

Oxford University Computing Laboratory
11 Keble Building
11 Keble Road
Oxford OX1 3QD

FOUR PIECES ON
ERROR, TRUTH AND REALITY

by

Joseph A. Goguen

ACQUISITION ✓	DATE 22 FEB 2002
OXFORD	
 303397003S	

Technical Monograph PRG-89

October 1990

Oxford University Computing Laboratory
Programming Research Group
11 Keble Road
Oxford OX1 3QD
England

Copyright © 1990 Joseph A. Goguen

Oxford University Computing Laboratory
Programming Research Group
11 Keble Road
Oxford OX1 3QD
England

Electronic mail: goguen@uk.ac.oxford.prg (JANET)
Electronic mail: goguen@prg.oxford.ac.uk (elsewhere)

Four Pieces on Error, Truth and Reality

Joseph A. Goguen¹

Preface

This monograph consists of four papers on social and philosophical aspects of computing. The first, third and fourth were written for the book *Software Development and Reality Construction*, which grew out of an interdisciplinary conference held in Schloß Eringerfeld, Germany, in September of 1988. The second was written as a position paper for the conference *Formal Methods 89*, which was held in Halifax, Nova Scotia in July of 1989.

The first paper is concerned with the role of errors in computing, and in particular, with the regrettable tendency within some schools of Formal Methods to claim that errors can and should play no role at all. This paper claims on the contrary that errors are inevitable, and that we must therefore develop ways to deal with them. It also claims that much of software production and of individual growth and learning necessarily occur in the context of misunderstandings and mistakes. Moreover, the necessity for dealing with errors can actually become a pleasure.

The second paper is largely concerned with philosophical aspects of Formal Methods, and in particular with the recent controversies about whether computing systems can be "proved correct," and indeed, with what we mean by "proved" and by "correct," and how such mathematical concepts connect with the real world. Such problems are important in the context of Safety Critical Systems, for example. In keeping with its origin as a position paper, some recommendations concerning research directions are given, and there are no references.

The third paper goes somewhat deeper into certain philosophical problems about meaning and truth. It contrasts the "modern" formalist position of the logical positivists like Carnap with the views of Heidegger and Wittgenstein. This has serious consequences for our understanding of correctness problems in computing.

The fourth paper takes us somewhat further afield. It is an attempt to connect the process of interpretation with the philosophy of Buddhist meditation. "Hermeneutics" is the study of interpretation, arising historically from problems in Biblical exegesis. It is relevant to attempts to understand computational artifacts in their social and personal contexts, whether the artifact is software, hardware, specification, requirement or documentation. In particular, philosophical hermeneutics are Buddhism are relevant to ethical issues in computing, and to the understanding of creativity in computing.

¹Also, SRI International, Menlo Park CA 94025.

Contents

The Denial of Error

1	Introduction	1
2	Science and Technology	2
3	Error-Free Programming	4
4	Software Quality	5
5	The Being of Software Development Projects	7
6	Conclusions	8

Formal Methods: A Position Paper

1	Introduction	11
2	What are Formal Methods?	11
2.1	Formal Methods and Mathematics	11
3	What Good are Formal Methods?	12
3.1	The Myth of Control	13
4	Hyperprogramming	14
5	Recommendations	15

Truth and Meaning beyond Formalism

1	Introduction	18
2	Heidegger, Carnap and Wittgenstein on Dread and "the Nothing"	18
3	What is "the Nothing"?	21
4	What are Truth and Meaning?	22
5	Where are we?	24

Hermeneutics and Path

1	Introduction	28
2	The Paramitas	28
3	Confusion	29

CONTENTS

4	Hermeneutics as Path	30
5	Hermeneutics in the Practice of Science	30
6	Emptiness and Beyond	31
7	Coloophon	32

The Denial of Error

Joseph A. Goguen

1 Introduction

This paper claims that the modern world has developed a kind of arrogance which is damaging the very projects that it seeks to sustain: in proposing methodologies to guarantee the absence of error, we deny the incredible richness of our own experience, in which confusion and error are often the seeds of creation; in this way, we limit our own creativity.

This arrogance is not an isolated phenomenon that is found only in computer science. Indeed, I claim that it arises in a natural way from our preoccupation with and immersion in science and technology, which are strongly oriented toward *control*. The obsession of Western culture with control can be seen in many different areas, including the following:

1. In *myth*; for example, if you know a demon's name, then you can control its behavior (we may relate this to the phrase "knowledge is power").
2. In *science*, which is based upon the idea of the controlled experiment (this is control of the intellectual process, rather than of its result).
3. In our theories of *behavior*; for example, the psychiatrist Ernest Becker has said that "All social life is the obsessive ritualization of control" [4]; see also point 5. below.
4. In *technology*, which seeks to control nature through the application of science, as discussed in more detail later in this paper.
5. In our theories of *information* and *knowledge*; for example, in the "Representational Theory of Meaning," which says that our minds contain representations of external "objects," or in current Cognitive Science theories which posit explicit goals to control behavior, in the same way for both machines and humans (see Section 2 below).

Aspects of the viewpoint common to these items have been called "instrumentality," "teleology," "rationalism," "selfishness," "objectivity," "analysis," "subjectivism," "ego," "positivism" and "conceptualism," depending on the author and the context. The obsession with control is also one aspect of what has come to be called "modernism".

The *denial of error*, that is, the denial of deviation from announced goals, seems to be closely associated with the attempt to maintain control, especially for phenomena that are actually difficult or even impossible, to control. For example, consider the economy of a country, especially one that is highly collectivized.

The history of science contains many instances of accidental discoveries, for example, that of penicillin. These are often taken as surprising, embarrassing, or amusing, but they actually point to a serious and important facet of scientific knowledge, indeed of all knowledge: its basis is the free play of the mind against the unexpectedly rich worlds revealed within each real situation. The following quotation from Heidegger [16] may be relevant:

The area, as it were, which opens in the interwovenness of being, unconcealment, and appearance – this area I understand as error. Appearance, deception, illusion, error stand in definite essential and dynamic relations which have long been misinterpreted by psychology and epistemology and which consequently, in our daily lives, we have wellnigh ceased to experience and recognize as powers.

Formalism is also a form of control: it attempts to control the use of language, and through that, to control behavior. The tighter and more rigorous the formalism, i.e., the more circumscribed its syntax and semantics, the smaller the domain to which it is applicable. The ultimate in this development may be the attempts of mathematical logic (e.g., Tarski [24]) to formally capture the notion of *Truth*; yet the manipulation of uninterpreted tautologies literally tells us nothing, about nothing (see [13] for some further discussion of meaning, truth and logic along these lines).

Section 2 below attempts to describe the essence of modern science and technology, loosely based on ideas of late Heidegger, and illustrated with some quotations from Bacon and Newell. Section 3 discusses the goal of error-free programming, using some work of Dijkstra as an example. Section 4 considers the goals of software quality, using U.S. Department of Defense procurement procedures as an illustration. Finally, Section 5 suggests that software development projects could be considered holistically, using some ideas from the so-called New Biology.

2 Science and Technology

At the dawn of modern science, Francis Bacon was obsessed with the concept of what we now call an experiment, using what now seem rather extreme metaphors of torture and the inquisition [1]:

... if any expert Minister of Nature shall encounter Matter by mainforce, vexing¹ and urging her with intent and purpose to reduce her to nothing; she contrariwise ... being thus caught in the straits of necessity, doth change and turn herself into diverse strange forms of things. ... the reason of which constraint or binding will be more facile and expedite, if matter be laid hold on by Manacles, that is by extremities.

Today, this language seems a bit shocking, and of course, no reputable contemporary scientist would want to sound quite so gleefully sadistic about his work. But perhaps we should give Bacon credit for a degree of honesty that has been lost to us, as the passage of time has dulled our sense of surprise at the methods of science and technology. For scientific experiments on animals can be quite gruesome, and technology has much to answer for in its destruction of the environment.

The fundamental problem here is not that there are some isolated, unfortunate incidents (e.g., strip mining in the Brazilian rainforest), nor even that there are potential massive dislocations looming on the horizon, such the effects of global warming and deforestation. Rather, the fundamental problem is that man has come to view nature as a "resource," as something to be used, for his convenience and comfort, or against his enemies, or to enhance his prestige through the acquisition of knowledge. As Heidegger [17] says,

¹ At the time of this translation, "vex" had much more the connotation of torture, from the Latin *vezo*.

The hydroelectric plant is not built into the Rhine River as was the old wooden bridge that joined bank with bank for hundreds of years. Rather, the river is dammed up into the power plant. What the river is now, namely, a water-power supplier, derives from the essence of the power station. In order that we may even remotely appreciate the monstrousness that reigns here, let us ponder for a moment the contrast that is spoken by the two titles: "The Rhine" as dammed up into the *power* works, and "The Rhine" as uttered by the *art* work, in Hölderlin's hymn by that name. But, it will be replied, the Rhine is still a river in the landscape, is it not? Perhaps. But how? In no other way than as an object on call for inspection by a tour group ordered there by the vacation industry.

In this way, we lose the capacity to be in the world with a sense of harmony, joy, or wonder.

The dark edge to science, so clear in the writing of Bacon, has to do with this fundamental alienation, that is, with man's will to what Bacon called "Dominion over the Universe," more than it has to do with the subject/object split, or with any particular difficulties. Bacon was as much the prophet of technology as he was of science. Let us listen to Heidegger [16] again:

Today science is admonished to serve the nation, and that is a very necessary and estimable demand², but it is too little and not the essential. The hidden will to refashion the essent into the manifestness of its being demands more. In order to recapture the pristine knowledge that has degenerated into science, our being-there must attain a very different metaphysical depth. It must again achieve an established and truly built relation to the being of the essent as a whole.

Let us now consider an example closer to home, from Artificial Intelligence. In [21], Allen Newell proposes a theory of mind based on what he calls a "physical symbol system," which is essentially an automaton, that is, a (mathematical) machine, intended to model the use of symbols. Newell claims that this notion is "the most fundamental contribution so far of Artificial Intelligence and Computer Science to the joint enterprise of Cognitive Science," and that it is "what the theory of evolution is to all biology, the cell doctrine to cellular biology, the notion of germs to the scientific concept of disease, the notion of tectonic plates to structural geology," namely, it is (he hypothesizes) "adequate to all symbolic activity this physical universe of ours can exhibit, and in particular to all symbolic activities of the human mind." The basic definition of "symbolization" is as follows [21]:

An entity X designates an entity Y relative to a process P , if, when P takes X as input, its behavior depends on Y .

