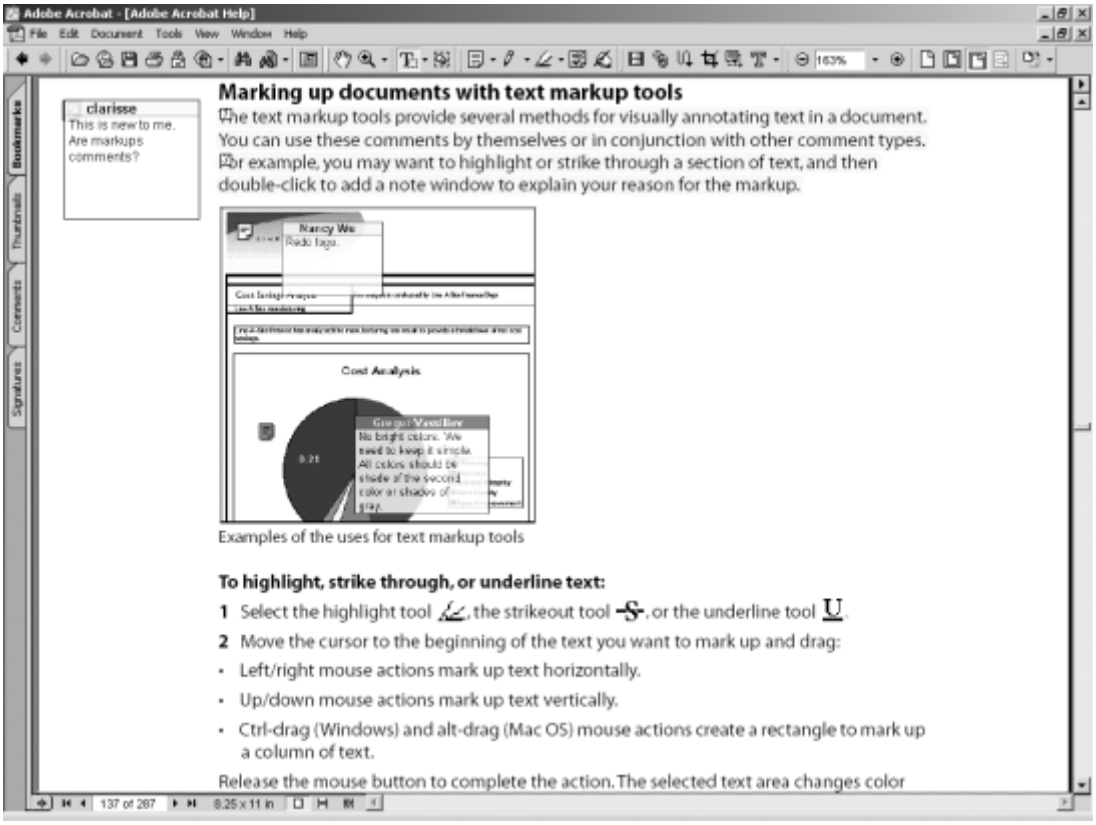

Figure 1.9
Visual similarities between comments and markups as a result of changing preferences. Adobe® product screen shot reprinted with permission from Adobe Systems Incorporated.

comments, we can also expect to double-click on highlighted texts in order to open the same kind of windows and type in comments.

The effect of this discovery on our user is much deeper than we may at first suspect. In fact, when she realizes that every highlight is a comment, just like a note, it dawns on her that she does not need to create a note to comment every highlight in the document. She can write her suggestion, justification, or question directly in the comment window associated with the highlight. And as she begins to do this, she realizes that the note window referred to in Acrobat's documentation indeed contains the whole extent of marked-up text. If she chooses the following interactive steps:

