An earlier version of this paper without figures appears in *TOOLS 10: Technology of Object-Oriented Languages and Systems*, edited by Boris Magnusson, Bertrand Meyer and Jean-Francois Perrot, Prentice-Hall, 1993, pages 5–10.

On Notation[1]

Joseph A. Goguen
Department of Computer Science and Engineering
University of California at San Diego
La Jolla CA 92093-0114

Abstract: It has often been noted that discussions among computer scientists about notation can be more heated and protracted than discussions about issues usually considered more substantive. This suggests there may be more to notation than is admitted in phrases like "mere syntax" and "syntactic sugar." Building on C.S. Peirce's classification of signs into icon, index and symbol, I argue for the importance of notation, and in particular, for the importance of diagrams and other visual forms of presentation in addition to purely textual presentations. I also suggest some criteria for evaluating the quality of notation. These considerations are illustrated by notation from Euclidean geometry, programming languages, arithmetic, algebra, and formal methods; applications to databases and Computer Supported Cooperative Work are also mentioned, as are connections with recent work in philosophy and sociology, including postmodernism and ethnomethodology.

1 Introduction

The goal of this paper is to begin exploring what makes one notation better than another for the same purpose. I use the word "notation" in a broad sense that includes screen layout, "icons," color, motion and even interaction, as well as the more conventional linear syntax, keywords, etc. We examine particular notational systems, including some standard features of programming languages, mathematics, and design notations.

It should be evident that this is not purely a problem in computer science, because it involves human beings in an essential way. It may be less evident that it is not purely a problem in experimental psychology; to demonstrate this, we first discuss some ideas from Charles Saunders Peirce's semiotics, and then show that social issues are involved in an essential way.

This motivates our giving a brief survey of sociology, which suggests there may be fatal flaws in the classical normative and statistical approaches, while the experimental approaches of individual and social psychology are only a little better. However, a new and rather radical approach called *ethnomethodology* seems promising, and has been applied to diverse problems.

1.1 The Importance of Notation

Some computer scientists feel that notation is essentially arbitrary, and that what really matters is the semantic content, rather than the syntactic window dressing. However, even a relatively slight experience with a difficult technical paper employing unnecessarily opaque notation should incline most people towards a contrary opinion. Other salutary experiences might include trying to use a software tool

[1]The first version of this paper was written while the author was at the Programming Research Group, of the Oxford University Computing Lab, and was supported in part by grants from the Science and Engineering Research Council, and contracts with British Telecom, Fujitsu Laboratories Limited, and the Information Technology Promotion Agency, Japan, as part of the R & D of Basic Technology for Future Industries "New Models for Software Architecture" project sponsored by NEDO (New Energy and Industrial Technology Development Organization).

with a bad interface design (e.g., a debugger, editor, or mailer); and many languages for programming, specification and requirements can also provide instructive experiences. Good notation makes the difference between a *transparent* interaction, in which the actual problem dominates the user's attention, and a nightmare, in which the user cannot get the system to do what he wants, and cannot understand what the system "thinks it's doing."

Some computer scientists (perhaps most famously, Edsgar Dijkstra) feel that any use of diagramatic or verbal reasoning is bad, because it enables, and even encourages, sloppy intuitive thinking, as opposed to precise, linear, logical reasoning[2]. "The moral is clear: for doing high-quality mathematics effectively, verbal and pictoral reasoning have to be replaced by calculational reasoning" [4]. I argue that such views are profoundly wrong, and even harmful; they represent an extreme form of modernism that calls forth postmodernism as a counter-movement. In particular, such views may have significantly impeded acceptance by a larger community of much of the work done in formal methods.

2 Aspects of Peirce's Semiotics

First, let us distinguish between signals and signs: *signals* (sometimes also called *tokens*) are configurations of matter and/or energy that do not necessarily signify anything, whereas *signs* involve communication and have meaning or significance. In computer science, we might distinguish between data and information (or "knowledge") in a similar way. For Peirce, a sign has three aspects: the first is the form of the sign, which he called the **representatamen**; the second is the **interpretant**, which is the connection between the representatamen and the **object** of the sign, which is its referent, which may be a physical object, but also may be another sign.