In this case, X is a *symbol* for Y . I do not wish to dwell on how this definition is too permissive for many applications to science, nor on how it radically excludes most of the symbolism that is important in the arts, humanities and religion, nor on the arrogance of attempting to reduce symbolism in general to causality, but rather, I wish to relate this theory to the themes of control and error which are central to the present paper. Newell says,

²Note that in this 1935 passage, "the nation" refers to Nazi Germany!

A general intelligent system must somehow embody aspects of what is to be attained prior to the attainment of it, i.e., it must have *goals*. ...

A general intelligent system must somehow consider candidate states of affairs (and partial states) for the solutions of these goals (leading to the familiar search trees).

But in order to use the familiar method of search trees, one must not only have a goal that is fixed in advance, but one must also be able to enumerate the possible solutions. Thus, we are dealing here with a form of top-down control that is even less flexible than feedback control, and less able to deal with errors. Thus, despite Newell's desire that his ideal physical symbol system should "behave robustly in the face of error" and "learn from its environment," it is far from clear that it could do so with anything like human intelligence; in particular, it is unclear how it could devise entirely new conceptual organizations in response to its errors, let alone learn such things as compassion.

I do not believe that rigidly mechanistic models, with top-down goal structures, are adequate for explaining human cognition, nor even for explaining how to do science. Although this approach is characteristic of "modern" explanations of science, from the seventeenth century into the twentieth – the so-called "Received View" – there is an emerging "post modern" view of science and technology which advocates more flexible organizations, less rigid logics, and more natural control structures. Examples include the so-called New Biology of Bateson, Maturana, Varela and others (discussed in Section 5 below), hermeneutics and other movements in linguistics and philology (again, see Section 5), and fuzzy logic and fuzzy control (e.g., [9, 23]). Within computing, neural nets, highly distributed and open systems, and hypermedia and hyperprogramming may also fit this emerging paradigm.

3 Error-Free Programming

What we may call the "Dijkstra School" aims for error-free programming. For example, [7] claims that

we have ... "a calculus" for a formal discipline – a set of rules – such that, if applied successfully: (1) it will have derived a correct program; and (2) it will tell us that we have reached such a goal.

From a narrow point of view, [7] achieves its aim, modulo certain technical difficulties³. But its fundamental difficulty is that it attempts to control the programming process by imposing a rigid top-down derivation sequence, working backwards from the initial top level specification (the "postcondition") to the final code, in which each step is derived by applying a "weakest precondition" (hereafter, "wp") formula.

Perhaps not unexpectedly, this "wp calculus" requires significant human "invention" at exactly the most difficult points, namely the loops. And for most programs that go much beyond the trivial, the insights needed to write the loop invariants are tantamount

³These include the following: (1) there is a gap in the logical foundations, in that the first order logic used for expressing conditions is not actually sufficiently expressive – something like the infinitary logic proposed by Erwin Engeler in the 1960s is needed; (2) many important programming features are not treated, including procedures, blocks, modules, and objects – in general, all large grain features are omitted; and (3) data structures, types, variables that range over programs, and variables that range over specifications are all treated in a loose manner.

to already knowing how to write the program; moreover, these insights are more difficult to achieve in the wp context than they would be in a more operational context. Indeed, I have seen good students who had been taught that the wp calculus was the right way to program, become so discouraged over the difficulties that they experienced, that they came to believe that they could never learn how to program and should therefore seek some other profession! In general, such a rigid, top-down ideology inhibits experimentation, the exploration of tradeoffs, accidental discoveries, and so on. Moreover, it can be harmful to students, wasteful of time, reinforcing of an inflexible view of life, and inhibiting to intuition and creativity.

But we must not get carried away with criticism: It is not that the wp calculus is entirely mistaken or useless, but rather that claims have been made for it that do not take adequate account of its limitations. For example, the wp calculus can be very useful in getting initializations right (many real bugs arise at this point), as well as for simple loops, and I have also found it useful in convincing students that coding can be treated with mathematical precision. Moreover, Dijkstra's *style* is very elegant and careful, his examples are very well chosen, and personally I admire and have learned from these qualities. However, it seems very difficult to scale up Dijkstra's approach beyond programs of more than a few dozens of lines.

Let me be clear that I am not criticizing formal methods as such – in fact, I believe that they can be very useful in practice, *especially* for large programs⁴, and have myself done research in this area [10, 14] – rather, I am criticizing the tendency to apply formal methods in a rigid, top-down hierarchical manner. In fact, I believe that if appropriate formal methods are used in a flexible, non-ideological way, they can lead to better programs, with greater efficiency and fewer bugs.

But bugs are inevitable. If they don't occur in coding, they will appear in design, specification, requirements, or use; they may arise by misinterpretation of what the customer says, by inadequate modelling of the situation in which the program must run, by inadequate documentation or understanding of the tools being used (such as a compiler for a high level language), and in many other ways. (An overview of some recent debates on the philosophical foundations of formal methods is given by Barwise [2].)

Of course, no one wants bugs, or wants to spend any more time than necessary on debugging, because it is difficult and unpleasant. But nevertheless, bugs are interesting and important in themselves: they define the boundary between what is understood and what is not. Hence, they show us where our weaknesses are, and they provide opportunities for us to learn and grow.

4 Software Quality

The Brooks Report [8] notes that the procurement process generally used by the U.S. Department of Defense for large software systems is inappropriate for such systems (although they might be reasonable for buying boots, hats, or even rifles): bids are invited on a con-

⁴This can be achieved by providing formal specification for the *interfaces* between program components, thus greatly enhancing the accuracy of communication between different groups working on different components, and providing a "fire wall" to protect each group from purely internal changes made by other groups. Also, sufficiently powerful mechanisms for parameterization and modularization can greatly improve the reuse of both code and specifications.

tract to build a system that meets a given "requirements document," which tends to be excessively elaborate, specific, and optimistic. There is also a tendency for lower bids to win, whether or not they are realistic; and once the contract is let, large cost over-runs are common.

It is important to note that we are *not* talking here just about the processes used internally by a software vendor, but rather about the procurement process *as a whole*, including those processes internal to the *client* as well as those internal to vendor(s), and of course those processes of communication that occur on the interfaces among them. It is convenient to use the terminology of *process models* in this discussion, even though it was originally developed to describe just vendor processes (see Boehm [6] for an overview of this field). To be more precise now, it is the government processes of requirements generation and procurement that are rigidly top-down, based on assumptions formalized in the linear structure of a so-called *stagewise model*, which says that a software development project begins with requirements, which then "fall" without essential error into specifications and finally into code. Once the processes internal to a vendor are reached, it is not unusual to see a more sophisticated process models in use, at least a so-called *waterfall model*, which allows feedback between contiguous stages, and perhaps also a single (non-rapid) prototype, or even a *spiral model* [6], which can be sufficiently adaptive to be considered a meta-process model. (Also, note that software procurement is generally less rigid in the commercial sector than in the government sector.) All this suggests that an important topic for further research might be the development of *multi-party* process models, which would allow for different processes within different parties, and for multi-stage interaction between parties.

For large, complex systems, especially if they are unlike anything previously constructed, we can hardly expect to know what is possible or impossible, what is adequate or inadequate, what is expensive or inexpensive, or more generally, what are the design tradeoffs for that class of system. Moreover, it has been found far more expensive to correct errors during the maintenance stage than during earlier stages (by up to a factor of 100) [5].

Thus, it would seem very desirable to debug requirements until they reflect a reasonable compromise between what users want and what is achievable within reasonable cost. The Brooks Report [8] suggests that integrating rapid prototyping with the procurement process might achieve this goal, and thus save vast amounts of time and money. It could also lead to discovering useful capabilities not anticipated in the original requirements document, which are nonetheless relatively easy to provide. It seems very reasonable to suppose that some such more adaptive approach could yield better results than trying to control the entire process of production in advance of exploring the basic pitfalls and tradeoffs that are involved.

The failure of U.S. Government procurement processes to acknowledge the possibility of error in setting requirements is a shocking example of arrogant theological thinking run wild; even some crude form of feedback control would be an improvement, and it is amazing that large Department of Defense systems come close to working correctly as often as they seem to.

I think it is fair to say that Software Engineering is presently more like a medieval craft than it is like a modern engineering discipline. This is because modern technology (see [17]) involves the construction of causal calculative theories, and we are only now beginning to develop such theories for Software Engineering. In particular, the relatively neglected, and sometimes maligned, field of formal methods is still at an early stage of development. A promising approach, I believe, is to integrate formal methods with software process models

in a way that better supports flexibility and adaptation, rather than mere competition and control.

It may be that such revolutionary techniques as hyperprogramming [12], which involve the multimedia exploration of program structure by visualization and explanation, based on technology developed for formal specification and verification, can be developed to the point where they can be used in a routine way.

What is crucial is to provide environments for software development in which the overall vision of the program can be clearly felt at all times, and used flexibly in organizing the programming task. Such a vision is not at all the same thing as a top-down hierarchically structured system of goals, but rather should have an adaptive living quality, in roughly the sense discussed in the next section.

5 The Being of Software Development Projects

Anyone familiar with multi-person software development projects knows that there is a sense in which such projects "have a life of their own": some projects seem healthy and vibrant from the start, and overcome even unexpected obstacles with enthusiasm and intelligence, while others always seem to be disorganized and depressed, suffering, for example, from such symptoms as unrealistic goals, inadequate equipment, poor planning, (seemingly) insufficient funding, faulty communication, indecisive leadership, frequent reorganizations, and/or deep rifts between internal factions.

A software development project is not primarily a formal mathematical entity. Perhaps it is best seen as a *dialogical* or *linguistic* process, an evolving organization of certain informational structures, continually recreating itself by building, modifying, and reusing these structures. In the language of Maturana, this might be described as "development through mutually recursive interactions among structurally plastic systems" [18].

In this view, computers, printouts, compilers, editors, design tools, and even programmers, can be seen as supporting substrates, just as body parts are supporting substrates for a person⁵. Maturana and Varela [19] define an *autopoietic system* to be

... a network of processes of production of components that produces the components that: (i) through their interactions and transformations continuously regenerate the network of processes that produced them; and (ii) constitute it as a concrete unity in the space in which they exist by specifying the topological domain of its realization as such a network.