1. to select the highlight tool
2. to press down the mouse button at the beginning of the text portion to be highlighted
3. to drag the mouse to the end of the text portion to be highlighted
4. to release the mouse button
5. to double-click on the highlighted text
6. to type in comments and suggestions
7. to close the note (optional)

the summary of comments will look smaller and more readable like this:

```
"Sequence number: 1
Author: clarisse
Date: 12/27/2003 6:57:22 PM
Type: Highlight

The text markup tools provide several methods for visually
annotating text in a document. You can use these comments
by themselves or in conjunction with other comment types.
For example, you may want to highlight or strike through a
section of text, and then double-click to add a note window
to explain your reason for the markup.

This is new to me. Are markups comments?"
```

---

and not like what is shown in figure 1.8. The design vision, according to what can be read in Acrobat's online documentation, aligns with the "comment-on-highlight" strategy, although the system image projects the highlight-and-add-note strategy with greater salience. Moreover, spatial contiguity problems disappear in the textual summary.

Without getting into a "bug or feature" discussion with respect to this example (i.e., assuming that similarities in visualizations of notes and highlights is a feature and not an unanticipated side effect of parameter-setting implementation), the truth is that operating with comment sequence numbers on and off communicates different meanings in Acrobat. And not all of these values have a logical connection with numbering comments. In particular, if the user knows that highlights (and other markup tools) automatically generate a comment, this can make reviewing more agile and consistent. If she doesn't, not only does this introduce unnecessary redun-

dancies in the reviewing process, but this may also cause loss of information (as when the reader must interpret a word like *this*, referring to something other than what is spatially contiguous to it). Users are simply not likely to guess the best strategy from just looking at Acrobat's interface and focusing on the system image.

It is useful now to contrast two design objectives—producing and introducing an application that meets certain requirements. If Acrobat's designers had been given the goal to produce an application that allows users to operate in different ways and customize it to meet their needs, the problem with this interface would be that the system image does not portray equally well all of the product's features. In other words, the flexibility to tailor the application to different users' profiles is there, but it is poorly communicated. However, if designers had been given the goal to introduce such application, they wouldn't have even partially met their goal. Acrobat's more sophisticated uses are far from self-evident and do not speak for themselves. They may not go without an explicit introduction. In particular, tactical and strategic choices that users may or should make to increase their productivity and/or satisfaction cannot be derived from the strict meaning of the tools that they see in the application's interface. Tools tell them a number of things about Acrobat's operational features, but nothing about how this technology may bring about desirable changes in work strategies or tactics.

1.4 New Nonpredictive Theories for HCI

Nonpredictive theories of HCI, such as the semiotic theories that have guided the analyses of the examples above, can help us gain access to some relevant issues. The first is to add a new perspective on usability and focus on some factors influencing usability that are not totally obvious from design guidelines. For example, if we take Shneiderman's eight golden rules (Shneiderman 1998):

1. strive for consistency
2. enable frequent users to use shortcuts
3. offer informative feedback
4. design dialogs to yield closure
5. offer error prevention and simple error handling
6. permit easy reversal of actions
7. support internal locus of control
8. reduce short-term memory load

we see that they concur to helping users understand the application, of course, but they may not be sufficient to help designers communicate efficiently the intellectual value added in software tools. Acrobat's interface (with respect to our example) can be said to follow the eight golden rules, but yet it fails to *tell* the users about some relevant features of this technology regarding the various reviewing strategies that users can adopt. So, what do we mean by "telling users about the design vision"?

By shifting the design goal statement from producing to introducing technology, we can approach the answer. Introducing technology aims at making adopters understand valuable strategic aspects of it, and not only learning how to operate the system. Users must be told how technology can add value to their work and activities. Note that the eight golden rules refer to operational aspects: to what kinds of actions the user must perform, and the resources he needs in order to interact smoothly with the system. They do not refer to the strategic aspects of the technology, as for example to the relative advantages of choosing one interactive path over other possible ones. In the Acrobat example we see that users can easily achieve their reviewing goals using the highlight-and-add-note strategy (which can be said to comply with Shneiderman's golden rules). But they would very probably be much more efficient (and would prefer to be so) if they were aware of the comment-on-highlight strategy that can be adopted by changing preference parameters.