Peirce divided signs into icons, indices, and symbols, according to the manner in which they signify. He defined an *icon* as a "sign which refers to the Object that it denotes merely by virtue of characters of its own ... such as a lead-pencil streak representing a geometrical line." In contrast, a sign x is an *index* for an object y if x and y are regularly connected, in the sense "that always or usually when there is an x, there is also a y in some more or less exactly specifiable spatio-temporal relation to the x in question" [3]. "Such, for instance, is a piece of mould with a bullet-hole in it as sign of a shot" [20]. In this example, the spatio-temporal relation is a causal one, and applies with great generality. However some indices only work in certain particular spatio-temporal contexts, such as proper names for persons. Finally, Peirce defines a *symbol* as a "sign which is constituted a sign merely or mainly by the fact that it is used and understood as such."

For purposes of design, other things being equal, there is a natural ordering to these three kinds of sign: icons are better than indices, and indices are better than symbols. However, things are *not* always equal. For example, base 1 notation for natural numbers is iconic, e.g., 4 is represented as `||||`, 3 as `|||`, and we get their sum just by copying,

```
|||| + ||| = ||||||| ,
```

so that addition is also iconic. But base one notation is very inconvenient for representing large numbers, and very inefficient for multiplication.

With Arabic numerals, the use of 1 for "one" is iconic (one stroke), but the others are symbolic; using the blank character for "zero" would be iconic, but of course this would undermine the positional aspect of decimal notation. Chinese notation for several of the small numerals is iconic.

Peirce's classes overlap, so some signs will be hard to classify. Also, complex situations will involve all three kinds of sign, interacting in complex ways; indeed, different aspects of the same sign may be iconic, indexical, and symbolic. It may be necessary to consider the context of a sign, e.g., how is it used in practice, and its relation to other signs in the same system. Also, it is sometimes necessary to interpret the notion of sign in a very broad way. Peirce's theory of signs is not a simple *representational theory*

[2]Prof. Dijkstra is also famous for walking out of lectures as soon as an intuitive visual diagram appears.

of meaning, in which a sign has a denotation; the interpretant makes it a *relational theory of meaning* instead; indeed, representational theories of meaning are increasingly under attack in contemporary philosophy, despite their broad acceptance in computer science.

2.1 Further Examples

When an operation like + is associative, it is usual to omit parentheses; thus we write $a+b+c$ instead of $(a+b)+c$ or $a+(b+c)$; this becomes ever more convenient for larger expressions, such as we might see in doing our income tax:

22.34 + 148.89 + 18.23 + 68.40 + 25.92

Dropping parentheses is *iconic* of the fact that it doesn't matter where they are; spread sheets also exploit this. Using 0 for the identity of addition is only symbolic, but using 1 for the identity of multiplication is indexical.

Unique identifiers in databases and variable names in programs are symbolic. Actually, the latter case can be more subtle if a variable name is a word (or phrase) in a natural language, because then the connection to the denotation goes through two levels: the connection between the variable and the word is indexical, while the connection between the word and its denotation is (usually) symbolic.

Most signs called "icons" in graphical computer interfaces are *not* icons in the Peircean sense. For example, the use of a square containing a feather (to suggest a quill pen) for an editor is not iconic, but rather indexical. On the other hand, *moving* a file into a garbage can is iconic. However, using a garbage can to indicate *deletion* is only indexical.

Let's consider file system navigation using the UNIX text-based command line format. Here are some commands, showing a user struggling to find the directory where some reports are stored:

```
cd  papers
cd  work/papers
cd  reports
cd  ~/reports
cd  ~/work/reports
```

This representation of navigation in a file system is incomplete, (partially) symbolic, and confusing. By contrast, the diagrammatic form is iconic and much easier to use (unless perhaps the file system is really huge), as shown in Figure 1, where navigation can be accomplished just by mouse click.