(See [3, 25, 18, 20] for more information, and see [15, 26] for some possibly ill-advised attempts at formalization.) For example, an unhealthy project may struggle for survival by reassigning responsibilities, redefining subprojects, and even trying to reconstruct the conditions that define its success. On the other hand, a healthy project may develop new tools to enhance its own productivity.

Autopoietic systems are about as far as we know how to get from rigid top-down hierarchical goal-driven control systems; autopoietic systems thrive on error, and reconstruct themselves on the basis of what they learn from their mistakes. Since organizations naturally

⁵Of course, I do not intend these remarks to imply that the group has moral or spiritual priority over the individual, or that people should be viewed as components of systems in anything like the same way that Ada packages can be.

strive for their own survival, it would seem natural to study autopoietic software process models.

It is interesting to notice that the discourse which is the life blood of a software project is conducted in a variety of languages, which differ in both their level of abstraction and in their degree of formality. Most discussions are conducted in a kind of pigeon natural language, infused with technical terms and technical ways of thinking. But there are also requirements documents, designs (which may involve graphics), specifications, code, and much more.

I believe that a promising research direction is to apply techniques from hermeneutics to the "softer" areas of the software development process, and particularly to the so-called "requirements acquisition" phase, in which an analyst attempts to determine what the customer really wants. Hermeneutics is concerned with the interpretation of "texts" in a very broad sense which can include programs, dialogues, contracts, live interaction, specifications, history files, proofs, and so on. Another promising application of hermeneutics might be to study the social dynamics of the entire life cycle, or of selected parts of it. See [22] for an overview of some theoretical aspects of hermeneutics, and [11] for some further discussion along the lines of this paper.

6 Conclusions

Important avenues for further progress in Software Engineering seem to be blocked by our inadequate understanding of the processes involved in developing software systems. It seems that formal methods, despite their power, are not applicable to some of the most significant aspects of such processes. But it also seems possible that a better understanding may be attained by using some insights from the New Biology of Bateson, Maturana, Varela and others, and from the hermeneutics of Heidegger, Gadamer, and others. A basic step in this direction is to recognize the important role that error plays in any process of construction. The surprisingly widespread belief that it is both desirable and possible to go from requirements to specification, to code, without making any errors, would seem to be a major inhibiting factor to the successful application of formal methods.

Although formal methods can be very powerful when they are properly applied, they also have definite limitations, and formal, rationalistic understanding is only one of many approaches to understanding. Intuition and spiritual understanding are alternatives that seem more important in certain ways. For example, formal methods will never tell us why the U.S. Department of Defense persists in its manifestly wasteful practices. Nor will they explain the success of object oriented programming.

Some specific proposals for further research mentioned earlier in this paper include: the application of hermeneutic techniques to the software development process, both as a method of study, and also as a specific technique for use in the requirements acquisition phase; the development of multi-party process models; the study of autopoietic process models; and the integration of formal methods with such more "organic" process models, through techniques like hyperprogramming.

By some such route, we might go further than merely recognizing the inevitability of error — we might learn to experience our errors as a path that leads to deeper understandings and to better relationships. We must make the programming process not merely tolerant of error, but also able to *take advantage* of the creative possibilities inherent in the interplay between concept and perception. Until we acknowledge the dialectical, creative, and living

dimensions in programming, we shall be doomed to participate in software processes that are unwieldy, unpleasant, and ineffective. The denial of error is the denial of life.

Acknowledgements

I wish to thank my wife Kathleen for assistance with preparing this paper, including reading several drafts, undertaking some library research, and providing many helpful comments and conversations. I would also like to thank both the Naropa Institute in Boulder, Colorado, and the Center for the Study of Language and Information at Stanford University for providing stimulating environments in which to think about the kind of issue discussed here.

References

- [1] Francis Bacon. *The Wisdom of the Ancients*. Da Capo Press (Amsterdam), 1968. Facsimile of 1619 translation by Arthur George (printed by John Bill, London).
- [2] Jon Barwise. Mathematical proofs of computer system correctness. Technical Report CSLI-89-136, Center for the Study of Language and Information, Stanford University, August 1989.
- [3] Gregory Bateson. *Mind and Nature*. Bantam, 1980.
- [4] Ernest Becker. *The Denial of Death*. Free Press, 1973.
- [5] Barry Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.
- [6] Barry Boehm. A spiral model of program development and enhancement. *Software Engineering Notes*, 11(4):14-24, 1986.
- [7] Edsger Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18:453-457, 1975.
- [8] Frederick Brooks *et al.* Report of the Defense Science Board Task Force on Military Software. Technical Report AD-A188 561, Office of the Under Secretary of Defense for Acquisition, Department of Defense. Washington DC 10301, September 1987.
- [9] Joseph Goguen. The logic of inexact concepts. *Synthese*, 19:325-373, 1968-1969.
- [10] Joseph Goguen. Reusing and interconnecting software components. *Computer*, 19(2):16-28, February 1986. Reprinted in *Tutorial: Software Reusability*, Peter Freeman, editor, IEEE Computer Society Press, 1987, pages 251-263.
- [11] Joseph Goguen. Hermeneutics and path. In Reinhard Budde, Christiane Floyd, Reinhard Keil-Slawik, and Heinz Züllighoven, editors, *Software Development and Reality Construction*. Springer, 1990.
- [12] Joseph Goguen. Hyperprogramming: A formal approach to software environments. In *Proceedings, Symposium on Formal Approaches to Software Environment Technology*. Joint System Development Corporation, Tokyo, Japan, January 1990.

- [13] Joseph Goguen. Truth and meaning beyond formalism. In Reinhard Budde, Christiane Floyd, Reinhard Keil-Slawik, and Heinz Züllighoven, editors, *Software Development and Reality Construction*. Springer, 1990.
- [14] Joseph Goguen and José Meseguer. Unifying functional, object-oriented and relational programming, with logical semantics. In Bruce Shriver and Peter Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 417–477. MIT Press, 1987. Preliminary version in *SIGPLAN Notices*, Volume 21, Number 10, pages 153–162, October 1986.
- [15] Joseph Goguen and Francisco Varela. Systems and distinctions; duality and complementarity. *International Journal of General Systems*, 5:31–43, 1979.
- [16] Martin Heidegger. *An Introduction to Metaphysics*. Yale University Press, 1959. Translation by Ralph Manheim; original from 1935.
- [17] Martin Heidegger. The question concerning technology. In *Basic Writings*, pages 283–217. Harper and Row, 1977. Translated by David Krell; original from 1953.
- [18] Humberto Maturana. Biology of language: The epistemology of reality. In *Psychology and Biology of Thought and Language: Essays in Honor of Eric Lenneberg*, pages 27–64. Academic Press, 1978.
- [19] Humberto Maturana and Francisco Varela. *Autopoiesis and Cognition: The Realization of the Living*. Reidel, 1980.
- [20] Humberto Maturana and Francisco Varela. *The Tree of Knowledge*. Shambhala, 1987.
- [21] Allen Newell. Physical symbol systems. *Cognitive Science*, 4:135–183, 1980.
- [22] Richard Palmer. *Hermeneutics*. Northwestern University Press, 1969.
- [23] Witold Pedrycz. *Fuzzy Control and Fuzzy Systems*. John Wiley, 1989.
- [24] Alfred Tarski. The semantic conception of truth. *Philos. Phenomenological Research*, 4:13–47, 1944.
- [25] William Irwin Thompson, editor. *Gaia: a Way of Knowing*. Lindisfarne Press, 1987.
- [26] Francisco Varela and Joseph Goguen. The arithmetic of closure. *Journal of Cybernetics*, 8:125ff, 1978. Also in *Progress in Cybernetics and Systems Research*, Volume 3, edited by R. Trappl, George Klir and L. Ricciardi, Hemisphere Publishing Co., 1978.

Formal Methods: A Position Paper

Joseph A. Goguen

1 Introduction

This paper has four main parts. The first asks what formal methods are and the second asks what they are good for, while the third describes a specific technique in formal methods called *hyperprogramming*, and the fourth presents some recommendations.

2 What are Formal Methods?

“Formal” means “having to do with form” and does not necessarily entail logic or proofs of correctness. Of course, the word “formal” can also be used in many other senses, but I think that this may be the appropriate sense for “formal” in the phrase “formal methods.” For example, a formal development method gives rules that restrict the allowed forms of program development, and perhaps also the allowed forms of some texts that occur during the process. But this does not mean that form is trivial — far from it. Indeed, everything we do is done with form. And everything that computers do is formal, in that definite syntactic structures are manipulated according to definite rules. Usually, we don't do things just to follow the form — we have some purpose in mind, and the formal structures that we use, whether PERT charts, programs, parse trees, or differential equations, have a meaning for us.

For many people the prime example of a formal system is first order logic. This system encodes first order model theory with certain formal rules of deduction that are provably sound and complete. However, our experience with theorem provers shows that it can be difficult to work with this system. Moreover, formal systems that try to capture even higher levels of meaning, e.g., languages for expressing requirements, tend to be even harder to work with, and to have even less pleasant properties. (Later I will argue for a natural hierarchy of levels of meaning, from abstract mathematical objects up towards concrete social values.)

In summary, formal methods are syntactic in essence, but semantic in purpose. In computing science, form does not embody content, but rather encodes it.

2.1 Formal Methods and Mathematics

There has been much confusion about the relationship between computing science and mathematics, and particularly about the relationship between computing science and logic. Unlike numbers, computers have a real physical existence, and so do programs. On the other hand, algorithms and models of computation (such as Turing machines, or term rewriting systems) are abstract mathematical entities. What seems problematical is the relationship between the physical entities and the mathematical abstractions.

In my view, this relationship is the entirely familiar one that the ancient Greeks discovered between bodies in the real world and abstractions in axiomatic geometry. Thus, we can prove theorems about (abstract) points, lines, planes and pyramids, but not about the

Great Pyramid of Cheops, whose edges and faces are not very regular. Although we can apply suitable theorems to a physical pyramid, we cannot expect the conclusions of a theorem to be more valid than warranted by its assumptions. The situation is much the same with programs and computers. We cannot prove the correctness of a real program running on a real computer. But we can prove the correctness of an algorithm, and we can expect a program on a computer to behave as we wish to the extent that the program's execution conforms to the algorithm.

