SOFTWARE AND MIND

Andrei Sorin

EXTRACT

Introduction: *Belief and Software*

This extract includes the book's front matter and the introductory chapter.

This chapter is an introduction to the mechanistic myth and the mechanistic software myth, and an analysis of the similarity of mechanistic software beliefs to primitive beliefs.

The entire book, each chapter separately, and also selected sections, can be viewed and downloaded at the book's website.

www.softwareandmind.com

SOFTWARE AND MIND

The Mechanistic Myth and Its Consequences

Andrei Sorin

ANDSOR BOOKS

Published by Andsor Books, Toronto, Canada (January 2013)
www.andsorbooks.com

For disclaimers see pp. vii, xv–xvi.

Designed and typeset by the author with text management software developed by the author and with Adobe FrameMaker 6.0. Printed and bound in the United States of America.

Acknowledgements

Excerpts from the works of Karl Popper: reprinted by permission of the University of Klagenfurt/Karl Popper Library.

Excerpts from *The Origins of Totalitarian Democracy* by J. L. Talmon: published by Secker & Warburg, reprinted by permission of The Random House Group Ltd.

Excerpts from *Nineteen Eighty-Four* by George Orwell: Copyright ©1949 George Orwell, reprinted by permission of Bill Hamilton as the Literary Executor of the Estate of the Late Sonia Brownell Orwell and Secker & Warburg Ltd.; Copyright ©1949 Harcourt, Inc. and renewed 1977 by Sonia Brownell Orwell, reprinted by permission of Houghton Mifflin Harcourt Publishing Company.

Excerpts from *The Collected Essays, Journalism and Letters of George Orwell*: Copyright ©1968 Sonia Brownell Orwell, reprinted by permission of Bill Hamilton as the Literary Executor of the Estate of the Late Sonia Brownell Orwell and Secker & Warburg Ltd.; Copyright ©1968 Sonia Brownell Orwell and renewed 1996 by Mark Hamilton, reprinted by permission of Houghton Mifflin Harcourt Publishing Company.

Excerpts from *Doublespeak* by William Lutz: Copyright ©1989 William Lutz, reprinted by permission of the author in care of the Jean V. Naggar Literary Agency.

Excerpts from *Four Essays on Liberty* by Isaiah Berlin: Copyright ©1969 Isaiah Berlin, reprinted by permission of Curtis Brown Group Ltd., London, on behalf of the Estate of Isaiah Berlin.

Library and Archives Canada Cataloguing in Publication

Sorin, Andrei
Software and mind : the mechanistic myth and its consequences / Andrei Sorin.

Includes index.
ISBN 978-0-9869389-0-0

1. Computers and civilization. 2. Computer software – Social aspects.
3. Computer software – Philosophy. I. Title.

QA76.9.C66S67 2013 303.48'34 C2012-906666-4

Printed on acid-free paper.

Don't you see that the whole aim of Newspeak is to narrow the range of thought?… Has it ever occurred to you … that by the year 2050, at the very latest, not a single human being will be alive who could understand such a conversation as we are having now?

George Orwell, *Nineteen Eighty-Four*

Disclaimer

This book attacks the mechanistic myth, not persons. Myths, however, manifest themselves through the acts of persons, so it is impossible to discuss the mechanistic myth without also referring to the persons affected by it. Thus, all references to individuals, groups of individuals, corporations, institutions, or other organizations are intended solely as examples of mechanistic beliefs, ideas, claims, or practices. To repeat, they do not constitute an attack on those individuals or organizations, but on the mechanistic myth.

Except where supported with citations, the discussions in this book reflect the author's personal views, and the author does not claim or suggest that anyone else holds these views.

The arguments advanced in this book are founded, ultimately, on the principles of demarcation between science and pseudoscience developed by philosopher Karl Popper (as explained in "Popper's Principles of Demarcation" in chapter 3). In particular, the author maintains that theories which attempt to explain non-mechanistic phenomena mechanistically are pseudoscientific. Consequently, terms like "ignorance," "incompetence," "dishonesty," "fraud," "corruption," "charlatanism," and "irresponsibility," in reference to individuals, groups of individuals, corporations, institutions, or other organizations, are used in a precise, technical sense; namely, to indicate beliefs, ideas, claims, or practices that are mechanistic though applied to non-mechanistic phenomena, and hence pseudoscientific according to Popper's principles of demarcation. In other words, these derogatory terms are used solely in order to contrast our world to a hypothetical, ideal world, where the mechanistic myth and the pseudoscientific notions it engenders would not exist. The meaning of these terms, therefore, must not be confused with their informal meaning in general discourse, nor with their formal meaning in various moral, professional, or legal definitions. Moreover, the use of these terms expresses strictly the personal opinion of the author – an opinion based, as already stated, on the principles of demarcation.

This book aims to expose the corruptive effect of the mechanistic myth. This myth, especially as manifested through our software-related pursuits, is the greatest danger we are facing today. Thus, no criticism can be too strong. However, since we are all affected by it, a criticism of the myth may cast a negative light on many individuals and organizations who are practising it unwittingly. To them, the author wishes to apologize in advance.

Contents

Preface

The book's subtitle, *The Mechanistic Myth and Its Consequences*, captures its essence. This phrase is deliberately ambiguous: if read in conjunction with the title, it can be interpreted in two ways. In one interpretation, the mechanistic myth is the universal mechanistic belief of the last three centuries, and the consequences are today's software fallacies. In the second interpretation, the mechanistic myth is specifically today's mechanistic *software* myth, and the consequences are the fallacies *it* engenders. Thus, the first interpretation says that the past delusions have caused the current software delusions; and the second one says that the current software delusions are causing further delusions. Taken together, the two interpretations say that the mechanistic myth, with its current manifestation in the software myth, is fostering a process of continuous intellectual degradation – despite the great advances it made possible. This process started three centuries ago, is increasingly corrupting us, and may well destroy us in the future. The book discusses all stages of this degradation.

The book's epigraph, about Newspeak, will become clear when we discuss the similarity of language and software (see, for example, pp. 411–413).

Throughout the book, the software-related arguments are also supported with ideas from other disciplines – from philosophy, in particular. These discussions are important, because they show that our software-related problems

are similar, ultimately, to problems that have been studied for a long time in other domains. And the fact that the software theorists are ignoring this accumulated knowledge demonstrates their incompetence. Often, the connection between the traditional issues and the software issues is immediately apparent; but sometimes its full extent can be appreciated only in the following sections or chapters. If tempted to skip these discussions, remember that our software delusions can be recognized only when investigating the software practices from this broader perspective.