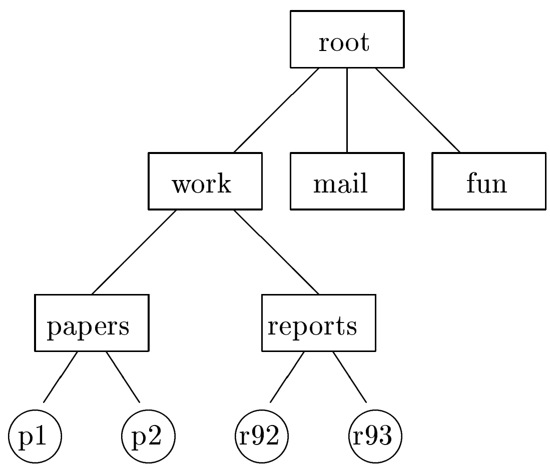

Figure 1: A File System

Classical Euclidean plane geometry is profoundly iconic; first, its constructions are iconic, as with dropping a perpendicular or bisecting an angle. Hence its proofs are generally iconic: you can see the truth of the result by carrying out the steps of the construction. On the other hand, proofs in algebra are almost never iconic, and the same holds for algebraic notation. But as discussed above, some algebraic notation is indexical, and some is even iconic, although most is symbolic. Perhaps mathematics could only get started through the iconic notation of its earliest achievements. And certainly hiding what iconicity exists is harmful to students trying to learn mathematics.

Many programming languages have the convention of structuring programs by lines that are numbered sequentially and that are normally executed in sequential order; this convention is iconic. For example,

```
1    X := X + Y
2    Y := Y - 1
3    IF Y .GT. 0 GOTO 1
```

On the other hand, the characteristic language features of so called structured programming, such as `do_while_od` and `if_then_else_fi`, that are used to replace line-oriented programming and `GOTO`s, are purely symbolic, as in

```
do X := X + Y; Y := Y - 1 while Y > 0 od
```

This suggests an explanation for the preference that many engineers and scientists still have for FORTRAN over more modern languages: so-called structured programing loses iconicity. But of course the `GOTO` itself is only symbolic and its extravagant use can be confusing.

2.2 The Social Nature of Iconicity and Indexicality

For a design to successfully use smoke as indexical for fire (say in a display for a building's fire alarm system) requires that the association between smoke and fire is known by the social group that includes the likely operators of the system. Similarly, we need to know how garbage cans are used, not merely what they look like, in order to use one indexically for file deletion. The necessity for socially shared knowledge of characteristics also applies to iconicity. The social character for symbolic signs is perhaps the most obvious: we need to know the convention before we can use it.

3 Sociology

In brief, most work in sociology and psychology is (what is called) "normative", in that it assumes certain category systems as already given (e.g., class, race, gender, intelligence) and then tries to determine which category applies to a certain case, often using statistical techniques. But someone seeking to design a system that is easy to use should not just assume that certain category systems will be best for users; bitter – and often expensive – experience shows that such assumptions are frequently wrong; rather, the designer should understand what category systems users already have, and how they make use of them in practical action. (See [21] for an ethnomethodological explication of category systems.)

An excellent case study of an expert system for advising on the use of a photocopy machine has been done by Lucy Suchman [23]. She shows that certain categories, relationships, and methods assumed by the designer are not shared by users. Indeed, an overly sophisticated builtin help system can actually make it more difficult for users to cope when things go wrong.

Ethnomethodology was created by Harold Garfinkel [6] essentially as a criticism of classical normative sociology. Largely through the work of Harvey Sacks [22], ethnomethodology developed into conversation analysis, a highly empirical, detailed and precise theory of how conversation is organized, and in particular, of how turntaking is accomplished. More recently, interaction analysis has taken advantage of video technology to extend the principles of ethnomethodology to non-linguistic interaction [14]. In this tradition, analysts do not impose their own categories and methods to understand social interaction, but rather seek to use the same categories and methods that members themselves use to make sense of their interaction. A basic principle is that socially meaningful actions are exactly those for which members are held accountable by their social group. See [9] for further discussion.