It is an error to conflate mathematical models with the concrete realities they are supposed to represent. Hence it is as much an error to claim that computing science has all the "good" properties of mathematics, as it is to claim that it has all the "bad" properties of the real world. We may call those who make the first error the "Dijkstra school" (everything is provable) and those who make the second error the "Fetzer school" (nothing is provable). Perhaps the excessive optimism of the first helps to explain the excessive pessimism of the second.

In summary, some parts of computing science are pure mathematics (concerned with ideal algorithms and models of computation) and some parts are applied mathematics (concerned with applying mathematical models to real programs and computers). Later I will argue that some parts are neither of these, but instead have to do with the social context of computing.

3 What Good are Formal Methods?

The most difficult problems do not arise in relating algorithms to programs, but rather in evaluating how well a program (running on a computer, which I will largely ignore hereafter) solves some real world problem, such as preventing the theft of funds or information, detecting enemy missiles, making a profit on the stock market, or ensuring the survival of an organism. The trouble in such examples is that the program must perform in an environment that is enormously complex, rapidly changing, and imperfectly understood. Moreover, the requirement for the program may also be complex, changing, and imperfectly understood; in some cases, it is so hound up with social and/or political issues that even trying to state it with greater precision can engender so much debate about larger issues that general agreement on its meaning is impossible.

The ideal of having accurate mathematical models of the real environment and the real requirement is not achievable for many large, complex, real world problems. This means it is inevitable that methods less than purely formal will play an important role in evaluating real programs. In fact, many informal methods are already important in computing science, starting at a relatively low level with communication media such as graphics, natural language documentation, animation, and audio. Furthermore, computers are becoming increasingly interconnected with each other and with other parts of the real world, through networks, modems, mice, Fax machines, digital audio chips, radar antennae, video cameras, etc., and this trend seems likely to continue. And finally, how can we be sure that we have formalized the right thing? Or formalized it the right way? Clearly, we need to get outside the formal system in order to make such judgements.

We do not have, and perhaps we never will have, fully adequate theories of the meaning of the information that is encoded in such complex forms as natural language. And the prospects are even less encouraging for fully adequate theories of the significance of such information in larger contexts. But we do have formal rules that describe the structure of

the data that encodes many kinds of meaning, and we also have many programs that can manipulate such data for particular purposes. For example, we can divide a message into words and sentences, and count the number of each; we can search for keywords, and do various statistical analyses; and we can do spelling correction to a useful extent (but not perfectly without human help).

There are various *levels* of structure that we might seek to formalize. A computer analysis of a message can be fairly certain about words and sentences, somewhat less sure about spelling, and quite unsure about meaning. We have formal methods that are applicable at each level of structure, but in general, the higher the level, the more important *informal* methods of analysis become. The obvious hierarchical structure has to do with whole/part relationships: words are parts of sentences, and sentences are parts of messages. This is a formal hierarchy, and also a hierarchy of forms. But there is an *informal* hierarchy that is perhaps even more important, whose levels correspond to meanings in wider and wider contexts. For example, one relatively high level of meaning might concern the artistic merit of a text in a certain culture.

I claim that the same is true of program correctness. We can formally verify syntax with considerable certainty. We can formally verify semantic assertions about state with reasonable certainty, although the effort needed seems to grow exponentially with program size. We can hope to better understand some of the interactions of a program with components of some larger systems of which it is a part. But current formal methods do not seem especially useful, for example, in determining the effectiveness of a program in achieving a business plan, or in benefiting society as a whole.

In summary, there is a tendency for formal methods that encode higher levels of meaning to require greater computational resources, and perhaps even to pass into regions that we do not know how to formalize adequately. However, there often exist some rather "syntactic" methods which achieve a useful compromise between expressive power and computational difficulty.

3.1 The Myth of Control

Managers want to control the programming process; they want to be sure that the product will meet its requirements, and will be finished in time and within cost. This is entirely reasonable, but we all know that in practice, managers often do not achieve this kind of control: programs often don't do exactly what they are supposed to do, and often take much longer and cost much more than estimated. The desire for control motivates the so-called "waterfall" models of the programming process, in which higher-level descriptions are supposed to determine lower level descriptions, in a strict hierarchy from user requirements down to code.

One trouble with such models is that they make no allowance for *error*. Clients do not usually know exactly what they want; for they do not usually know what is possible or impossible, or how much the various possibilities will cost; also, they cannot foresee all the ramifications of the various other systems with which the system that they are buying may interact. (Of course, neither can the programmers.) Furthermore, there is no room for the exploration of entirely new capabilities that may be revealed during the design or construction processes. This is frustrating to the programmers, as well as to the managers and customers.

It is my view that the total programming process should be as flexible as possible, so

that workers at each level can participate in a dialogue with the levels above and below. As advocated by the Brooks report, rapid prototyping can help to achieve this goal in some cases. However, it is difficult to construct prototypes directly from requirements, and the usual kinds of prototypes do not help much with many higher level evaluations, such as whether the system should be built at all, or how it should be used and managed, or how it will interact with other systems already in place.

It is sometimes claimed that by using formal methods we can avoid all errors in programming. Even if we interpret this claim in the narrow sense of guaranteeing the satisfaction of some formal mathematical specification (as opposed to an informal social requirement), it is still not true in practice, because we can make mistakes; for example, we can neglect to follow the method at some apparently trivial point, with unexpectedly serious consequences; or we can make a syntax error during proof, with the result that we prove a property of the wrong function.

Even worse, attempting to enforce the rigid use of a formal method can be very damaging, by preventing flexibility and inhibiting creativity. Rigid top-down design methodologies do not work for large programs, and are unpleasant and stifling to use even on small programs. In co-teaching a course at Oxford, I found that some students who believed my co-teacher's assertions that they should be able to get their programs right the first time by using weakest preconditions backwards from a post-condition, lost the confidence that they could ever learn to program at all. This is a great pity.

Under some conditions, a formal proof of correctness can be worse than useless, by encouraging misplaced confidence that the program will meet its intuitive requirement in its actual operating environment. For example, if a company promotes a formally verified heart pace-maker as infallible, physicians might neglect to provide adequate safeguards.

It is well-known that many of the most important scientific and technological discoveries were accidental (e.g., penicillin), or arose through trial and error (e.g., the light bulb), and I think that we should allow for similar processes of exploration in programming. Perhaps formal methods can help with this by providing techniques to ensure the inter-consistency of the many different texts that arise in producing large and complex systems. One approach to this is discussed in the next section.

4 Hyperprogramming

Large programs have many parts whose interactions and interconnections are under constant evolution during their development. This means that the many texts associated with the program, including its requirements, specifications, code, documentation, accounting information, test suites, and version and configuration files, will be changing constantly. It is highly desirable to provide support for maintaining the mutual inter-consistency of such texts as they change. It is not practical to do this for the contents of these texts, but it seems promising to apply formal methods at various levels of their forms. For example, consider the manual for an operating system that is to run on several machines, and is frequently corrected, augmented and ported. Then we can build programs to ensure that the organization of the manual remains consistent, and that if part of the program is changed, then the corresponding manual pages are re-examined to see if they also must be changed.

But perhaps we can go further. If we associate documentation to the parts of a program, then we can assemble the manual from its parts in the same way that we assemble the

program from its components. Furthermore, if the documentation and the program are parameterized in the same way, then we might be able to evaluate a single interconnection statement that would accomplish these two different purposes. In fact, we have already developed a theory of *module expressions* which can serve such a purpose. Their use in programming is called *parameterized programming*, and it can be considered a substantial generalization of the programming-in-the-large style embodied in the UNIX `make` statement. One dimension of the generalization is to provide powerful facilities for both generic modules and module inheritance; the former also allows us to specify semantic properties of module interfaces.

Parameterized programming has been implemented and tested for the functional programming language OBJ, and has also been suggested for Ada and other languages. *Hyperprogramming* is the extension of this approach to texts other than programs. For example, it could be used to combine graphical illustrations with written texts, to assemble a spoken explanation from parts and then "execute" it with a speech chip, to produce program animations from specifications of program parts, and perhaps even to combine such animations with speech to form "movies" that illustrate program operation.

Although these considerations motivated the name "hyperprogramming," its most important application might be the coherent integration of the many different components of a large software development project. Current practice does not support the integration of rapid prototyping with the evolution of specifications and code, nor does it support consistency checks between such texts as specifications, test suites and code. Moreover, accounting and management information are usually handled quite separately from code, and documentation is only developed after coding is completed. Hyperprogramming could integrate all these diverse aspects in a uniform way that guarantees certain important kinds of consistency.

5 Recommendations

The suggestions in this section are based on my own experience. Of course, this means that they are biased. But I think this should be considered a strength rather than a weakness, as long as I clearly indicate the source. Except for the first list, these recommendations can also serve as a summary of the preceding discussions. I begin with some observations that have to do with general funding policies:

- It is difficult to get funding for innovative ideas that require the development of prototypes, because the funding programs with sufficient money tend to have goals that are excessively narrow and short-term.
- Funding is too unstable and subject to excessive delays. As a result, it is difficult to hire and keep good people.
- Educational expectations are too low in computing science, and in particular, they are lower than in other engineering disciplines. Funding should be devoted to raising both the mathematical preparation (especially in logic and algebra) and the practical experience of computing science graduates. Formal methods should be taught in the universities.

- Open dissemination of basic research results and of experimental systems is essential in order to obtain the best use of research funding by maximizing the discussion of critical issues.

Next, I list some fairly specific research topics on the border between formal and informal methods. As previously argued, this area seems very important for software methodology. Although short term practical results seem unlikely, I believe that important basic results can be obtained by people who are proficient in both formal and informal methods. The informal (or perhaps one should say "semi-formal") methods that seem most relevant come from the social sciences, and include discourse analysis, socio-linguistics, and ethnomethodology.

- It seems likely that the dialogical processes between clients and designers that result in requirements could be formalized to a certain extent, and that this could, at the very least, result in more realistic expectations about what can be accomplished in this stage of the development of complex systems.
- A linguistic study of the relationship between requirements texts and the texts produced at lower levels, such as designs and specifications, might yield formal structures that would facilitate these important transitions in the program development process. In particular, it would be interesting to know where misunderstandings most often occur in the present process.
- It would be interesting to study the integration of various kinds of text in various media, to see what constraints must be satisfied to ensure that the intended relationships are actually perceived by users.
- More generally, it would be interesting to study the integrated use of multiple levels of formalization in programming environments. For example, it might be useful to do a critical path analysis of the arguments that support a given requirement, so that the formality of items on the critical path can be increased if desired.