Chapter 7, on software engineering, is not just for programmers. Many parts (the first three sections, and some of the subsections in each theory) discuss the software fallacies in general, and should be read by everyone. But even the more detailed discussions require no previous programming knowledge. The whole chapter, in fact, is not so much about programming as about the delusions that pervade our programming practices. So this chapter can be seen as a special introduction to software and programming; namely, comparing their true nature with the pseudoscientific notions promoted by the software elite. This study can help both programmers and laymen to understand why the incompetence that characterizes this profession is an inevitable consequence of the mechanistic software ideology.

There is some repetitiveness in the book, deliberately introduced in order to make the individual chapters, and even the individual sections, reasonably independent. Thus, while the book is intended to be read from the beginning, you can select almost any portion and still follow the discussion. An additional benefit of the repetitions is that they help to explain the more complex issues, by presenting the same ideas from different perspectives or in different contexts.

The book is divided into chapters, the chapters into sections, and some sections into subsections. These parts have titles, so I will refer to them here as *titled* parts. Since not all sections have subsections, the lowest-level titled part in a given place may be either a section or a subsection. This part is, usually, further divided into *numbered* parts. The table of contents shows the titled parts. The running heads show the current titled parts: on the right page the lowest-level part, on the left page the higher-level one (or the same as the right page if there is no higher level). Since there are more than two hundred numbered parts, it was impractical to include them in the table of contents. Also, contriving a short title for each one would have been more misleading than informative. Instead, the first sentence or two in a numbered part serve also as a hint of its subject, and hence as title.

Figures are numbered within chapters, but footnotes are numbered within the lowest-level titled parts. The reference in a footnote is shown in full only the first time it is mentioned within such a part. If mentioned more than once,

in the subsequent footnotes it is usually abbreviated. For these abbreviations, then, the full reference can be found by searching the previous footnotes no further back than the beginning of the current titled part.

The statement "italics added" in a footnote indicates that the emphasis is only in the quotation. Nothing is stated in the footnote when the italics are present in the original text.

In an Internet reference, only the site's main page is shown, even when the quoted text is from a secondary page. When undated, the quotations reflect the content of these pages in 2010 or later.

When referring to certain individuals (software theorists, for instance), the term "expert" is often used mockingly. This term, though, is also used in its normal sense, to denote the possession of true expertise. The context makes it clear which sense is meant.

The term "elite" is used to describe a body of companies, organizations, and individuals (for example, the software elite); and the plural, "elites," is used when referring to several entities, or groups of entities, within such a body. Thus, although both forms refer to the same entities, the singular is employed when it is important to stress the existence of the whole body, and the plural when it is the existence of the individual entities that must be stressed. The plural is also employed, occasionally, in its normal sense – a group of several different bodies. Again, the meaning is clear from the context.

The issues discussed in this book concern all humanity. Thus, terms like "we" and "our society" (used when discussing such topics as programming incompetence, corruption of the elites, and drift toward totalitarianism) do not refer to a particular nation, but to the whole world.

Some discussions in this book may be interpreted as professional advice on programming and software use. While the ideas advanced in these discussions derive from many years of practice and from extensive research, and represent in the author's view the best way to program and use computers, readers must remember that they assume all responsibility if deciding to follow these ideas. In particular, to apply these ideas they may need the kind of knowledge that, in our mechanistic culture, few programmers and software users possess. Therefore, the author and the publisher disclaim any liability for risks or losses, personal, financial, or other, incurred directly or indirectly in connection with, or as a consequence of, applying the ideas discussed in this book.

The pronouns "he," "his," "him," and "himself," when referring to a gender-neutral word, are used in this book in their universal, gender-neutral sense. (Example: "If an individual restricts himself to mechanistic knowledge, his performance cannot advance past the level of a novice.") This usage, then, aims solely to simplify the language. Since their antecedent is gender-neutral ("everyone," "person," "programmer," "scientist," "manager," etc.), the neutral

sense of the pronouns is established grammatically, and there is no need for awkward phrases like "he or she." Such phrases are used in this book only when the neutrality or the universality needs to be emphasized.

It is impossible, in a book discussing many new and perhaps difficult concepts, to anticipate all the problems that readers may face when studying these concepts. So the issues that require further discussion will be addressed online, at *www.softwareandmind.com*. In addition, I plan to publish there material that could not be included in the book, as well as new ideas that may emerge in the future. Finally, in order to complement the arguments about traditional programming found in the book, I plan to publish, in source form, some of the software applications I developed over the years. The website, then, must be seen as an extension to the book: any idea, claim, or explanation that must be clarified or enhanced will be discussed there.

INTRODUCTION

Belief and Software

This book is largely a study of delusions – mechanistic delusions. But, whereas in the following chapters we discuss the *logical* aspects of these delusions, in this introductory chapter we concentrate on their *human* aspects.

Belief, as we all know, is stronger than reason. For a person who believes that the number 13 brings misfortune, a hundred logical arguments demonstrating the fallacy of this idea amount to nothing; at the same time, one story of an accident that occurred on the 13th day of a month suffices to validate the idea. Similarly, we will see, it is quite easy to expose the absurdity of the *mechanistic* beliefs. Yet hundreds of millions of people – people who think of themselves as modern and rational – spend a great part of their life engaged in activities that are, essentially, an enactment of these beliefs. Clearly, it would be futile to attempt to understand the mechanistic myth without taking into account its emotional roots.

It is in order to emphasize their primacy, therefore, that I deal with the human aspects of the mechanistic myth before its logical aspects. But this book is concerned, ultimately, with logical thinking. Thus, a second reason for including the study of human nature in the introduction is that it is only a brief discussion of this important topic.

Modern Myths

The historian of religions Mircea Eliade predicts that "the understanding of myth will one day be counted among the most useful discoveries of the twentieth century."[1] Myths used to be considered – along with fairy tales, legends, and fables – merely folklore: picturesque stories transmitted to us from ancient times, perhaps carrying some moral lessons, but generally of little value in the modern world. It is only recently, starting with the work of anthropologists like Bronislaw Malinowski, that we have come to view myths in a new light. These scholars studied the life of primitive societies extant in various parts of the world by living among those people and learning their languages and customs. As these cultures exemplify all early societies, this information, combined with our historical knowledge, has helped us to form a more accurate picture of the capabilities, values, and beliefs of archaic man. Even more importantly, it has helped us to understand the development and nature of our own, present-day culture.