One could argue that changing the label of the "show comment sequence number" parameter to something like "show comment icons and numbers" might improve the interface and evoke the strategic value of this alternative mode of operation. But this line of solution is not good enough, for two reasons. One is that it is episodic and ad hoc, a kind of solution that depends on the designer's capacity to foresee when and why it should be extended to other portions of the interface. In other words, by keeping with the goal of producing usable interfaces, and using operational usability metrics to assess the quality of the interface, a designer is not prompted to think systematically about the strategic decisions of the users. The other reason is that only by replacing the overall HCI design goal with one of introducing technology, strategic aspects naturally come first. If one cannot see why one should learn a new technology, one can easily choose not to learn it at all. Or, as seems to be the case with Acrobat, if one doesn't see that certain operations open the avenue for alternative strategies that can be considerably more efficient than others, the value of the technology is not fully understood.

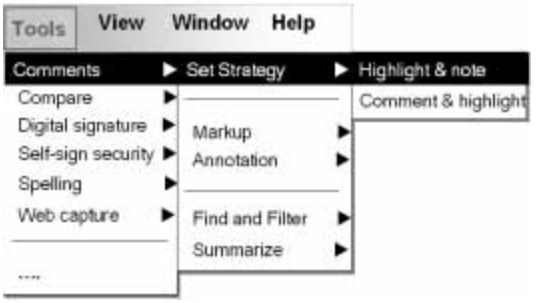

Figure 1.10
Communicating strategies to a user—a redesign of Acrobat®'s original menu structure.

So, by shifting focus in stating design goals, we are simultaneously shifting communication priorities in the interface. Strategies come first, then operations. This does not necessarily mean that we must have a two-stage communication structure in the interface, first telling users everything about the strategies and then, depending on their choice, telling them all about operations. Neither does it mean that interfaces must be verbose and loaded with text and explanations. Semiotics can be used to explore how we can convey both using one and the same sign structure. Just as an example of how strategies can be communicated along with operations, consider the proposed redesign of Acrobat's Tools menu structure, shown in figure 1.10.

Communicating strategies of use is an important factor in making users understand the design vision. The kind of communication suggested in figure 1.10 would benefit other system's interfaces such as Treemap's and WinVi's, illustrated earlier in this chapter.

1.5 The Contribution of Semiotic Engineering

What effects does a semiotic perspective bring about for HCI research and practice, and what is the value of semiotic theories in this discipline? To answer these questions we must analyze how semiotic engineering relates to the other theories—is it detrimental to others? is it prone to being complemented or supplemented by others? In the following subsections we will discuss

- the semiotic engineering homogeneous model for users' and designers' activities in HCI;
- its ontological, epistemological, and methodological constraints and commitments; and
- the consequences of having homogeneous models.

1.5.1 A Homogeneous Model for Users' and Designers' Activities in HCI

Semiotic engineering views HCI as a particular case of computer-mediated human interaction. This perspective is invariant across most semiotic approaches to HCI (e.g., Kammersgaard 1988; Nadin 1988; Andersen 1997), but the specificity of semiotic engineering is to characterize interactive computer applications as metacommunication artifacts (de Souza 1993; Leite 1998). Metacommunication artifacts communicate a message about communication itself. The message is elaborated and composed by designers and intended for the users. This is the first stance in which HCI designers and users are brought together under the same communicative process. In order to decode and interpret the designers' message, users proceed to communicate with the message, which gradually unfolds to them all the meanings encoded in it by the designers.[1] The message actually speaks for the designers in the sense that it contains all the meanings and supports all meaning manipulations that the designers have rationally chosen to incorporate in the application in order to have it do what it has been designed to do. In semiotic engineering terms, the message serves as the designer's deputy, presenting not only an artifact that can perform a certain range of functions and be used within a certain range of contexts, but also the rationale and design principles that have been followed while synthesizing this product. Rational decisions and choices relative to meaning and meaning manipulation are the best evidence that interactive computer applications are intellectual in nature. And as is the case of any other intellectual object its meaning can only be fully grasped and understood when conveyed in an appropriate semiotic discourse—one in which basic metalinguistic functions are available for mutual communication.