Finally, I list some (relatively) specific research topics that lie entirely within the area of formal methods and that I think could yield very substantial advances within the medium-term time frame:

- The integration of specification, prototyping and theorem proving. This could be done by using an executable specification language that is rigorously based upon logic; indeed, every execution of a program in such a language is a proof of something, and if the language is rich enough, it could be a proof of something interesting. We have done some experiments in hardware verification using the OBJ3 system which suggest that this is a promising research direction.
- The development of semantically rationalized object-oriented programming languages. For example, an integrated functional and object-oriented language could be useful for many purposes, including structured verification and integration with the so-called structured design methodologies, like those of Jackson and Yourdon.
- The development of languages that support object-oriented design; this is related to the development of rationalized semantics for object-oriented programming, but would most usefully be realized in connection with existing languages like Ada. It would also be interesting to develop systems that support object-oriented verification.

- The development of prototype hyperprogramming systems. This might include integrating the graphical conventions used in structured development methods. It would also be interesting to study the automatic generation of graphical representations of abstract data types from their algebraic specifications; it seems likely that this could be extended to the automatic generation of animations.

Of course, the pursuit of these research goals will not solve the philosophical problems discussed in Sections 2 and 3 of this paper. However, the philosophical considerations do lead to support the belief that such research goals may be both important and feasible.

I hope that no one will misunderstand my interest in informal methods, my criticisms of current methodologies, or my general philosophical positions as being critical of formal methods. On the contrary, I believe that we may be on the threshold of a golden age of formal methods, provided that we neither interpret the field too narrowly, nor expect it to achieve the impossible. There seem to be many areas where formal methods can be developed and/or synthesized with informal methods to solve important problems in the development of large systems.

Truth and Meaning beyond Formalism

Joseph A. Goguen

1 Introduction

Logic and formal semantics have been enormously helpful in understanding programs and programming languages, and in automating some aspects of the programming process. Therefore computer scientists have good professional reasons to be interested in truth and meaning construed in a narrow technical sense, through symbolic logic and formal semantics.

But computer scientists also need to better understand the processes that create and sustain software, and in particular, the complex relationships between computer systems, individuals, and societies. Moreover, we also need to develop more humility about our own role in the scheme of things. Unfortunately, these problems raise deep questions about truth and meaning which cannot be addressed by formal semantics.

This short paper is intended to suggest why computer scientists might be interested in the work of Heidegger, Wittgenstein and others¹, and to stimulate some further thought about some of the questions that they address. Although Heidegger did not write very much that is explicitly about formal logic, what he did write is quite pertinent, and much of his other work is relevant to questions of meaning in the larger sense. Wittgenstein was concerned with the limits of language, that is, with “what cannot be said”. We will see that their views are fundamentally opposed to those of logical positivists such as Carnap, as well as to the whole Anglo-American tradition of analytic philosophy, and in particular, to Russell and Moore. We will also see some interesting parallels to Buddhist philosophy.

The end of the paper will return to consider what all this has to do with computing.

2 Heidegger, Carnap and Wittgenstein on Dread and “the Nothing”

This section tells the story of an encounter (in print) between three of the most influential philosophers of the early twentieth century. In 1931, the logical positivist Carnap [3] took Heidegger's 1929 essay “What is Metaphysics?” [10] as a paradigmatic example of what he called “metaphysical pseudostatements”. Carnap's program was to develop an ideal language based on logic, in which all words would refer to observable sense data, experiences or things, and in which a “logical syntax” would guarantee that all sentences are meaningful by eliminating all “nonsensical” combinations of words that are still permitted by grammatical syntax. Such a language would eliminate metaphysics (and much of the rest of philosophy) as well as all literature and poetry, in much the same way that Orwell's “Newspeak” in *Nineteen Eighty-Four* would eliminate all language that is inconsistent with the ideology of Big Brother. As Orwell [15] says,

¹ Among the attempts to summarize Heidegger's philosophy that I have found the most useful is [16]; see also [20] for other indications of its relevance to computer science, and the excellent paper by Capurro [2]. [11] is an excellent source of background information on Wittgenstein.

Newspeak was designed not to extend but to *diminish* the range of thought, and this purpose was indirectly assisted by cutting the choice of words down to a minimum.

According to Carnap [3],

The meaning of a statement lies in the method of its verification. A statement asserts only so much as is verifiable with respect to it. Therefore a sentence can be used only to assert an empirical proposition, if indeed it is used to assert anything at all. ... Logical analysis, then, pronounces the verdict of meaninglessness on any alleged knowledge that pretends to reach above or behind experience. ... The (pseudo)statements of metaphysics do not serve for the *description of states of affairs*, either existing ones (in that case they would be true statements) or nonexisting ones (in that case they would be at least false statements).

Specifically, Carnap criticizes Heidegger's use of the word "nothing" in the following assemblage² from [10], by showing that it violates his "logical syntax":

What is to be investigated is being only and — *nothing* else; being alone and further — *nothing*; solely being, and beyond being — *nothing*. *What about this Nothing?* ... *Does the Nothing exist only because the Not, i.e., the Negation, exists?* Or is it the other way around? *Do Negation and the Not exist only because the Nothing exists?* ... We assert: *the Nothing is prior to the Not and the Negation.* ... Where do we seek the Nothing? How do we find the Nothing? ... We know the Nothing. ... *Anxiety reveals the Nothing.* ... That for which and because of which we were anxious, was 'really' — nothing. Indeed: the Nothing itself — as such — was present. ... *What about this Nothing?* — *The Nothing itself nothings.*

As an example of Carnap's analysis, Heidegger's sentence "We know the nothing" is represented as "*K(no)*" and then claimed meaningless because "nothing" (denoted "*no*") is used as a noun. But Heidegger says "The nothing is neither an object nor any being at all" [10]. Hence, Carnap is accusing Heidegger of something which Heidegger clearly says cannot be done, namely taking "the nothing" as "a something".

Indeed, Heidegger anticipates precisely the sort of attack which Carnap mounts when he says [10]:

But perhaps our confused talk already degenerates into an empty squabble over words. Against it science must now reassert its seriousness and soberness of mind, insisting that it is concerned solely with beings. The nothing — what else can it be for science but an outrage and a phantasm? If science is right, then only one thing is sure: science wishes to know nothing of the nothing.

This quotation indicates that Heidegger knows he is playing with words. But in order to explore the foundations of something, it is necessary to step outside its bounds. As Wittgenstein says [23], in direct contradiction to Carnap.

²The italics and deletions are Carnap's, and it is worth noting that by taking these widely scattered sentences out of their contexts, and by his selected italicization and capitalization, Carnap creates a very distorted view of Heidegger's text.

I can readily understand what Heidegger means by Being and Dread. Man has the impulse to run up against the limits of language³. Think, for example, of the astonishment that anything exists⁴. This astonishment cannot be expressed in the form of a question, and there is also no answer to it⁵. Everything which we feel like saying can, a priori, only be nonsense ... Yet the tendency represented by the running-up against *points to something*. St. Augustine already knew this when he said: What, you wretch, so you want to avoid talking nonsense? Talk some nonsense; it makes no difference!

(The history of this little passage is interesting. A "sanitized" version appeared in the *Philosophical Review* in 1965 without the first sentence and without the original title, which was *On Heidegger*.)

Unfortunately, Wittgenstein himself did not take St. Augustine's advice to "talk nonsense," and as a result, the *Tractatus* is in some ways very obscure. In fact, Wittgenstein was rather systematically misunderstood, and hence distorted, by the logically oriented philosophers of the Vienna Circle and the Anglo-American tradition (this is clearly explained in [11]). Russell's preface to the *Tractatus* [21] is a good example, since it criticizes Wittgenstein on precisely those points where his contribution was perhaps most original and significant, namely the pages from Proposition 6.4 onward, which discuss such topics as ethics and the "problem of life".

It is notable that Carnap and Wittgenstein share not only a common interest in logic, science and language, but also in what cannot be said. However, their attitudes towards this area were entirely different: Carnap considered that everything outside his ideal logical language was nonsense without meaning, whereas Wittgenstein considered that everything of the greatest value and interest was contained in this realm. In contrast, Heidegger was not only willing to talk "nonsense", he was willing to break the bonds of language by making up new words, and by using old words in new, often ungrammatical or "illogical" ways to indicate deeper meanings which strictly speaking cannot be said at all, but only pointed at.

Carnap [3] recognized that art operates outside of the strictly verifiable, but he still considered that the metaphysician

confuses [science and art] and produces a structure which achieves nothing for knowledge and something inadequate for the expression of an attitude. ... The metaphysician believes that he travels in territory in which truth and falsehood are at stake. In reality, however, he has not asserted anything, but only expressed something, like an artist. ... [The statements of metaphysicians] serve for the expression of the general attitude of a person toward life. ... Metaphysicians are musicians without musical ability.

Thus, Carnap does not intend to ban everything that falls outside his language, but only to label it meaningless. However, both his view of meaning, and his view of art as expressing a general attitude toward life, are very limited. In contrast, Heidegger says "Art lets truth originate. Art ... is the spring that leaps to the truth of beings ..." [9].

³Murray [13] explains that this is a reference to Kierkegaard, who was the first philosopher to give a serious treatment of anxiety ("Angst" in German, translated "dread" in the previous sentence) before Heidegger.

⁴This is a reference to the last sentence of [10], which is "Why are there beings at all, and why not rather nothing?" This fundamental theme of Heidegger is, for example, the central question of [6].

⁵Although this is a criticism of Heidegger's formulation, the last sentence of this quotation from Wittgenstein suggests that they may not differ so much after all.

3 What is "the Nothing"?

It is no coincidence that Heidegger, Carnap and Wittgenstein collide on the issues of "the nothing" and anxiety. All three philosophers can be seen as advancing Kant's program to stem the turgid tide of traditional metaphysics that still today flows on at us from out of the Middle Ages. One major goal of Kant's "critical philosophy" was to show the limits of reason *from within*, that is, using the tools of reason, in order to prevent its misuse. In particular, Kant wished to separate the realm of reason from that of value. For example, Kant wrote a treatise which denied that there could be any justification for blaming the great Lisbon earthquake of 1775 either on the presence of a few Protestants there (as did many Catholics in Lisbon) or on the adherence of the majority to Catholicism (as did some English clerics).