One thing we have discovered from these studies is the critical function that myth fulfils in a human society. Myth, according to Malinowski, "supplies the charter for ritual, belief, moral conduct and social organization."[2] Far from being simply folklore, myths are the foundation upon which the entire social system rests: "Studied alive, myth … is not symbolic, but a direct expression of its subject matter; it is … a narrative resurrection of a primeval reality, told in satisfaction of deep religious wants, moral cravings, social submissions, assertions, even practical requirements. Myth fulfills in primitive culture an indispensable function: it expresses, enhances, and codifies belief; it safeguards and enforces morality; it vouches for the efficiency of ritual and contains practical rules for the guidance of man. Myth is thus a vital ingredient of human civilization; it is not an idle tale, but a hard-worked active force."[3]

It is wrong to study a myth by inquiring whether it makes sense. Myths are a sacred tradition, and "the main object of sacred tradition is not to serve as a chronicle of past events; it is to lay down the effective precedent of a glorified

[1] Mircea Eliade, *Myths, Dreams, and Mysteries: The Encounter between Contemporary Faiths and Archaic Realities* (New York: Harper and Row, 1975), p. 38.

[2] Bronislaw Malinowski, "Myth as a Dramatic Development of Dogma," in *Malinowski and the Work of Myth*, ed. Ivan Strenski (Princeton, NJ: Princeton University Press, 1992), p. 122.

[3] Bronislaw Malinowski, *Magic, Science and Religion, and Other Essays* (Garden City, NY: Doubleday Anchor, 1954), p. 101.

past for repetitive actions in the present."[4] We must study, therefore, not so much the *text* of a story or legend, as its effects on living society. Myths make absurd claims, but we must ignore their scientific inaccuracy. The "extravagant elements in the myth … can only be understood by reference to ritual, ethical, and social influences of the story on present day conduct."[5]

A myth, then, must be judged solely by its power to inspire large numbers of people. Blatant impossibilities or inconsistencies do not detract from its power. On the contrary, since it is precisely the fantastic elements in a myth that impress us, they are its most important value. Thus, a story that makes only reasonable and verifiable claims cannot possibly serve as myth.

Eliade notes how quickly our perception of the function of myth has changed, from the belief that it is "only fables," to the appreciation that "a man of the traditional societies sees it as the *only valid revelation of reality*."[6] The function of myth is the exact opposite of what we thought it to be: rather than relying on proven knowledge in their important activities and turning to myths in their diversions, it is actually in their important activities that the primitives rely on myths. Because they are inherited from previous generations, myths are believed to represent unquestionable facts: "The myth is thought to express *absolute truth*, because it narrates a *sacred history*.… Being *real* and *sacred*, the myth becomes exemplary, and consequently *repeatable*, for it serves as a model, and by the same token as a justification, for all human actions."[7] Conversely, something that is not reflected in myths is deemed to be untrue and profane.

The greatest benefit that emerges from the study of myth is not a better understanding of primitive cultures, but a better understanding of our own, modern culture. We must "integrate the myth into the general history of thought, by regarding it as the most important form of collective thinking. And, since 'collective thinking' is never completely abolished in any society, whatever its degree of evolution, one did not fail to observe that the modern world still preserves some mythical behaviour."[8] Thus, "*upon the plane of social living*, there was no break in the continuity between the archaic world and the modern world."[9]

But there seem to be few myths left in the modern world. Moreover, those that still exist do not seem to provide anywhere near the powerful inspiration that myths provided in earlier civilizations. So this important question arises: "If the myth is not just an infantile or aberrant creation of 'primitive' humanity, but is the expression of *a mode of being in the world*, what has become of myths

[4] Malinowski, "Dramatic Development," p. 123.
[5] Ibid.
[6] Eliade, *Myths, Dreams, and Mysteries*, p. 24.
[7] Ibid., p. 23.
[8] Ibid., p. 24.
[9] Ibid.

in the modern world? Or, more precisely, what has taken the *essential* place occupied by myth in traditional societies?"[10] "It seems unlikely that any society could completely dispense with myths, for, of what is essential in mythical behaviour – the exemplary pattern, the repetition, the break with profane duration and integration into primordial time – the first two at least are cosubstantial with every human condition."[11]

The absence of myths in modern society, thus, is an illusion. In reality, because human nature has not changed, modern cultures too are founded on myths. All that has happened is a shift in the type of myths that inspire us: our preoccupations are different from those of our ancestors, so our myths too are different. It is a mistake to study the *old* myths, and to conclude that, since we no longer take *them* seriously, we no longer depend on myths.

To understand the mass delusions that possess our present-day society, we must uncover the myths that shape our collective thinking *today*. Let us briefly review some of these myths.

❖

George Steiner refers to the intellectual, political, and social ideologies of the nineteenth and twentieth centuries as surrogate creeds, anti-theologies, meta-religions, or mythologies.[12] These ideologies emerged as a result of the decline of formal religion since the Renaissance. Thanks to the growth of knowledge, Western man's absolute belief in God, which had guided him for centuries, suddenly came to an end. This created a spiritual vacuum and the longing for a new, equally powerful subject of belief: "Where there is a vacuum, new energies and surrogates arise. Unless I read the evidence wrongly, the political and philosophic history of the West during the last 150 years can be understood as a series of attempts – more or less conscious, more or less systematic, more or less violent – to fill the central emptiness left by the erosion of theology."[13]

Steiner discusses three ideologies: Marxism, Freudian psychoanalysis, and Lévi-Strauss's structuralism. These systems of ideas have several characteristics in common: "totality, by which I simply mean the claim to explain everything; canonic texts delivered by the founding genius; orthodoxy against heresy; crucial metaphors, gestures, and symbols."[14] And it is these characteristics that betray their mythological nature: "The major mythologies constructed in the

[10] Ibid. [11] Ibid., p. 31.

[12] George Steiner, *Nostalgia for the Absolute* (Toronto: CBC Enterprises, 1974), p. 2.

[13] Ibid.

[14] Ibid., p. 4. We will also encounter these three ideologies in chapter 3, where we will see that, unsurprisingly, they are based on pseudoscientific theories.