Suchman [23] gives an excellent overview of ethnomethodology and conversation analysis, as well as a general critique of cognitivist modelling. Goguen and Linde [13] also give an exposition of ethnomethodology, along with a critical survey of other techniques from sociology, with a view to their applicability to Requirements Engineering. Eric Livingston [16, 17] gives an interesting discussion of the ethnomethodology of mathematics, with examples from plane geometry and metamathematics. My (former) research group at Oxford used ethnomethodology and interaction analysis to study requirements for equipment to support stock dealers, and for an ISDN multimedia system, and my research group at UCSD is applying semiotics to user interface design [11, 10, 12].

4 Graphical Notation

Bruno Latour, a pioneer in the sociology of science, has pointed out the tremendous importance of graphical representations in the historical advance of science and technology [15]. For example, maps played a critical role in the "Age of Exploration," during which European navigators charted the world, and opened it to trade and colonization. Some other examples, taken almost at random, include: Feynman diagrams in physics; scatter diagrams in statistics (and their many applications, e.g., manufacturing); the classical Greek use of diagrams for geometry; star charts in astronomy; isometric drawings in Engineering Design; architectural drawings; blueprints; family trees; the planar representation of complex numbers and its applications in electronics; commutative diagrams in algebra; and much much more. For example, there is no doubt that the graphical and interactive character of spread sheets explains their success; they do not enable any *new* calculations to be done, but rather their iconic and indexical representations make the usual calculations much more natural and convenient. No doubt the graphical notation of Petri nets also helps to explain their popularity.

All this sophistication is in sharp contrast to most notation currently used in the formal methods area of computer science, where new symbols and concepts are often introduced that are decidedly not shared by the community of practitioners, and that as a rule are purely symbolic. As I have worked in formal methods myself, I do not wish to say that mathematical notations are bad merely because they are unfamiliar to most programmers; rather, I wish to explain their unpopularity. In fact, I believe that programmers should learn more mathematics, but I also say that designers of notation for formal methods should make better use of graphics.

A successful example is the use of "boxes" in Z [1] to graphically indicate certain scoping conventions; this has been extended further in object oriented versions of Z, such as Object-Z [5] and OOZE [2], to allow nested scoping and module delimitation. This use of graphical notation is iconic, without compromising the underlying semantics of Z. Similarly, the comment convention used in Z is indexical.

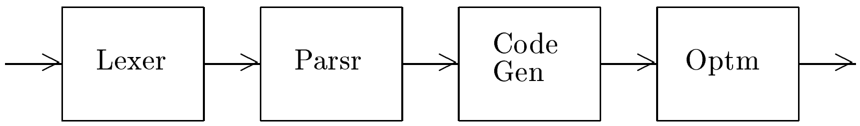

Figure 2: A Box Diagram

Informal, non-rigorous notation is often very useful in practice. One has only to think of the "black box" diagrams that are so widely used in system design, and the many exotic notations used in so-called structured analysis [19], including object oriented notations; e.g., see Figure 2. It is almost impossible to find a group of system analysts and/or programmers discussing a design without seeing them draw such diagrams. Indeed, I would go so far as to claim that constraining practitioners during early design to use some fixed notation with a fixed semantics would slow them down, by forcing them to pay more attention to the limitations of the notation than to the details of their problem. On the other hand, exactly such precision can facilitate communication at later phases of specification. I have claimed that the flexibility, informality and even the vagueness and ambiguity, of natural language and of informal graphical notations can actually be advantageous in the development of requirements [8].

4.1 Object Oriented Notation

Object oriented programming and specification can benefit from appropriate graphical notations, taking advantage of the intuitive nature of their basic concepts. For example, certain *direct manipulation* conventions allow mouse motion and clicks to iconically represent abstract data type operations, i.e., methods for objects, with no loss of mathematical rigour [7] (see Figure 3); of course, pens or joysticks would work just as well.