Kant took subject centered rationalism to its limit, declaring that we construct objects according to a *a priori* given faculties of mind, which include space, time, causality, objectness, number, affirmation, negation and possibility. This analysis assumes a world of "things in themselves" and an idealized human subject, both of which are unknowable. Thus, Kant's so-called "second Copernican revolution" actually went in the *opposite* direction from that of Copernicus, since it placed man in the *center* again, as the constructor of perceived objects (the phenomenal). Since the time of Descartes, this kind of move has been seen as necessary to secure a firm grounding for the objectivity of science.

Wittgenstein's *Tractatus* can be seen as a Kantian critical (i.e., from within) deconstruction of logical positivism, despite its significant contributions to the technique of logic (e.g., truth tables). In particular, Wittgenstein argues that it is impossible to express the meaning of a formal language within the language itself, and instead uses a "picture" theory of meaning in which interpretations can be shown but cannot be said. Russell and Carnap tried to counter this by proposing a "meta"-language in which the meaning of an "object" language can be expressed, and even an infinite tower of meta-, meta-meta-, meta-meta-meta-, ... languages. But as Russell admits in his preface to the *Tractatus*, Wittgenstein's argument seems to apply just as well to such a tower of languages as it does to a single language.

If it is impossible to express the meaning of a sentence *within* its language, then it is necessary to move *outside* the language. This led the later Wittgenstein to investigate the conventions which determine when and how sentences can be used. He called these flexible rule-governed symbol using activities "language games" [22]. They, in turn, get their meanings from the even more flexible and larger grained patterns of symbol using activities of which they are parts, which he called "forms of life". This point of view is quite different from that in the *Tractatus*, which had simply assumed that the relation between language and reality is one of "picturing" (i.e., representation). But in late Wittgenstein, it is the rules of language games which determine the limits of what can be said.

This whole development, starting from Kant's clarification of the subject centered approach of traditional metaphysics and science, and proceeding to the work of Heidegger, Derrida, Barthes and other modern French thinkers, can be seen as an on-going *deconstruction of the self*, which is nothing other than the exploration of "the nothing" as it applies to the knowing subject of Descartes and Kant. The inevitable conclusion is that there is no rational basis for assuming such a subject. Instead, subject and object continually emerge and dissolve together.

Many books have been written in the Buddhist tradition about "the nothing," called

shunyata in Sanskrit. One of the most famous is the *Mulamadhyamakakarika*s of Nagarjuna, from around the second century A.D.; a more recent one is *Religion and Nothingness* [14] by Nishitani, considered the dean of the Kyoto School of philosophy. Buddhism says that the experience of *shunyata* is egolessness, the lack of any subject, and says that egolessness is a fundamental fact of human existence. One way that this experience can manifest is described by the Tibetan meditation master Trungpa Rinpoche [19] as follows:

It is a very desolate situation. It is like living among snow-capped peaks with clouds wrapped around them and the sun and the moon starkly shining over them. Below, tall alpine trees are swayed by strong, howling winds and beneath them is a thundering waterfall. From our point of view, we may appreciate this desolation if we are an occasional tourist who photographs it or a mountain climber trying to climb to the mountain top. But we do not really want to live in those desolate places. It's no fun. It is terrifying, terrible.

No wonder, as Heidegger says, "science wishes to know nothing of the nothing." For science wishes to banish dread and proceed with its objectivity firmly established in the subjectivity of its scientists, reducing "the nothing" to mere negation, which is a rational operation on beings, as opposed to the terrifying emptiness from which beings emerge into authenticity.

However, if Heidegger and the Buddhists are right, it is the possibility of non-being which gives beings their character of luminosity⁶, and hence the nothing, i.e., *shunyata*, is not only prior to negation, but also to beings.

The effect of this, as Heidegger says, is to rob logic of its claim to supremacy, and in particular, to rob it of its claim to provide foundations for science and even for mathematics. Indeed, we must conclude that foundations in the sense sought by logicians are simply not possible. The judgements that we make, and in particular negative judgements, are necessarily grounded in our being-in-the-world, and not in any pre-existing unshakable truths, or eternal world of ideal things.

More significantly, we may conclude that it is the finitude, limitation, or mortality of beings which makes them luminous. The fundamental importance of finitude for Being is expressed in the thundering series of questions which close Heidegger's major work, *Being and Time* [7]. The finitude and luminosity of beings are two of the many suggestive points of contact between Heidegger and Buddhism. For "impermanence" (i.e., finitude) is one of the Three Marks of Existence (the other two are egolessness and suffering).

4 What are Truth and Meaning?

The intimate relationship between truth, meaning and being in the Western philosophical tradition goes back to the ancient Greeks, and is extensively discussed, for example, by Aristotle; these three correspond (roughly) to the Greek words *aletheia*, *logos* and *on*.

Most attempts to explicate these notions and their relationship have taken as paradigmatic the "eternal" sentences of mathematics (such as " $2 + 2 = 4$ "), whose "meaning" is a truth value that is independent of any context in which the sentence might be uttered. But

⁶It is impossible to "define" the experience of luminosity. But perhaps it might be some help to say that it refers to the flickering of beings between presence and non-presence. On the other hand, this may be an example of something which really cannot be said.

such sentences are exceedingly rare in "earthly" discourse, where meanings can be far more complex than just "true" or "false," and where context has a profound effect upon meaning.

In his essay "On the Essence of Truth" [8], Heidegger criticizes semantic theories that are based on the so-called "Correspondence Theory of Truth":

"Truth" is not a feature of correct propositions which are asserted of an "object" by a human "subject" and then "are valid" somewhere, in what sphere we know not; rather, truth is disclosure of beings through which an openness essentially unfolds.

That is, according to the Correspondence Theory, a statement is "true" just in case what it asserts is a fact about the world. Although the assertion is made by a (human) subject about some object, the true statements themselves are ideal forms in a Platonic realm that is only dimly perceived by humans. Instead of this, Heidegger says that truth is a process of unfolding, of disclosure. That is, [6],

The essence of being is *physis* [i.e., appearing]. ... Appearing makes manifest. Already we know then that being, appearing, causes to emerge from concealment. Since the essent⁷ as such is, it places itself in and stands in *unconcealment, aletheia*. ... The Greek essence of truth is possible only in one with the Greek essence of being as *physis*. On the strength of the unique and essential relationship between *physis* and *aletheia*, the Greeks would have said: The essent is true insofar as it is. The true as such is essent. This means: The power that manifests itself stands in unconcealment. In showing itself, the unconcealed as such comes to stand. Truth as un-concealment is not an appendage to being.

This is a radically different notion of truth from that which we find in logic or empirical science. It has nothing to do with operations of measurement or of verification, carried out by some human subject. Rather, it has to do with authentic presence, with the power of beings to emerge from the nothing. This approach to truth and being does not presuppose a knowing subject, and does not reduce beings to objects of knowledge; for Heidegger and the ancient Greeks, being and truth are pre-conceptual.

It seems clear that from this perspective, "meaning" is not an "object," whether as part of some formal causal theory, as an abstract logical intention, or as some set-theoretic entity. The following may provide some reference points in a search to understand our alternative sense of "meaning":

1. *Meaning is ontological.* All experience is inextricably bound up with Being and with beings, i.e., with luminous appearance. In particular, meaning arises through openness to being, or as Heidegger says, "The essence of truth is freedom" [8].
2. *Meaning is dialectical.* Meaning is only disclosed through engagement with beings, through uncertainty and questioning, through making mistakes, exploring oppositions, and seeking roots (for some further discussion of the development of meaning, see [5]; for more on error, see [4]).

⁷The word "essent" was made up by the translator Ralph Manheim to translate Heidegger's made-up word "Seiende," which (roughly speaking) means "something that exists," an "existent." In this paper, I have mostly used the word "being" for this.

3. *Meaning is historical.* Because meaning is dialectical, it only emerges through time, through the accumulation of questionings, encounters and revealings, in the context of a tradition or lineage (see [5] for some related discussion).

Interestingly enough, recent efforts to extend formal semantics beyond mathematics and science, for example to natural languages, can be seen as embodying (diluted versions of) similar principles. In particular, recent work in philosophy and linguistics has proposed new formalisms for complex meanings that can vary with context, and can model discourse and other interactions. Examples include the work of Montague, using intensional logic [12], work of Barwise, Perry and others on "situation semantics" [1], of Strachey and Scott [17, 18] on "denotational" semantics, and many other formalisms developed for the semantics of programming languages. But all these theories posit abstract entities, such as "intentions," "situations" or "denotations" that are quite remote from the human experience of meaningfulness, and it is not clear that they can tell us anything important about what it means to be human. In particular, they do not deal with truth as the unconcealment of beings.

On the other hand, it seems clear that these advances are technically useful. For example, they may help us to write programs that are more accurate, more general, more efficient, and more reusable; they may also help us to write programs that can help us in programming. They may even some day lead to machines that can understand and speak the sort of utilitarian languages of which Carnap would approve.

5 Where are we?

The *Tractatus* [21] concludes with the following mysterious proposition,

What we cannot speak about we must pass over in silence.

which is perhaps intended as a summary of Wittgenstein's arguments that the meaning of a language cannot be expressed in the language itself.

In a sense, this whole paper has been about that which cannot be said. We first presented arguments against Carnap's narrowly dogmatic "logical syntax" and his rejection of Heidegger as nonsense. While accepting that a line of the kind that Carnap wants to draw can in fact be drawn, we agreed with Wittgenstein that all the most important things lie on what Carnap would regard as the wrong side of it. On the other hand, I cannot agree with Wittgenstein that we must remain silent about these things. Even though they may not make strict logical sense, they are too important not to bring into the open through dialogue.

As an illustration, we tried to explore Heidegger's "nothing" and why it might be prior to negation, with some help from the later Wittgenstein and Buddhist philosophy. This also perhaps gave some insight into the foundations of logic. We next tried to follow Heidegger's approach to truth, beginning with his rejection of the Correspondence Principle, and then moving on to *physis* and *aletheia*, which reveal a completely different perspective from that of formal semantics. We concluded with some pointers toward the meaning of meaning, followed by a short summary of some recent work in formal semantics.

But how does all this relate to computing?