West since the early nineteenth century are not only attempts to fill the emptiness left by the decay of Christian theology and Christian dogma. They are themselves a kind of *substitute theology*. They are systems of belief and argument which may be savagely anti-religious, which may postulate a world without God and may deny an afterlife, but whose structure, whose aspirations, whose claims on the believer, are profoundly religious in strategy and in effect."[15]

Isaiah Berlin[16] shows that, in their attempt to explain social evolution "scientifically," the modern social theories were compelled to ignore the role played by individuals. Human history, according to these theories, is controlled by some mysterious forces and processes – variously represented as class struggles, cultural clashes, geo-political conditions, technological changes, etc. While described in scientific terms, these mighty forces and processes are perceived as supernatural, mythological entities. They manage to explain social evolution only by remaining unexplained themselves, so in the end, these theories – in reality, pseudosciences – are no different from the religious beliefs of the past: "There has grown up in our modern time a pseudo-sociological mythology which, in the guise of scientific concepts, has developed into a new animism – certainly a more primitive and naive religion than the traditional European faiths which it seeks to replace."[17]

Eliade compares the modern political myths with the classical myths: "Eschatological and millennialist mythology recently reappeared in Europe in two totalitarian political movements. Although radically secularized in appearance, Nazism and Communism are loaded with eschatological elements: they announce the end of this world and the beginning of an age of plenty and bliss."[18]

Both Communism and Nazism were seen by their followers as the modern equivalent of the struggle between good and evil – a common mythological theme. Communism is based on "one of the great eschatological myths of the Middle Eastern and Mediterranean world, namely: the redemptive part to be played by the Just (the 'elect,' the 'anointed,' the 'innocent,' the 'missioners,' in our own days by the proletariat), whose sufferings are invoked to change the ontological structure of the world. In fact, Marx's classless society, and the consequent disappearance of all historical tensions, find their most exact precedent in the myth of the Golden Age which, according to a number of traditions, lies at the beginning and the end of History."[19]

[15] Ibid.

[16] Isaiah Berlin, "Historical Inevitability," in *Four Essays on Liberty* (Oxford: Oxford University Press, 1969). [17] Ibid., p. 110.

[18] Mircea Eliade, *Myth and Reality* (New York: Harper and Row, 1975), p. 69.

[19] Eliade, *Myths, Dreams, and Mysteries*, pp. 25–26.

As for the other great political myth of the twentieth century, "in its effort to abolish Christian values and rediscover the spiritual sources of the 'race' – that is, of Nordic paganism – Nazism was obliged to try to reanimate the Germanic mythology."[20] Thus, "the 'Aryan' represented at once the 'primordial' Ancestor and the noble 'hero,' … the exemplary model that must be imitated in order to recover racial 'purity,' physical strength, nobility, the heroic 'ethics' of the glorious and creative 'beginnings.'"[21]

Some of our myths are embodied in literary works, movies, television shows, sports, and popular entertainment. Archaic societies had no need for such distractions, because in their normal life – daily work, hunting, war, family and social activities – they were constantly reenacting sacred myths. Having desacralized our world, and especially our work, we had to invent some useless activities, collective and personal, as substitutes for the reenactment of myths.[22]

For example, a popular myth in current American culture is the myth of the lone saviour: "A community in a harmonious paradise is threatened by evil: normal institutions fail to contend with this threat: a selfless superhero emerges to renounce temptations and carry out the redemptive task: aided by fate, his decisive victory restores the community to its paradisal condition: the superhero then recedes into obscurity."[23] Variations of this myth form the main theme in countless movies and television series, and its popularity can be explained by comparing it with the old religious myths: "The supersaviors in pop culture function as replacements for the Christ figure, whose credibility was eroded by scientific rationalism. But their superhuman abilities reflect a hope for the divine, redemptive powers that science has never eradicated from the popular mind. The presentation of such figures in popular culture has the power to evoke fan loyalties that should be compared with more traditional forms of religious zeal."[24]

Similarly, "the characters of the comic strips present the modern version of mythological or folklore Heroes."[25] For instance, "the myth of Superman satisfies the secret longings of modern man who, though he knows that he is a fallen, limited creature, dreams of one day proving himself an 'exceptional person,' a 'Hero.'"[26]

Cultural fashions – in literature, art, music, philosophy, even science – act in effect as modern mythologies: "One of the fascinating aspects of the 'cultural fashion' is that it does not matter whether the facts in question and their

[20] Ibid., p. 26. [21] Eliade, *Myth and Reality*, p. 183.
[22] Eliade, *Myths, Dreams, and Mysteries*, p. 37.
[23] Robert Jewett and John S. Lawrence, *The American Monomyth* (Garden City, NY: Anchor/Doubleday, 1977), p. xx. [24] Ibid.
[25] Eliade, *Myth and Reality*, pp. 184–185. [26] Ibid., p. 185.

interpretation are true or not. No amount of criticism can destroy a vogue. There is something 'religious' about this imperviousness to criticism.… Their popularity, especially among the intelligentsia, reveals something of Western man's dissatisfactions, drives, and nostalgias."[27]

Paul Kurtz[28] discusses the similarities between classical religions and modern belief systems. He accepts the fact that human beings are susceptible to irrational beliefs, that we are possessed by a "transcendental temptation." But, he says, we must find a way to overcome this weakness, because a society dominated by myths faces great dangers: "The transcendental temptation lurks deep within the human breast. It is ever-present, tempting humans by the lure of transcendental realities, subverting the power of their critical intelligence, enabling them to accept unproven and unfounded myth systems. Can we live without myths? Can we overcome the defect, as it were, in our natures? Is it so rooted in our natures that it cannot be overcome, but will crop up in generation after generation, the forms and functions of the transcendental temptation the same, with only the content different?"[29]

Although the growth of science seems to offer a hope for overcoming it, we must remember that "these are relatively recent developments and of short duration in human history.… The transcendental temptation has held sway for millennia, and to hope to mitigate or obviate its continued power may be to engage in wishful thinking.… What guarantee do we have that science too will not be overwhelmed and superseded by new faiths of unreason commanding human imagination?… One cannot predict the future course of human history with any degree of confidence. Regrettably, often the unthinkable becomes true. Will the unimaginable again overtake us, as we slip into a new dark age of unreason? The only option for us to prevent this is to continue to use the arts of intelligence and skeptical criticism against the blind faiths, old and new.… Is there any hope that a scientific, secular, or humanist culture can develop and prevail, devoid of transcendental myths?… If salvation myths are no longer tenable, what will take their place? The dilemma is always that new faiths and new myths may emerge, equally irrational."[30]

Science, however, has been redefined in our universities to mean a blind pursuit of mechanistic theories – whether sound or not, whether useful or not. Science, thus, has *already* been "overwhelmed and superseded by new faiths of unreason" – by the mechanistic dogma. *The mechanistic belief is the new myth that has emerged to replace the old ones.*

[27] Mircea Eliade, *Occultism, Witchcraft, and Cultural Fashions: Essays in Comparative Religions* (Chicago: University of Chicago Press, 1978), p. 3.

[28] Paul Kurtz, *The Transcendental Temptation: A Critique of Religion and the Paranormal* (Buffalo, NY: Prometheus Books, 1991).

[29] Ibid., pp. 477–478.

[30] Ibid., pp. 481–482.