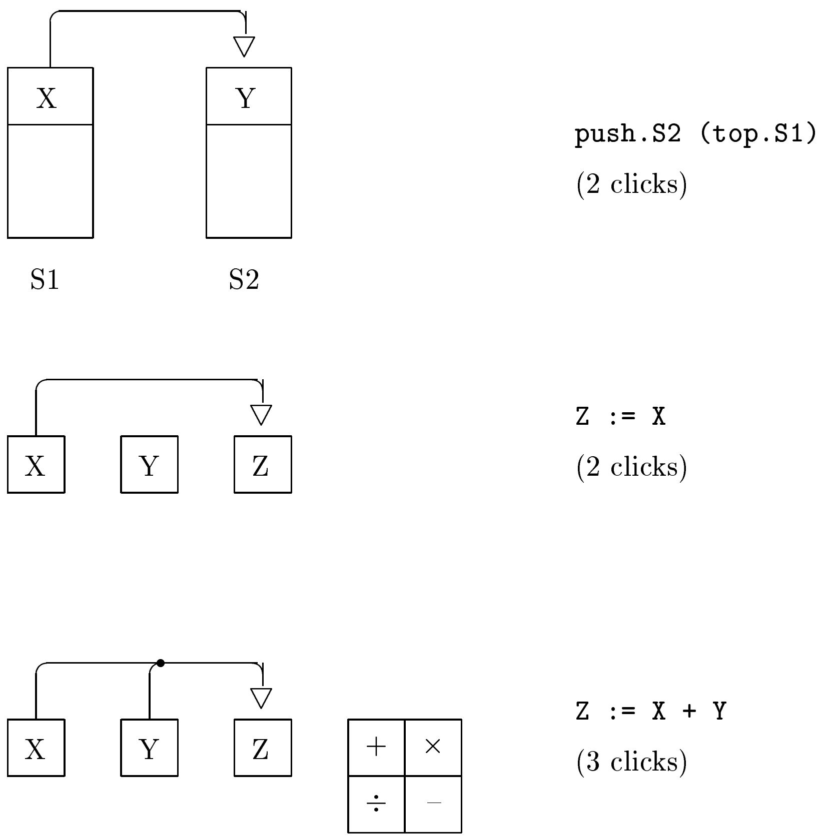

Figure 3: Direct Manipulation of Objects

In the top part of Figure 3, S1 and S2 are stacks and a single mouse movement with a left click to "pick up" X from S1 and a right click (or a left "unclick") to "put it down" onto S2. In the middle part of the figure, X, Y, Z are three "cells" and an "assignment" of X to Z can also be done with one mouse movement and two clicks. In the bottom part of the figure, two left clicks are needed to indicate a binary operation, plus a right click to indicate where to put the result, because + is the default binary operation; selecting a different operation would require one more click, in the popup operation menu. The mouse motion is iconic but the choice of left or right button to click is indexical (based on the convention of reading from left to right), while the use of "unclick" (letting go of the button after holding it down until the new location is reached) is again iconic.

The representation of class hierarchies by trees (usually "upside down" in computer science, with the root on top, as in Figure 4) is indexical of the ordering relationships involved, again in a mathematically rigorous way.