The four-stage evolutionary schema of metacommunication involving designers and users in HCI involves the following steps:

- The designer studies users, their activities, and their environment;
- The designer expresses, in the form of computer technology, his views about how the users, their activities and environment may or must change (because users so wish);

- Users unfold the designer's message through interacting with the system; and
- Users finally make full sense of the designer's message and respond to it accordingly.

The whole process thus starts with the designer studying the users, their activity, and their environment. He takes into account ergonomic, cognitive, social, and cultural factors that can affect usability and adoption of the technology being built. He consolidates all this knowledge into his interpretation of who the users are, what they need, what they like and prefer, and so on. The second stage in the process is an articulation of the knowledge gained in the first stage with the technical knowledge and creativity of the designer, who then produces an artifact that tells his mind in the form of an interactive message. The third stage corresponds to the user's unfolding of the designer's message, gradually making sense of the various meanings encoded in the artifact. The fourth and last stage in this particular type of computer-mediated human communication corresponds to the user's final elaboration of the meanings unfolded during interaction, a stage when the user finally makes full sense of the message. At this stage the user's response to the technology is mature and complete. The user knows the technology, no matter if he is fully satisfied with it or not. Semiotic engineering focuses on optimal communication, not on other crucial aspects of design such as aesthetics, productivity, and user satisfaction, to name but a few.

The metacommunication artifact is thus a designer-to-user message, whose meaning can be roughly paraphrased in the following textual schema:

> Here is my understanding of who you are, what I've learned you want or need to do, in which preferred ways, and why. This is the system that I have therefore designed for you, and this is the way you can or should use it in order to fulfill a range of purposes that fall within this vision.

Notice that this overall content of the designers' message to users holds true in single-user and multi-user applications, stationary and mobile applications, fixed and extensible applications, Web-based or non-Web-based applications, basically visual or basically textual applications, virtual reality, work and fun applications, and so on. It is a robust characterization of HCI across media, domains, and technology, which is a promising feature for a prospect foundational theory.

1.5.2 Ontological, Epistemological, and Methodological Constraints and Commitments

By underwriting a semiotic approach, the semiotic engineering of human–computer interaction is bound to live with certain ontological, epistemological and methodological constraints and commitments that may or may not exist in other alternative theories. We should start this discussion by committing to a definition of semiotics itself. Our choice is to adopt Eco's definition (Eco 1976, 1984), according to which semiotics is the discipline devoted to investigating signification and communication. Signification is the process through which certain systems of signs are established by virtue of social and cultural conventions adopted by the users, who are the interpreters and producers of such signs. Communication is the process through which, for a variety of purposes, sign producers (i.e., signification system users in this specific role) choose to express intended meanings by exploring the possibilities of existing signification systems or, occasionally, by resorting to non-systematized signs, which they invent or use in unpredicted ways.

Eco's definition of semiotics can easily justify why HCI is legitimately an object for semiotic investigation. It also supports the prevalent view in semiotic approaches to HCI, namely, that HCI is a particular case of computer-mediated human communication. All interactive computer-based artifacts are meant to express a design vision according to an engineered signification system. And it finally reveals the nuances of the semiotic engineering notion of metacommunication artifacts by helping us see that the signification system used by designers to communicate their vision to users must become instrumental for the users to express their particular intent and content back to the application (lest interaction becomes impossible).