The Mechanistic Myth

1

In this book we are concerned with one particular myth – the *mechanistic* myth; and we are especially concerned with its latest manifestation – the *software* myth. Mechanism is the belief that everything can be represented as a *hierarchical structure*; that is, as a structure of things within things. This is true, we are told, because every entity is necessarily made up of simpler entities, which are in their turn made up of even simpler ones, and so on, down to some basic building blocks.

Thus, if we want to understand a complex phenomenon, all we have to do – according to the mechanistic doctrine – is discover what simpler phenomena make it up. Then, for each one of those, we must discover what phenomena make *it* up, and so on. Clearly, if we continue this process to lower and lower levels of complexity, we are bound to reach, eventually, phenomena simple enough to understand intuitively. So, by understanding those simple phenomena and the process of simplification that revealed them, we will understand the original, complex phenomenon. Ultimately, working in this fashion, everything that exists in the world can be understood.

Similarly, if we want to build a complicated machine, all we have to do is design it as a combination of subassemblies. Because the subassemblies on their own are simpler than the whole machine, they are easier to design and make. Then, we design the subassemblies themselves as combinations of simpler subassemblies, the latter as combinations of even simpler ones, and so on, down to some small parts that can be made directly.

If we want to study a set of related entities – the people in an organization, the parts stored in a warehouse, the various types of animals – all we have to do is depict them with a hierarchical classification. We divide them first into several categories in such a way that all the entities in a category share a certain attribute. Then, we divide each category into several smaller ones on the basis of a second attribute, and so on, until we reach some categories where the entities share all their important attributes and are therefore very similar. In the case of an animal classification, for example, we may divide them into wild and domestic, the domestic ones into types like horses, chickens, and dogs, and finally each type into various breeds.

If we wonder how linguistic communication works, we start by noting that language is made up of sentences, sentences are made up of clauses, and clauses are made up of words. Words correspond to the facts that exist in the world – nouns for objects, verbs for actions, adjectives for properties, and so on. Thus, since everything in the world can be represented as a hierarchical structure, it

seems that what we do when communicating is create hierarchical structures of linguistic elements which correspond to the structures that exist in the world.

Finally, if we want to create large and complex software applications, we must start by breaking them down into modules. We then break down each module into smaller ones, and so on, until we reach some simple software constructs, which we can program directly. This method, clearly, allows us to implement the most complex applications with skills no greater than those required to program the smallest constructs.

It appears, thus, that the mechanists are right: everything in the world can indeed be represented with a hierarchical structure. The explanation for this versatility lies in the two principles that constitute the mechanistic philosophy: reductionism and atomism. Reductionism assures us that everything can be represented as a combination of simpler things; at the same time, atomism assures us that there is an end to this reduction, that we will eventually reach some *elementary* entities, which cannot be further divided into simpler ones. Together, therefore, these principles assure us that every problem can be solved.

The term "mechanism" derives from the fact that in the seventeenth century, when this philosophy was established, the elementary entities were believed to be the simplest *mechanical* entities; namely, bits of matter. All phenomena, in other words – from those encountered in the study of mechanics to those encountered in the study of minds and societies – were believed to be reducible, ultimately, to the phenomena associated with the motion of bits of matter.

Formal reductionism still claims this, although the idea is so absurd that most scientists today avoid discussing it. In any case, rigorous mechanism – that is, a reduction to truly elementary entities – is too difficult to practise, so it is an easier variant that has been adopted in universities as "the method of science." This form of mechanism employs *partial* reductionism, and academics like it because it can make trivial activities resemble scientific research. Thus, to explain a given phenomenon we no longer have to *actually* reduce it to some basic, indivisible entities; we are free to end the reduction at any convenient level, and simply *call* those entities elementary. Theories grounded on this method explain nothing, of course; but they *look* scientific, so the method is very popular.

Mechanism is also described as a method that leads to precise and complete explanations – *mathematical* explanations, in particular. It is easy to see why mathematical models are logically equivalent to the hierarchical structures of mechanism: Mathematical systems are themselves based on hierarchical

structures. In a given system, a complex theorem can be expressed as a combination of simpler theorems, which can then be reduced to even simpler ones, and so on, until we reach the premises, axioms, and basic elements upon which the system is founded. Thus, since we can always invent a mathematical system whose entities correspond to entities from the real world, a phenomenon that can be represented with a hierarchical structure can also be represented mathematically.

And indeed, those aspects of the world that have been successfully explained through reductionism and atomism also have exact, mathematical models. They include the subjects studied by sciences like physics, chemistry, and astronomy, and their applications – engineering, manufacturing, construction. Mechanism and mathematics, however, have been far less successful in other areas. Sciences like biology, physiology, and medicine benefit to some extent from mechanistic theories, but their main problems are non-mechanistic. As for those sciences that study human phenomena – psychology, sociology, linguistics, economics, politics, history, anthropology – their problems are almost entirely non-mechanistic. Finally, our software-related activities, despite their dependence on computers and hence on engineering, entail largely non-mechanistic problems.

So the mechanistic principles only *appear* to be universal. In reality, they are useful for some phenomena and useless for others. In three hundred years of mechanistic philosophy, *not one* mechanistic model was successful in the human sciences. Countless mechanistic theories have been advanced, and more are being advanced today than ever before, but when a theory fails no one tries to understand the reason. The response, invariably, is to start working on another mechanistic theory. Reductionism and atomism have been so successful in those fields where they do work that science is now universally identified with mechanism. For most of us, science means simply the attempt to extend the success of mechanism to every other aspect of the world. So an individual is perceived as scientist simply if pursuing a mechanistic theory. No one cares whether the theory works or not, or whether mechanism is valid at all in that particular field. Thus, while known as the method of science, mechanism is now largely the method of charlatanism.

2

The obsession with finding a mechanistic representation for every aspect of the world is especially silly in view of the fact that it is quite easy to see why mechanism *cannot* explain every phenomenon. All that the researchers have to do is study with an open mind any one of their failures. For, when mechanism

fails, the reason is always the same: the phenomenon is too complex to be represented with a neat structure of things within things. We will examine these failures in the following chapters, but from what we have discussed so far we can already recognize why mechanism is limited.

In the hierarchical structure that is the mechanistic representation of a phenomenon, what determines the relations between levels is the totality of attributes possessed by the structure's elements. Thus, for the structure to provide an exact and complete explanation, the elements must possess these attributes in such a way that the relations we see in the structure are the *only* relations between them. But this is rarely true.

The entities that make up the world possess *many* attributes, and are therefore interrelated in many different ways. For certain types of phenomena, though, a few of these attributes, and the resulting relations, are much more important than the others; and, sometimes, these attributes also happen to define a hierarchical relationship. Thus, if we agree to *ignore* the other attributes, a hierarchical structure will provide a useful approximation of reality. For these phenomena, then, we note that mechanistic theories work. Putting this in reverse, for certain types of phenomena the other attributes *cannot* be ignored, so the phenomena *cannot* be usefully approximated with a hierarchical structure; for those phenomena, then, we note that mechanistic theories fail.