I think we must conclude that the techniques of computer science, such as formal semantics, logic, and even simulation, cannot tell us the meanings of computer systems, in the

broad human sense of "meaning". This becomes an issue especially for so-called "embedded computer systems". For example, consider the question of what the Star Wars weapons system really means: is it a defense system, as its proponents tend to claim, or is it really an offensive system, intended to provide some protection after a first strike has been launched? Such a question cannot be answered without a careful consideration of social and political factors, as well as a careful assessment of technical capabilities. To remain silent on such issues is to invite manipulation, or even tyranny.

To address such questions, it is not necessary to be "an expert," that is, to have everything already worked out. Indeed, it is not even desirable, because genuine meaning only arises through uncertainty and questioning, even through confusion and error. It is necessary to enter into a dialogue in order for truth to emerge from concealment.

Similar considerations hold for many less dramatic and more ordinary situations. For example, suppose that we are part of a team that is producing a large business system, and one day the customer tells us of an unexpected change in the tax laws, which it turns out will require keeping much more data than had previously been anticipated; unfortunately, this means that the system will have to run on different hardware, because the old requirements led to choosing hardware that cannot handle so much data. The customer has trouble understanding why his system will now cost more, and threatens to sue. The head of the company threatens to counter-sue. Some team members panic and consider quitting.

Is there any way that formal semantics can save the day? No, there is not. We will have to negotiate. Of course, formal semantics might play some role, for example, in revising the specifications, but the real meaning of this situation is a human one, involving a conflict of interests between the company and its customer.

Numerous other examples could be given. There are many aesthetic decisions to be made in programming. These are not without meaning. If a program is elegantly designed and coded, then it may be easier to debug, maintain, and reuse.

Also, the members of a programming team have to work together, and the project will only prosper if there is a spirit of friendly cooperation, rather than, say, envy, bitterness, or competition. Formal semantics might be used to specify a component, but anger could cause someone to write it a particularly obscure way.

In all such situations, it is vital to understand the difference between issues that can be resolved by appeal to formal semantics (e.g., "is this code right?") and issues which cannot (e.g., "is this code elegant?"), and it is vital to approach each kind of meaning in an appropriate way. I would like to think that the philosophy of Wittgenstein, Heidegger, and the Buddhists might be some help in this regard, and I have tried to explain how this might be so. But really, common sense is likely to be more valuable than philosophy here, unless perhaps some antidote is needed against previous large doses of positivistic or analytic philosophy. Moreover, even this would require thinking about things that are difficult or even impossible to say clearly. So I do not imagine that I have done more than provide a few pointers for those who may want to pursue such issues further, and I hope that the reader will take this paper in that light, and will enjoy looking into some of the original source material, and thinking things through on his/her own.

Acknowledgements

I would like to thank my wife Kathleen and my son Healdene for reading through several drafts of this paper and providing many helpful comments and conversations. I would also

like to thank both the Naropa Institute in Boulder, Colorado, and the Center for the Study of Language and Information at Stanford University for providing stimulating environments in which to think about the kind of issue discussed here.

References

- [1] Jon Barwise and John Perry. *Situations and Attitudes*. MIT Press, 1983.
- [2] Rafael Capurro. Informatics and hermeneutics: Some criticisms of the Winograd/Flores view of computer-based information systems. In *Software Development and Reality Construction*. Springer, 1990.
- [3] Rudolph Carnap. The overcoming of metaphysics through logical analysis of language. In Michael Murray, editor, *Heidegger and Modern Philosophy*. Yale University Press, 1978. Original in *Erkenntnis* 2, 1931; translation by Arthur Pap.
- [4] Joseph Goguen. The denial of error. In Reinhard Budde, Christiane Floyd, Reinhard Keil-Slawik, and Heinz Züllighoven, editors, *Software Development and Reality Construction*. Springer, 1990.
- [5] Joseph Goguen. Hermeneutics and path. In Reinhard Budde, Christiane Floyd, Reinhard Keil-Slawik, and Heinz Züllighoven, editors, *Software Development and Reality Construction*. Springer, 1990.
- [6] Martin Heidegger. *An Introduction to Metaphysics*. Yale University Press, 1959. Translation by Ralph Manheim; original from 1935.
- [7] Martin Heidegger. *Being and Time*. Blackwell, 1962. Translated by John Macquarrie and Edward Robinson from *Sein und Zeit*, Niemeyer, 1927.
- [8] Martin Heidegger. On the essence of truth. In *Basic Writings*, pages 113–141. Harper and Row, 1977. Translated by David Krell; original from 1930.
- [9] Martin Heidegger. The origin of the work of art. In *Basic Writings*, pages 149–187. Harper and Row, 1977. Translated by David Krell; original from 1936.
- [10] Martin Heidegger. What is metaphysics? In *Basic Writings*, pages 91–116. Harper and Row, 1977. Translated by David Krell; original from 1929.
- [11] Allan Janik and Stephen Toulmin. *Wittgenstein's Vienna*. Simon and Schuster, 1973.
- [12] Richard Montague. *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, 1974. Edited and with an introduction by Richard Thomason.
- [13] Michael Murray, editor. *Heidegger and Modern Philosophy*. Yale University Press, 1978.
- [14] Keiji Nishitani. *Religion and Nothingness*. University of California Press, 1982.
- [15] George Orwell. *Nineteen Eighty-Four*. Penguin, 1989. First Edition published by Martin Secker and Warburg, 1949.
- [16] Richard Palmer. *Hermeneutics*. Northwestern University Press, 1969.

- [17] Dana Scott and Christopher Strachey. Towards a mathematical semantics for computer languages. In *Proceedings, 21st Symposium on Computers and Automata*, pages 19–46. Polytechnic Institute of Brooklyn, 1971. Also Technical Monograph PRG 6, Oxford University, Programming Research Group.
- [18] Joseph Stoy. *Denotational Semantics of Programming Languages: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.
- [19] Chögyam Trungpa. *The Myth of Freedom*. Shambhala Press, 1976.
- [20] Terry Winograd and Fernando Flores. *Understanding Computers and Cognition*. Addison-Wesley, 1987.
- [21] Ludwig Wittgenstein. *Tractatus Logico-Philosophicus*. Routledge and Kegan Paul, 1922. English translation by D.F. Pears and B.F. McGuinness, with an Introduction by Bertrand Russell; original German edition in *Annalen der Naturphilosophie*, 1921.
- [22] Ludwig Wittgenstein. *Philosophical Investigations*. Macmillan, 1968. English translation of the Third Edition by G.E.M. Anscombe.
- [23] Ludwig Wittgenstein. On Heidegger on Being and Dread. In Michael Murray, editor, *Heidegger and Modern Philosophy*. Yale University Press, 1978. Translation and commentary by Michael Murray. The complete German text first appeared as “Zu Heidegger” in *Ludwig Wittgenstein und der Wiener Kreis: Gespräche, aufgezeichnet von Friedrich Waismann*, 1967.

Hermeneutics and Path

Joseph A. Goguen

1 Introduction

Hermeneutics is the study of interpretation, particularly the interpretation of linguistic texts, but also of human experience in general, since this can be seen as both “textual” and “linguistic” in appropriately broad senses of these words.

The works of Heidegger [5], Gadamer [3] and others¹ say many interesting things about the nature of interpretation and its philosophical implications, but they contain very little for the person who wants to learn how to do interpretation better, or for the person who wants to know how to teach others how to do it; the *practical* dimension is missing from this tradition. There is a striking difference between *philosophy*, which is content to make distinctions and debate issues, and a *path* which provides practices and guidelines for practice, constituting a way forward which is nevertheless based on acknowledging where we are.

Interpretation is a demanding discipline, encompassing essentially everything we are and everything that is. Its practice has the potential to open us up to what we are and what our world is. What is missing is a set of *guidelines* that tell us how to deal with the problems that inevitably arise, and other practices that are less involved with conceptual content and have the possibility of sharpening our general mindfulness and awareness.

This short paper suggests that perhaps Buddhism, and in particular Mahayana Buddhism, can supply this missing practical element. (The word “*mahayana*” means “great path” in Sanskrit, and describes the tradition from which Zen and Tibetan Buddhism have sprung, among others.) The result is that the activity of hermeneutics, that is, of interpreting, can also be a path, by interpreting the term “hermeneutics” sufficiently broadly.

2 The Paramitas

To be a good interpreter, I believe that it helps to be a “good person” in roughly the same sense expressed in Buddhism by the “six paramitas.” The word *paramita* means “other shore” in Sanskrit, and refers to action which is not selfish, and which thus transcends this shore of the river of confusion and neurosis. Some hint of this may perhaps be glimpsed in the following brief characterizations, which have been specialized to the interpretation of a text which you should think of as coming from your own time and place:

1. *Dana*, which means generosity in Sanskrit, is the joy of discovering that you don’t have to impose your own conceptions on the text, that you can afford to be open to it, that you can give up your conceptual (and preconceptual) territory.
2. *Sila*, which means discipline or morality, is that you don’t have to make any special effort, you already have (what is called in ethnomethodology – see [12]) “member’s

¹Palmer [10] gives a relatively accessible summary of this tradition; see also [1].

- competence": you are sufficiently grounded in your own tradition and in that of the text to begin work on it, and you are inspired to do so. There is no need for dogma, and you can work with what is actually there.
3. *Ksanti*, or *patience*, is that you don't have to "succeed," i.e. to satisfy your own, or anybody else's, expectations about the interpretation; you can therefore go at the speed which is proper to the task, and not worry about whether what you discover will be "acceptable".
 4. *Virya*, or *energy*, is to work with what is given, with what you are and what the text is (including the whole context of the text and of yourself); you completely accept the tradition of the text, and then you work from there, without, however, being bound by 'conventional wisdom.' You can actually take delight and inspiration in whatever contradictions and difficulties may arise.
 5. *Dhyana*, *chan* [in Chinese], *zen* [in Japanese], or meditative awareness, is to be completely absorbed in the text, without distinguishing between yourself and it, but being fully aware of the environment of the text and of yourself. Your horizon merges with that of the text; or perhaps there is no horizon, that is, no center and no fringe.
 6. *Prajna*, or transcendental knowledge, is the precision of discriminating awareness, which is willing and able to recognize and to cut through your preconceptions, as well as those in the text; you can learn from mistakes without worrying about ability or inability, superiority or inferiority. This is "stable awareness" rather than confused awareness.