But Eco, as many other leading semioticians, fully embraces the fundamental notion of unlimited semiosis, proposed by the American nineteenth-century philosopher and co-founder of contemporary semiotics, Charles Sanders Peirce. This notion rests on the idea that a sign is not just a relational structure—binding together a representation, an object that warrants it the status of being a representation, and a mental interpretation triggered by it. It is a recursive function which can be applied to itself any number of times, yielding interpretations that become the argument of the next function application. The recursive process of its application is called unlimited semiosis.

In spite of its apparently convoluted formulation, the meaning of unlimited semiosis is relatively easy to grasp. In practice, it amounts to a number of interesting formulations of intuitive facts involved in human communication. It says that the

meaning of a representation cannot be defined as a fixed object or class of objects, although, for there being a representation at all, there must be an object or class of objects that requires a representation. For example, for the word *HCI* to be a representation (at all) there must be an object or class of objects that directly determine its codification into a signification system. In the absence of these, nothing can be a representation "of" anything. The existence of a mental stance directly determined by the presence of a representation, but only indirectly determined by the presence of an object or class of objects, opens the possibility of subjective (internal) ingredients to interplay with objective (external) ones, generating arbitrary interpretations for any given representation. So, there are three components to a sign: representation, object, and interpretation (often called *meaning*). For mutual understanding to be possible, there must be (a) regulatory mechanisms that can determine if and when shared meanings have been achieved, and (b) generative mechanisms that can produce other mental stances of meaning when the first triadic configuration of semiosis proves to be insufficient for achieving shared meanings. The principle of unlimited semiosis states that in communication, for instance, semiosis will continue for as long as communicating parties are actively engaged in mutual understanding. Because there is no way to predict the length of this process, and because in theory it can last for as long as one cares to carry it on, semiosis is unlimited (although it is finite on an individual scale, given the finitude of our existence). The same can occur in noncommunicational semiosis, when the human mind ponders reflectively over the meaning of things. This is the very nature of philosophical activity, and by definition it can extend to any length of time, even beyond the limits of an individual's life span, if we consider schools of thought to behave as a collective mind. Culture, ideology, archetypal myths are yet other instances of unlimited semiosis in action.

So, a semiotic theory of HCI, such as the one proposed by semiotic engineering, has to incorporate signification, communication, and unlimited semiosis in its ontology, no matter how uncomfortable the consequences of this move in epistemological and methodological terms.

At the ontological level, semiotic engineering splits meaning into two very diverse categories. Human meanings (that of designers and users) are produced and interpreted in accordance with the unlimited semiosis principle. Computer meanings (human meanings encoded in programs), however, cannot be so produced and interpreted. For a computer program, the meaning of a "symbol" (a kind of degraded sign, in this context) is no more than an object, or class of objects, encoded into

another symbol system (or directly into a physical state of a physical device). Theoretical constraints on algorithmic computation cannot support the notion of unlimited semiosis (Nake and Grabowski 2001) and thus set the borderline between human and computer semiotics (see section 7.1 for an extensive discussion of this issue). Moreover, computer interpretation is itself a sign of human interpretations about meaning and meaning-related processes. The way programs assign meanings to symbols is a reflection, or representation, of how programmers believe meanings are logically assigned to symbols.

At the epistemological level, some important constraints must be spelled out. First, in a semiotic perspective, there is no room for a purely objective and stable account of meaning, whether the designers' or the users'. Meaning carries inherent subjective and evolutionary ingredients determined by unlimited semiosis that cast a shadow of doubt upon the idea that the users' context, requirements, and capacities can be fully captured by any human interpreter at any given time. Even when we let go of the idea of capturing all meanings, the choice of which are the relevant ones is equally prone to misjudgment and incompleteness. Consequently, in this sort of epistemological context, a researcher cannot possibly assume a positivist attitude, commonly adopted by many who aim to build predictive theories. From a semiotic perspective one cannot observe and interpret empirical phenomena without being affected by his or her own subjectivity, and the sociocultural context around him or her. Therefore, the value of observation and interpretation cannot be dissociated from the purposes the empirical study is designed to serve in the first place.