Recall the earlier examples. Hierarchical classifications of things are possible only if we take into account *some* of their attributes (one attribute, or a small set of attributes, per level) and ignore the others. It is impossible to include *all* their attributes in one classification. Thus, animals can be divided into wild and domestic, into types, and into breeds, as we saw. But this is just *one* way to represent them. The biological classification – dividing animals into classes, orders, families, genera, and species – is based on different attributes, and the resulting hierarchy is different. Tigers and horses belong to different categories (wild and domestic) in one classification, but to the same category (class of mammals) in the other. Clearly, there are many ways to classify animals, all valid and useful; and each classification can take into account only *some* of their attributes. It is impossible to represent *all* their attributes in *one* hierarchical structure. The totality of animals and their attributes is, therefore, a non-mechanistic phenomenon. A mechanistic representation – one structure – is valid only if we agree to study animals from one narrow perspective; it becomes useless as soon as we remember their other attributes.

Similarly, we can represent an appliance as a hierarchy of parts and subassemblies only if we restrict ourselves to those attributes that determine their position and function in that appliance. For, the same parts and subassemblies form at the same time *other* hierarchical structures, based on other attributes –

their cost, or supplier, or life expectancy. We purposely design appliances in such a way that the other attributes can be ignored in the manufacturing process. But the attributes *are* important when we study the appliances from *other* perspectives. And the other hierarchies are usually different from the one that represents the physical and functional attributes; for example, parts made by the same supplier may belong in different subassemblies. It is impossible to represent the parts and *all* their attributes in *one* hierarchical structure. Again, a mechanistic representation is valid only if we can restrict ourselves to one view.

Sentences appear to form a neat hierarchy of clauses and words only if we take into account the syntactic structure and ignore the *meaning* of the words. For, the things represented by words possess many attributes, and are therefore related through many structures. Consequently, the words themselves are related through many structures, which are different from the syntactic one. It is impossible to depict, with a syntactic structure alone, everything that a sentence can convey.

Finally, software applications appear to form perfect hierarchies of smaller and smaller entities (modules, blocks of statements, statements) only if we study them from the perspective of *one* attribute. The attributes of a software entity are such things as files, variables, subroutines, and business practices. Software entities possess many attributes, and are therefore related through many structures – one structure for each attribute. The programming theories attempt to simplify programming by forcing us to view each application as a neat hierarchical structure of software entities. Thus, since applications consist in fact of multiple, simultaneous structures, it is not surprising that the theories keep failing.

❖

Mechanism, then, is not the solid scientific concept it is believed to be. Its prestige is due largely to its early successes in the exact sciences, and especially to its successes relative to the scholastic doctrines of the Middle Ages, which it was displacing. Just as the religious philosophy had been accepted for centuries as the absolute truth, the mechanistic philosophy was seen now as an absolute method – a method that can explain everything. Mechanism became, in effect, a new religion. It seems that societies cannot exist without some great ideas to inspire them – ideas that people can accept blindly.

Most of us perform both rational and irrational acts, but the two kinds appear to us equally important. In the easier pursuits, when our knowledge guarantees success, we are completely rational and follow only sound and proven principles. But in difficult pursuits, when our knowledge is insufficient,

we behave irrationally. Irrationality, thus, emerges when we have no proven theories to rely on: if we wish to understand a given phenomenon but lack the necessary knowledge (and if, in addition, we believe that all phenomena can be understood as we understand the simple ones), we are bound to invent a fantastic concept and use *it* as explanation. This is how myths are born. People are always in need of myths, because there is always much that is unknown or unpredictable, in any society. Consequently, people always display a blend of rational and irrational thinking, rational and irrational activities.

We like to justify our acts by basing them on accepted concepts, but we are less keen on justifying the concepts themselves. As a result, we perceive the two kinds of activities, rational and irrational, as equally effective. The former become pursuits like science and business, while the latter make up pursuits like magic and superstitions. But the *individual* activities that make up these pursuits are very similar: they are always logical and consistent, always grounded on an accepted concept. The difference is only that the concept is a valid theory in one case and a fantasy in the other.

Thus, as we will see in the course of this book, it is possible for a person, and even an entire society, to engage in activities that are perfectly logical *individually*, while the body of activities as a whole constitutes a delusion. So, to judge whether a certain pursuit is rational or not, it is not enough to study the logic of the *individual* activities which make up that pursuit.

In chapter 3 we will learn that the best way to distinguish between rational and irrational pursuits is by studying, not the successes, but the *falsifications* of an idea. Just as important is how people *react* to these falsifications. Serious researchers react by doubting the idea. Most people, however, react by ignoring the falsifications, or by contriving ways to cover them up. They never admit that the idea has been refuted. This shows that, for them, the idea is not a rational pursuit but a belief.

Astrology, for instance, has been around for thousands of years, and we could always show that it doesn't work. All we have to do is note the predictions made in the course of a year, and then count how many actually materialized. Believers, though, never do this. Similarly, today we can note the mechanistic claims in a field like linguistics, economics, or software, and count how many actually materialize. But, again, believers never do this. Mechanism continues to be trusted, regardless of how successful or unsuccessful it is.

We will see that it *is* possible to distinguish between the two types of thinking, the scientific and the pseudoscientific. And we will see that what the mechanists do is simply ignore the falsifications, just like the traditional pseudoscientists. Thus, our mechanistic theories – while embraced by famous scientists, taught in respected universities, and practised throughout society – form in reality a new kind of pseudoscience.

The conclusion must be that mechanism does not function as scientific doctrine in our society, but as myth. It is precisely the lack of doubts that betrays its mythical status. When a method works, we are not afraid to debate it, modify it, or replace it with a better one. Only concepts that cannot be proved become unquestionable truths. Were mechanism perceived merely as an important research method, we would rely on it in those fields where it is useful, and seek other methods in those fields where it fails. But this is not what we see. Mechanism is considered the *only* valid method of science, in *all* fields. Academics are trained to think mechanistically, and are expected to pursue only mechanistic ideas, regardless of whether these ideas are useful or not. Moreover, non-mechanistic ideas are dismissed as "unscientific," even if shown to be useful. We have redefined science, in effect, to mean simply the pursuit of mechanism. And as a result, our academic institutions have degenerated into a self-serving bureaucracy.