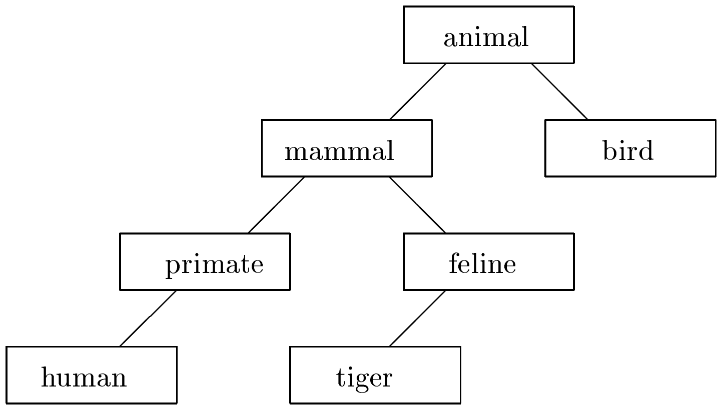

Figure 4: An Inheritance Diagram

4.2 Implications for Requirements and Computer Supported Cooperative Work

Taking a very broad view of signs, (what an ethnomethodologist might call) *constructively socially shared events* — i.e., events that are constructed to be apprehended by others, in such a way that everyone concerned can know that the others involved have also apprehended it — can be seen as *iconic*, because they are seen as what they are in virtue of certain displayed characteristics. For example, consider stock brokers in a cooperative environment speaking loudly into a phone so that his colleague will overhear him and know that a certain deal is being done in a certain way; the broker doing the deal also looks to ensure that his co-worker is actually apprehending this event.

One problem with older approaches to the computerization of work is that they assume that all interactions will be mediated through the computer system; they model workers as autonomous cognitive agents, who process symbols only through a computer terminal. But this model does not support constructively socially shared events: a worker can enter something into the computer, and thus change its state, but he cannot know at the same time whether some other worker has apprehended that change; this can only be determined by communication. However, exchanging messages through a computer can take much more time than exchanging glances, and this time can be critical in real-time applications, e.g., air traffic control, stock trading, etc.

This suggests research programs for requirements engineering and computer supported cooperative work based on techniques derived from ethnomethodology, using video data and interaction analysis to determine requirements for supporting constructively socially shared events.

5 A Philosophical Epilogue

Postmodernism is an important recent movement in philosophy, the arts, and other fields. It does not claim that we are now living in tomorrow or the day after, rather than the present. Rather, this movement seeks to characterize and criticize broad cultural tendencies called "modern." Unfortunately, this field has become so trendy that it has attracted a great deal of very dubious work; nevertheless, there is also some very good work, some of which is especially focused on Information Technology. I would particularly mention the work of Jean-François Lyotard [18], who says that modernism is characterized (among other things) by the use of "grand metanarratives" (i.e., all-inclusive stories, or explanatory systems) to "legitimate" (i.e., to justify the structure of) social systems. Examples include communism, socialism, capitalism, progressivism and liberalism, at the level of whole societies.

This tendency to include everything in one grand system is sometimes called *totalization*, and there are many examples of this tendency in modern computer science. One need only think of the tedious debates among proponents of various programming languages (or, with somewhat more sophistication, programming paradigms), where each side claims that it can do anything that the other can do. The same phenomenon can be found in debates about editors, operating systems, theorem provers, specification languages, semantic foundational formalisms, and many other areas of computer science. All of these can be seen as debates among different local cultures, or (what Lyotard calls, following the late Wittgenstein) different *local language games*. Ethnomethodology gives us techniques for exploring these in minute detail, and also acts as an antidote to the grand metanarratives.

Notation is only the surface reflection of these deeper, essentially social struggles. But perhaps it is time we realize that no metanarrative can be demonstrably superior to all others for all purposes, and that, in this sense, we live in many different worlds, rather than in just one world. In my view, this realization is the essence of postmodernism.

It follows that, as designers of languages and tools, we should try to discover the categories and methods of our user communities, and base our designs upon them, rather than trying to impose our own categories and methods on users from afar. On the other hand, no one should think that this will be easy, or that it is the only thing that needs to be done.

References

[1] Jean-Raymond Abrial, S.A. Schuman, and Bertrand Meyer. Specification language. In R.M. McKeag and A.M. MacNaughten, editors, *On the Construction of Programs: An Advanced Course*, pages 343–410. Cambridge, 1980.

[2] Antonio Alencar and Joseph Goguen. OOZE: An object-oriented Z environment. In Pierre America, editor, *European Conference on Object Oriented Programming*. Springer, 1991. Lecture Notes in Computer Science, Volume 512.

[3] William P. Alston. Sign and symbol. In Paul Edwards, editor, *Encyclopaedia of Philosophy, Volume 7*, pages 437–441. Macmillan, Free Press, 1967. In 8 volumes; republished 1972 in 4 books.