In order to preserve the theory from being declared anarchic and ultimately useless, for scientific purposes in general and for HCI design in particular, this epistemological challenge must be counterbalanced by sound methodological choices. They must enable researchers to identify the limits of subjectivity, the power of cultural determination, the conditions of social contracts in communicative phenomena, and, in the particular domain of HCI, the effects of computer technology upon human signification and communication processes. Sound choices will protect researchers and professional practitioners against adopting a naïve idealized view in which interactive computer-based artifacts can be built according to laws that are similar to the ones that allow civil engineers to build bridges. And they will also keep them from adopting a nihilist view in that, because users can never be fully understood and systems must always be built, any approximation of the actual use situation is as good as any other. One of the peculiar aspects of this theory is that

it carries in itself a certain ethics of design, which other theories don't explicitly discuss or assume to exist.

In sum, semiotic engineering operates on a homogeneous continuum of analysis, where designers, users, and computer artifacts have a role in an overarching communicative process. The unit of analysis is a metacommunication artifact—an interactive computer-based system engineered to communicate to its users the rationale behind it and the interactive principles through which the artifact can itself be used to help users achieve a certain range of goals. However, the adoption of a semiotic ontology of HCI calls for strict epistemological and methodological vigilance. We believe that this situation pays off for two main reasons. One is that the range of phenomena encompassed by a semiotic perspective is considerably wider than that covered by the main theoretical alternatives backing up HCI today. It suffices to mention, as a justification, that none of them brings together designers, computer artifacts and users under the same observable unit of investigation. The other is that we cannot be sure, given the lack of explicit statements about epistemological and methodological commitments in research work produced by alternative theories, that similar challenges would not be faced by cognitive, ethnographic, psychosocial, ergonomic, or purely computational traditions in HCI research. Therefore, it may well be the case that a stricter vigilance of this sort may yield novel and important knowledge about the nature of HCI, with positive impacts for both research and professional practice.

1.5.3 The Consequences of Having Homogeneous Models

A homogeneous continuum of analysis in HCI brings about a certain number of consequences. First, as we have already mentioned, because users and designers are involved in the same communicative phenomenon, semiotic engineering becomes a reflective theory. In other words, if designers use semiotic engineering to build interactive applications, they must think about their own role and intent in HCI, and about how their role and intent affect and are affected by those of the users. The theory is well-suited for problem situations in which introducing interactive technology seems more important or more precisely the case than producing it.

Second, this homogeneity brings about different possibilities not only for interpreting design problems, but also for structuring the solution space and exploring the comparative value of alternative solutions. These possibilities stem from our knowledge about signification and communication phenomena, and their relationship with culture and computer technology. As a consequence, a number of

constraints and concerns stand out in the design process. For example, when HCI design is cast as signification system engineering, we realize that this system must be used in three different situations. To begin, in order to communicate with users via the designer's deputy, designers themselves must use it. This is the context of the one-shot unidirectional communication voiced by the interactive discourse possibilities programmed into the application. It must also be used by the designer's deputy (or "system"), which will concretely interact with the user, systematically generating and interpreting signs according to the semiotic rules embedded in the program. And finally it must be learned and used by the users, who can only come to understand, apply, and enjoy a computer application if its signification system can effectively support communication.

Third, design intent is brought forth as a first-class citizen in this scenario. Because designers are actually communicating to users, the purpose of communication must be achieved. Two consequences follow from this. One is that designers must have a clear idea of what they want to do and why. The other is that they must be able to tell this to users in ways that are computable, easy to learn and remember, efficient and effective. Note that they are not talking about telling the users which button to press to activate this and that function, or what is the meaning of this and that visual form on the screen. They are telling them why the application makes sense to them, and why they expect that it will make sense to users, too. One of the leading reasons why designers expect that users will be pleased with the application, and make an effort to learn the specifics of it, is that they know who users are and have designed the application especially for them. If designers can't tell this—that is, if they don't know who the users are and have designed an application that has some inherent qualities that they expect will be intuitively perceived and naturally approved—they must still acknowledge that these qualities are there and help the users appropriate such qualities.