Recall the earlier quotations: modern societies are founded on myths, just like the primitive ones; myths are the most important form of collective thinking; myths are thought to express absolute truth; myths serve as models and as justification for all human action; and so on. Thus, if science and its applications – especially the pursuits we call technology – serve as warrant for our actions and decisions, and if science is grounded on mechanism, then, for us, mechanism serves the purpose of myth. When we judge something as important or unimportant, as useful or useless, as moral or immoral, as valid or invalid, simply by invoking a scientific or technological concept, we judge it in effect by invoking the mechanistic myth.

3

Myths can be good. When people possess only limited knowledge, as in a primitive society, most phenomena they observe are unexplainable. They have nothing to lose then, and much to gain, by attributing these phenomena to some mythical powers. The myths replace their anxiety and fears with a sense of confidence and security. The fact that this confidence is based on false assumptions does not detract from the value of the myths, since the primitives cannot arrive at the correct explanation in any case. If they wish to understand what caused a certain disease, for example, and they know nothing about microorganisms, the assumption that it was caused by sins, or demons, or black magic, is quite effective. As they cannot *cure* the disease, these beliefs provide at least the comfort of knowing its origin. With this comfort they are in a better position to face other problems, so they can accomplish more in those fields in which they *are* knowledgeable.

Thanks to the importance of myths, the individuals who provide myth-related services – magicians, shamans, astrologers – enjoy great respect. Their knowledge, limited as it is to myths, is necessarily specious. Nevertheless, just as the myths themselves fulfil a vital function in society while being in fact unreal, the services provided by these experts are crucial even while being specious. The experts, as a result, become a powerful elite. But this position is well-deserved: if a society benefits from its myths, and if the practice of myths requires a certain expertise, then the individuals who possess this expertise are as essential to society as the myths themselves. Thus, when the myths are good for a society, an elite whose existence depends on these myths is a good elite.

Myths, however, can also be bad. A society may reach a point in its evolution where enough knowledge has been accumulated to attain better explanations than what the myths can provide. Most likely, the new explanations include mythical elements of their own, rather than being completely rational. Even so, being closer to reality, they constitute an improvement. In retrospect, then, the practical benefits of abandoning the old myths are obvious. But the actual transition is difficult. The old myths are usually part of a belief system that had guided society for generations, and it takes more than the promise of an improvement to abandon them. So the same myths that hitherto *served* society are now turning *against* it, by preventing it from enjoying the benefits of the new knowledge. The good myths become bad.

The elite too – those experts whose privileged position depends on the myths – is now turning against society. Because they would be redundant without the old myths, the experts continue to praise their value even as society no longer needs them. Whereas formerly they were *practising* those myths, now they are *enforcing* them. They describe this struggle as an effort to preserve some proven social values, but in reality it is their own privileges that they want to preserve. Thus, when the myths turn from good to bad, the elite too becomes bad.

The best-known transition in Western history is the Renaissance and the Scientific Revolution, which took place between the fifteenth and seventeenth centuries. This is when modern science, expressed through the mechanistic philosophy, replaced the religious myths that had dominated Europe for more than a thousand years. One of the most remarkable aspects of this transition is the ferocity with which the church – guardian of the old myths – fought to prevent it. Previously, the church was perhaps a good elite, insofar as myths like the idea of salvation could provide some comfort in an age when science had little to offer. But now that the real benefits of the growing knowledge exceeded the emotional benefits of myths, the only way the church could maintain its power was by suppressing that knowledge. This was the task of the Inquisition.

Thus, regardless of how one feels about the value of the religious myths in earlier times, we all agree that obstructing the truth, and torturing and burning alive innocent people, is not something that a good elite would do. The myths, and with them the elite, had become bad.

❖

The foregoing analysis should help us to recognize that a similar transition is taking place in our own time. What is being defended now is *mechanism* – the very myth that was being *repressed* in the earlier transition. And the elite struggling to maintain its power is embodied now in our educational institutions – our universities, in particular. The academic bureaucrats are the greatest beneficiaries of the mechanistic myth, as this myth affords them a privileged position in society regardless of whether their activities are useful or not. So it is not surprising to see them defend the mechanistic ideology as fiercely as the church was defending earlier the religious one.

When astrology was important, astrologers retained their position regardless of whether their predictions were correct or not; when alchemy was important, alchemists continued to be trusted regardless of whether their transmuting methods worked or not; and when religion was important, the church bureaucracy retained its power regardless of whether its promises of salvation materialized or not. Today, *mechanism* is important, so we continue to trust and respect the academic bureaucrats even as the mechanistic theories are failing. As we will see in the following chapters, it is quite easy to prove that these theories are fraudulent; and yet we treat their defenders as scientists, not as charlatans.

As part of its power, the academic elite controls education. And it has used this monopolistic position to turn the process of education into a process of indoctrination: all we are taught is what can be explained mechanistically. Thus, while promoting knowledge, intelligence, and creativity, the academic elite has redefined these qualities to mean, not the utmost that human minds can attain, but merely the skills needed to follow the mechanistic ideology: knowledge of the latest mechanistic theories, the intelligence to appreciate the mechanistic principles, and the creativity to accomplish a task with mechanistic methods alone. Mechanism is not just *practised* – it is *enforced*. Together with the corporations (the other beneficiaries of the mechanistic myth), and protected by irresponsible governments, our universities have brought about a social order that is, in effect, a new form of totalitarianism.

Totalitarian ideologies differ in detail, but their goal is always the same: to create a perfect society. For us, this means a society founded upon solid, mechanistic principles. We have already proved the value of these principles in

certain areas – in the exact sciences, for instance, and in manufacturing – so all we have to do now is extend their use to every other aspect of human life.

Here is how we can accomplish this: Since everything can be represented with hierarchical structures, we can improve our performance by breaking down all challenges into simpler and simpler ones. In the end, we will only need to deal with the terminal elements of these structures; that is, with trivial issues. In practice, the structures will be embodied in theories and methods, and the terminal elements will be some simple rules. Thus, just by obeying these rules, anyone will be able to perform tasks that previously demanded much knowledge and experience.

Better still, once we represent our problems with hierarchical structures, we can build devices that embody these structures. Then, to solve a given problem, all we need to know is how to operate a device. The skills required to operate devices are easier than those required to solve problems, so we will all be more productive: first, because devices eliminate the lengthy learning periods we needed in the past, and second, because devices are faster, more accurate, and more dependable than humans.

Finally, with our latest invention, computers, we can implement even those structures that are too large or too complex for the traditional devices. Thanks to the power and versatility of software, practically every human endeavour can be translated into a series of easy acts – the acts required to operate a software device. From simple calculations to difficult decisions, from personal concerns to business issues, we can have a software device for every task. Various types of knowledge are now being incorporated into these devices, and made available to us through easy-to-use menus, lists, buttons, and the like; in other words, through a hierarchical structure of selections, and selections within selections, corresponding to the hierarchical structure that is the knowledge itself. So, just by purchasing a software device, we will be able to perform almost any task without having to develop that knowledge in our own minds.