These characterizations were obtained by combining my interpretation of Trungpa Rinpoche's treatment of the six paramitas in [11] with my interpretations of Heidegger, Gadamer and others. These aspects of interpretation (or of meditation) do not necessarily arise in strict sequential order, but there is still a sense in which they build on one another, so that *prajna* is the fruition of the others.

3 Confusion

The word "confusion" is used here in a somewhat technical sense, referring to mind that is not characterized by the paramitas. This is our ordinary confused mind, which sometimes mislays pens and papers, and is often misled by its own hopes and fears.

It is important to note that non-confused mind arises by transcending confused mind; clarity does not come to us from some separate pure realm of its own. It arises from accepting what actually happens to us, and working with it as it is, rather than as we wish it were. As Heidegger [5] says,

when something ready-to-hand is found missing, though its everyday presence has been so obvious that we have never taken any notice of it, this makes a *break* in those referential contexts in which circumscription discovers. Our circumscription comes up against emptiness, and now sees for the first time *what* the missing article was ready-to-hand *with*, and *what* it was ready-to-hand *for*. The environment announces itself afresh ... [and] is thus lit up.

In textual interpretation, this has a very practical meaning: the feelings of confusion, attraction, or aversion which we experience while reading a text, while not necessarily reliable in themselves, are the *energy* that we have for working with the text; they are the breaks in the seemingly seamless seas of meanings that can help us get deeper into the world of the text.

4 Hermeneutics as Path

Without an intimate awareness of how one's own mind works, especially how one's emotional and conceptual haggage get in the way of seeing things as they are, it is difficult to transcend one's confusion and actually use it in textual interpretation. Such an intimate awareness of the confused functioning of mind is difficult to obtain, and according to most Buddhist traditions, the practice of meditation is the most effective way forward.

Indeed, most Buddhist teachers insist that it is necessary to practice meditation in order for paramita practice to be meaningful, because it is necessary to develop the qualities of "mindfulness" and "awareness" first. This kind of meditation practice does not aim to produce either a hypnotic trance state, or to control the restlessness of mind; it is not concentration. Rather, mindfulness-awareness meditation takes as its subject something very simple and natural, such as breath. Mindfulness is attention to what is actually there, "one pointed," direct and precise. Awareness is the context, the space, within which mindfulness happens. This is not at all a matter of calculating or of grasping for meaning. As Trungpa Rinpoche [11] says,

Mindfulness provides some ground, some room for recognition of aggression, passion and so on. Mindfulness provides the topic or the terms or the words, and awareness is the grammar which goes around and correctly locates the terms. Having experienced the precision of mindfulness, we might ask the question of ourselves, "What should I do with that? What can I do next?" And awareness reassures us that we do not really have to do anything with it but can leave it in its own natural place. It is like discovering a beautiful flower in the jungle; shall we pick the flower and bring it home or shall we let the flower stay in the jungle? Awareness says leave the flower in the jungle, since it is the natural place for that plant to grow. So awareness is the willingness not to cling to the discoveries of mindfulness, and mindfulness is just precision; things are what they are.

Or in the words of Heidegger [6], "we should *do* nothing, but rather wait."

In this way, one comes to see the nature of the mind; that is, meditation is the interpretation of mind. Thus, the path of hermeneutics is the path of meditation. Of course, Buddhism is concerned with one's whole life, not just with how one interprets texts; but because one's life can be viewed as a text, these concerns are quite closely related.

5 Hermeneutics in the Practice of Science

There are many ways that the paramitas might be relevant to the practice of science. Perhaps the most obvious is also the most personal: the scientist might practice meditation, and hence change the way he relates to everything, including science. But let us consider something

simpler and more direct, reading a scientific text, for example [2], which is the basis for a course at Oxford which I have been teaching.

As it happens, Dijkstra has not been one of my favorite authors; so the first paramita, *dana* or generosity, has a particularly pointed meaning here: I should drop my prejudices and open up to the text; insofar as I succeed in this, my experience will both be more pleasant and more accurate; just that realization brings a sense of relief. Of course, I must also be aware of what I already know and do not know as I work with the text, and this is *śila* or discipline. *Kṣanti* or patience means not only that I should be willing to work through any technical difficulties that may arise as I read, but also that I don't have to compete with the author.

Virya or energy arises as I actually do all this; if I've got the first three paramitas right, there will be no particular pain or frustration to this process, but rather it will be natural and self-energizing. This leads to *dhyana*, awareness, in which I can be authentically engaged with the text and its context, including other related texts and my own being. This does not mean that I must accept everything the author says; on the contrary, I am now in a position to appreciate it properly and fully, both its strengths and weaknesses, as well as my own; this is *prajna* or discriminating awareness.

And what did I learn? The paper [2] is very concise, clearly and compellingly written, has excellent examples, and has stood the test of time (this can be seen by comparison with other *CACM* papers from that year, and also from the large literature that [2] has inspired). However, I was irritated that the author paid no attention to logical foundations or to model theoretic semantics, and gave no indications of the limitations of his methods; also, I kept wondering how to formulate things more algebraically. Eventually, I discovered that the issue of foundations is rather subtle (something like infinitary logic, as explored by Engeler in the 60's, is needed), that the model theoretic semantics is awkward, that the approach works poorly for large programs (since no account is taken of modules or of data structures), and that category theoretic formulations have already been given (for example, by Manes and Arbib [9]). Also, I discovered that worry about all of this got in the way of my appreciating the elegance of the language design, the beauty of the examples, the motivation in terms of programming style, and the richness of the research that this paper opened up. So in the end, I learned something about myself as well.

In discussing interpretation, we are not talking about discovering some objective truth about a text. Rather, there is a very intimate relationship between the interpreter and the interpreted, in which each is uncovered to the extent that the enterprise succeeds. As Heidegger [5] says, "interpretation is never a presuppositionless grasping of something given in advance." Indeed, it is typical that we can learn the most about ourselves from those texts, or parts of texts, where we have the strongest reactions.

6 Emptiness and Beyond

Buddhism might perhaps be described as a participatory phenomenological hermeneutics of mind, leading to the experience (not just the idea) of non-duality between self and other, and between mind and body. In contrast, traditional science is a hermeneutics of other, which already presupposes subject-object duality.

The traditions of meditation and hermeneutics that we have been discussing are not consistent with this classical version of science. In particular, Heidegger presents a stinging

critique of the Western metaphysical tradition, including science and technology [8]. Heidegger's hermeneutics opposes the idea that there are *objects* already given in the world, which are *observed* by *subjects*; it opposes the ideas of control and manipulation, whether for material or intellectual gain; and it opposes our usual idea of *idea*, a pre-existing intellectual structure which we see only dimly, as on the walls of Plato's cave. Similar views can be found in Buddhism.

Both Buddhism and science are complex evolving systems, with no ultimate commitment to any particular dogma, belief or theory; instead, each is united by its commitment to particular methods and by immersion in its particular historical tradition. Both are characterized by debate, and by the growth of insight. And contemporary science may even be developing some appreciation for the inseparability of subject and object. (See [4] for more discussion along similar lines.)

In Buddhism, "emptiness," or *shunyata* in Sanskrit, refers to this non-duality of self and other, that is, of subject and object. These are our two most basic and generic concepts, and without them, all other concepts are also empty. *Shunyata* is thus an opposite to Plato's doctrine of ideas. But *shunyata* is not a doctrine of nihilism. Indeed, without concepts, the world can shine forth more brightly. For if we ask, "Are self and other the same? Or are they different?" we find that they are neither the same nor different. For in any experience of self and other as being the same or different, self and other necessarily arise together. As Hayward [4] says,

as the mutual dependence [of objects] with the perceiver is felt, they shine with a spacious but self-luminous quality that is at the same time empty of inherent existence. This luminosity that is beyond concept is the fullness of *shunyata*.

Similarly, Heidegger says in "The Origin of the Work of Art" [7], of a Greek temple set in a valley, that it

causes [its material] to come forth for the very first time and to come into the Open of the work's world. The stone comes to bear and rest and so first becomes stone; metals come to glitter and shimmer, colors to glow, tones to sing, the word to speak.

7 Colophon

First of all, I thank Vidyadhara, the Venerable Chögyam Trungpa, Rinpoche for whatever little I know about Buddhism. I also thank Prof. Rod Burstall of the University of Edinburgh, Dr. José Meseguer of SRI International, and my wife Kathleen, for many helpful comments and conversations, and both Naropa Institute and the Center for the Study of Language and Information at Stanford University for partial support and for stimulating environments. Special thanks to Dr. Charlotte Linde, from which I learned much of what I know about discourse analysis and sociolinguistics.

The basic text of this piece was found buried in my computer file system; it was written about 1976 for a course on Discourse Analysis that I taught at Naropa Institute with Dr. Charlotte Linde. It was then lightly edited at SRI in 1988 for distribution at the *Conference on Software Development and Reality Construction*, and was put into its present form at Oxford in early 1990.

References

- [1] Rafael Capurro. Informatics and hermeneutics: Some criticisms of the Winograd/Flores view of computer-based information systems. In *Software Development and Reality Construction*. Springer, 1990.
- [2] Edsger Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18:453-457, 1975.
- [3] Hans-Georg Gadamer. *Philosophical Hermeneutics*. University of California Press, 1976. Translated and edited by David Linge.
- [4] Jeremy Hayward. *Shifting Worlds, Changing Minds*. Shambhala, 1987.
- [5] Martin Heidegger. *Being and Time*. Blackwell, 1962. Translated by John Macquarrie and Edward Robinson from *Sein und Zeit*, Niemeyer, 1927.
- [6] Martin Heidegger. *Discourse on Thinking*. Harper and Row, 1966. Translated by John Anderson and Hans Freud from *Gelassenheit*, Neske, 1959.
- [7] Martin Heidegger. *Poetry, Language, Thought*. Harper and Row, 1971. Translations by Albert Hofstadter.
- [8] Martin Heidegger. *The Question Concerning Technology and other Essays*. Harper and Row, 1977. Translations by William Lovitt.
- [9] Ernest Manes and Michael Arbib. *Algebraic Approaches to Program Semantics*. Springer, 1986.
- [10] Richard Palmer. *Hermeneutics*. Northwestern University Press, 1969.
- [11] Chōgyam Trungpa. *The Myth of Freedom*. Shambhala Press, 1976.
- [12] Roy Turner, editor. *Ethnomethodology*. Penguin, 1974.