Fourth, a homogeneous communicative framework for HCI where designers, users, and computer artifacts are brought together can be used to inspect the nature and the quality of design processes and products. The latter can be further analyzed both per se and in situ—that is, outside and inside the contexts of use. Since communication is such a pervasive activity in this semiotic perspective, we have the tools to answer a number of questions. For instance, we can analyze the following:

- What signs are being used to compose the signification system that will carry the one-shot message to users, and also support ongoing communication between the user and the designer's deputy?

- How does the signification system encode design intent? Is this encoding likely to be understood by the users? Why?
- Once the users understand design intent, how can they put this intent into operation for their own benefit?
- What kinds of semiosic processes triggered by the signs that compose the engineered signification system does the designer anticipate? Which ones can the designer's deputy process? Which ones can it not?
- If the users' semiosic processes evolve in directions that have not been anticipated by the designer, which signs will express this communicative breakdown? What remedial resources can be used to reestablish productive communication? Can users extend the signification system? How?

Finally, by including designers in the unit of investigation of HCI-related phenomena, semiotic engineering opens the door to an examination of its own limitations. In the process of analyzing this specific kind of computer-mediated human communication, designers may realize that they need tools, knowledge, or other resources that this theory is not able to provide. And as soon as they realize it, they can look for other theories that will complement semiotic engineering (i.e., provide the additions that will lead to a *complete* fulfillment of needs), supplement it (i.e., provide additions that will lead to an *increased* fulfillment of needs), or simply replace it (because others can yield better results altogether).

1.6 Expected Contributions for Professional Practice

Novel theoretical approaches to HCI have been repeatedly called for to advance research and contribute to improve the quality of information technology artifacts (see, e.g., opinions expressed in Barnard et al. 2000; Hollan, Hutchins, and Kirsh 2000; Shneiderman et al. 2002; Sutcliffe 2000). As we have already mentioned, however, the latter has been the object of some debate, since studies have reported that practitioners disagree with the view that theoretical research can have positive and timely impacts on the quality of HCI product design (e.g., Gugerty [1993]).

Donald Schön (1983) reviewed and critiqued alternative views on the relationship between scientific and practical knowledge, and proposed that there are two paradigmatic positions in this field. One is the technical rationality perspective, according to which practical knowledge is applied science. In this view, technical

professionals should be trained in basic sciences that provide them with generally applicable problem-solving methods, and then develop the necessary skills to select and apply the best-fit method to the problems they are likely to encounter (Simon 1981). The other is a reflection-in-action perspective, according to which practical knowledge involves the capacity to name the elements present in problematic situations and frame them as problems. These problems are often unique and usually unstable, unlike the general kinds of problems assumed to exist in the other paradigm. In the reflection-in-action view, technical professionals must be equipped with epistemological tools that can help them raise useful hypotheses, experiment with different candidate solutions and evaluate results. As a consequence, Schön suggests that the education of professional designers can benefit from more epistemic approaches to practice. Schön's message to HCI designers in particular (Schön and Bennett 1996), is that they should ask themselves what is the artifact they are about to design, and not only how they can make the artifact usable. The answer to the first question, as in the case of engineering models, for example, can be found by "interacting with [a] model, getting surprising results, trying to make sense of the results, and then inventing new strategies of action on the basis of the new interpretation" (181).