[4] Edsgar Wybe Dijkstra. On the economy of doing mathematics. In James Woodcock, Carroll Morgan, and Richard Bird, editors, *The Mathematics of Program Construction*. Springer, 1993.

[5] David Duke, Paul King, Gordon Rose, and Graeme Smith. The Object-Z specification language. Technical Report 91–1, Department of Computer Science, University of Queensland, 1991.

[6] Harold Garfinkel. *Studies in Ethnomethodology*. Prentice-Hall, 1967.

[7] Joseph Goguen. Graphical programming by generic example. In Steven Kartashev and Svetlana Kartashev, editors, *Proceedings, Second International Supercomputing Conference, Volume I*, pages 209–216. International Supercomputing Institute, Inc. (St. Petersburg FL), 1987.

[8] Joseph Goguen. The dry and the wet. In Eckhard Falkenberg, Colette Rolland, and El-Sayed Nasr-El-Dein El-Sayed, editors, *Information Systems Concepts*, pages 1–17. Elsevier North-Holland, 1992. Proceedings, IFIP Working Group 8.1 Conference (Alexandria, Egypt).

[9] Joseph Goguen. Towards a social, ethical theory of information. In Geoffrey Bowker, Leigh Star, William Turner, and Les Gasser, editors, *Social Science, Technical Systems and Cooperative Work: Beyond the Great Divide*, pages 27–56. Erlbaum, 1997.

[10] Joseph Goguen. An introduction to algebraic semiotics, with applications to user interface design. In Chrystopher Nehaniv, editor, *Computation for Metaphors, Analogy and Agents*, pages 242–291. Springer, 1999. Lecture Notes in Artificial Intelligence, Volume 1562.

[11] Joseph Goguen. Social and semiotic analyses for theorem prover user interface design. *Formal Aspects of Computing*, 11:272–301, 1999. Special issue on user interfaces for theorem provers.

[12] Joseph Goguen, Kai Lin, Grigore Roşu, Akira Mori, and Bogdan Warinschi. An overview of the Tatami project. In *Cafe: An Industrial-Strength Algebraic Formal Method*, pages 61–78. Elsevier, 2000.

[13] Joseph Goguen and Charlotte Linde. Techniques for requirements elicitation. In Stephen Fickas and Anthony Finkelstein, editors, *Requirements Engineering '93*, pages 152–164. IEEE, 1993. Reprinted in *Software Requirements Engineering (Second Edition)*, ed. Richard Thayer and Merlin Dorfman, IEEE Computer Society, 1996.

[14] Adam Kendon. *Conducting Interaction: Patterns of Behavior in Focused Encounters*. Cambridge, 1990. Studies in Interactional Sociolinguistics Number 7.

[15] Bruno Latour. *Science in Action*. Open, 1987.

[16] Eric Livingston. *The Ethnomethodology of Mathematics*. Routledge & Kegan Paul, 1987.

[17] Eric Livingston. Cultures of proving. *Social Studies of Science*, 29(6):867–888, 1999.

[18] Jean-François Lyotard. *The Postmodern Condition: a Report on Knowledge*. Manchester, 1984. Theory and History of Literature, Volume 10.

[19] James Martin and Carma McClure. *Diagramming Techniques for Analysts and Programmers*. Prentice-Hall, 1985.

[20] Charles Saunders Peirce. *Collected Papers*. Harvard, 1965. In 6 volumes; see especially Volume 2: Elements of Logic.

[21] Harvey Sacks. On the analyzability of stories by children. In John Gumpertz and Del Hymes, editors, *Directions in Sociolinguistics*, pages 325–345. Holt, Rinehart and Winston, 1972.

[22] Harvey Sacks. *Lectures on Conversation*. Blackwell, 1992. Edited by Gail Jefferson.

[23] Lucy Suchman. *Plans and Situated Actions: The Problem of Human-machine Communication*. Cambridge, 1987.