If we look at HCI research over the last decade or so, we see that a very large portion of the results it has produced falls in the category of predictive methods or explanatory models of users' behavior (Shneiderman et al. 2002). The products of research have typically taken the form of design guidelines and heuristic rules. Tacitly assuming that designers are faced with recurring problem types, many researchers struggle to articulate guidelines and rules into generalizable principles. Principles, in their turn, can be used to support the methods with which design problems can be skillfully resolved. Perhaps the most enduring instance of how theories can be used in this direction is Card, Moran, and Newell's model (1983) of the human information processor. For nearly two decades, this theory has influenced a considerable portion of HCI designers who view interaction as a cognitive problem whose solution can be found with the application of appropriate task and user modeling techniques derived from the original theory or some of its derivatives.

There are, however, some problems with the technical rationality perspective. Experimental validation of the methods derived from the various theoretical stances is difficult if not impossible to carry out. Keeping rigorous control over statistically relevant corpora of design situations in current HCI professional practice would not only involve enormous costs (having different design teams developing the same

"real world" application by using different comparable methods; tracing every step of their progress; and performing statistically significant product evaluation tests in the end), but involve methodologically dubious procedures (deciding whether an implemented program strictly follows a design specification; deciding whether a particular design specification represents all dimensions of design intent and content).

Another kind of difficulty with technical rationality is that the purposes and requirements that justify and trigger the design of various interactive software artifacts are constantly changing. Many HCI practitioners believe that users don't know what they want. But this isn't necessarily the case. Users may know very well what they want and yet change their minds every time they are asked about requirements and purposes. As everybody else, the more they think about the problem, the more they understand about it. So, new insights are constantly arising from their reflection. This situation makes it again very difficult to practice HCI design as applied science, because the problem to apply it to is intrinsically unstable (and not amenable to the kinds of generalizations targeted by technical rationality).

In spite of these difficulties, and the resonance of Schön's ideas in a number of HCI research centers, researchers have not turned to epistemic theories of HCI, or to producing epistemic tools. An epistemic tool is one that is not used to yield directly the answer to the problem, but to increase the problem-solver's understanding of the problem itself and the implications it brings about. In the same spirit as Kirsh and Maglio's account of epistemic actions performed by Tetris® players (1995), epistemic design tools are those that will not necessarily address the problem solution, but the problem's space and nature, and the constraints on candidate solutions for it. Suchman's ideas (1987) and Winograd and Flores's account (1986) both called attention to the fact that intelligent human behavior uses freely exploratory interpretation of signs belonging to the context of activity in order to take goal-related action. And good computer tools should help people do this more and better.

Given the objectives that we set out to achieve with semiotic engineering, we can say that the main theoretic contribution we intend to make is to address the epistemic needs of both designers and users when communicating through or with computer-based technologies. With a semiotic perspective on HCI, we intend to open the road for eventually producing epistemic design tools that will help designers deal with 'knowledge' itself. The reflective nature of semiotic engineering is particularly suited for the task. It can be easily articulated with Schön's

reflection-in-action perspective on design, and can strive to compensate the difficulties brought about by the technical rationality perspective by raising the designers' epistemological awareness on what they are doing and on how this affects what users are doing.

This book is organized in three parts. Part I offers an introductory overview of our goals and assumptions, followed by a concise presentation of some theoretical concepts used in semiotics that are necessary for understanding semiotic engineering. It ends with a detailed presentation of our theory of HCI. Part II shows how the theory can be applied to inform the evaluation and design of HCI products in three specific contexts. I start with communicability evaluation and the construction of online help systems. Then I explore customization and extension of applications in the context of end-user programming. And I conclude the second part of the book with a discussion of the semiotic engineering of multi-user applications. Part III contains a single chapter, in which I share with readers my questions, beliefs and expectations about the potential and the immediate opportunities for research in semiotic engineering. Because the theory is itself a sign that readers may add to their ongoing semiosis, there is no ultimate conclusion or prediction about what semiotic engineering means. This book is only my current discourse about what it means to me, hoping that it will be a useful and stimulating reading for those who like to think about HCI.