Our idea of a perfect society, then, is one where all human affairs have been reduced to the simple acts required to follow methods and to operate devices. The methods and devices are developed by various elites – experts who know how to translate the complexity of the world into concepts simple enough for us to understand. The responsibility of the elites is to represent the world with exact, mechanistic theories; and *our* responsibility is to obey these theories. Anything that cannot be represented mechanistically is unscientific, and hence devoid of value. Thus, as our goal is endless progress, we cannot

afford to spend any time with non-mechanistic notions, even if we might otherwise enjoy it.

If we doubt the efficacy of this scheme, we only need to recall the progress we have made in our *manufacturing* activities. From the handful of simple consumer products available two hundred years ago, and which few people could afford, we have arrived at today's astounding array of sophisticated products, which almost anyone can afford. And we have accomplished this, not by *increasing*, but by *reducing*, the knowledge and skills of the workers who make these products. The secret for the great progress in manufacturing is found, as everyone knows, in concepts like the assembly line (which permits us to employ unskilled workers and to control their output), division of labour and narrow specialization (which permit us to reduce each individual's education and training, and hence the cost of employment), and, in general, fragmentation of the labour process (which reduces all types of work to simple, routine activities, eliminating the dependence on personal skills or initiative) and scientific management (which creates a rigid environment, where everyone is forced to work in the manner dictated by a superior).

These principles are, clearly, an application of the mechanistic ideology: from a rather haphazard series of activities, the manufacturing process has been turned into an exact system – a system that can be represented with a hierarchical structure. In this structure, the elements are the various components, stages, persons, and activities, and the efficiency of this arrangement is assured by the mechanistic concept itself. So there can be little doubt that, to be as efficient in the other fields as we are in manufacturing, we must follow the same principles. We must modify the entire society to resemble, so to speak, a giant factory: each person, each act, each thought, must be designed to function as an element in a giant structure of things within things. We are currently in the process of implementing this idea in our educational and business activities; and soon we will extend it to all social and personal affairs.

Thus, while this may seem paradoxical, it is a fact that if we want to become more efficient we must be *less* knowledgeable, *less* skilled, *less* experienced. It is our natural tendency to gain knowledge that slows progress. So we must stop trying to develop such old-fashioned qualities as expertise or individuality, and admit that we can accomplish more by being an insignificant part in a great whole. We must allow the elites, who have proved the value of this idea in fields like manufacturing, to design that great hierarchical social structure for us. And we must restrict ourselves to those activities which they prescribe.

This ideology – totalitarianism – is quite old, in fact, and was always appreciated by enlightened leaders. The reason it seems new is that only in the twentieth century it became practical on a large scale. The first attempts,

Communism and Nazism, were rather crude and violent. They were *political* movements, and failed. We learned from these mistakes, however, and we rely now on universities and corporations, instead of political institutions, to implement it. Our totalitarianism is better, and it will succeed.

4

Despite its obvious benefits, totalitarianism is not without critics. The first objection concerns the process of dehumanization that inevitably accompanies it. Thinkers of various outlooks – philosophers, sociologists, science-fiction authors – have been warning us for a hundred years that we are being turned into automatons. The vision of a society where human beings are treated as parts of a giant machine, and restricted to some simple and repetitive acts, is not very appealing – even if this is done in the name of efficiency or progress.

As answer to this objection, we point to the great improvements in standard of living and in life expectancy that all sections of society have enjoyed thanks to totalitarianism. Thus, as in any social project, our decision to pursue this ideology amounts to a compromise: we are trading more and more aspects of our humanity for greater and greater prosperity. This has worked out well so far, and there is no reason to doubt that we can continue this trade in the future. Besides, people don't seem to mind this dehumanization: following rules and methods is easier than developing expertise, and most of us are quite happy to be merely parts of a whole, as this absolves us from responsibility for our acts and choices.

More recently, a second objection has arisen to the totalitarian ideology. This objection concerns the environmental problems associated with infinite progress. Specifically, we are reminded that, even if we agree to become full-fledged automatons in our unending quest for prosperity, we may never get there. Growth is limited by such factors as increasing pollution and diminishing natural resources, so the assumption that an ideology which worked in the past will continue to work in the future is invalid. In other words, our ideology is wrong, not so much because it dehumanizes us, but because at the current rate of growth we will destroy ourselves by ruining the environment *before* we do it by becoming automatons.

Unlike the first one, this objection is gaining in popularity, owing largely to the ease with which we can delude ourselves that we care about the environment. All we need to do is read books and articles, watch television documentaries, and discuss the issue from time to time – while keeping our lifestyles and expectations unchanged. This stratagem permits us to feel

concerned and involved, without having to give up anything. In reality, an endless increase in prosperity is possible only through an exponential growth in production and consumption. To prevent the environmental problems, therefore, we would have to reduce our prosperity even more than we would have to in order to prevent our dehumanization. And we already saw what is our attitude on the latter. People who agree to pay for prosperity by living their lives as automatons are not likely to renounce the same prosperity for the benefit of future generations. So, despite its apparent popularity, the second objection will not stop the spread of totalitarianism any more than the first objection did in the past.

It is not these two objections that ought to preoccupy us, however, but a *third* one; namely, the risk that the totalitarianism we are being offered may not be at all what it is said to be. We believe the problem is simply whether the price we pay for progress and prosperity is too high, while the real problem is whether we are getting anything at all for this price. The elites justify the totalitarian ideology by telling us that it is grounded on mechanistic, and hence scientific, principles. But if these principles are becoming less and less useful, the elites are deceiving us – regardless of the price we are willing to pay.

The justification entails a succession of ideologies: mechanism, scientism, utopianism, totalitarianism. The belief in mechanism leads to scientism – the application of mechanistic concepts in the study of minds and societies, where they cannot work. Then, despite the failure of their theories, the mechanists conclude that society can be greatly improved by actually implementing these theories; so, scientism leads to utopianism. Finally, everyone agrees that the only practical way to carry out this project is through totalitarianism: by allowing an elite to control all aspects of society.

Totalitarianism, thus, is justified by pointing to its origin, mechanism. Our infatuation with mechanism is so strong that even when noticing its failures, or its harmful consequences, we still do not question the ideology itself. So we accept and respect the idea of totalitarianism, even when criticizing it, simply because we believe it to be scientific. We have no evidence that totalitarianism works, but we cannot help trusting those who advocate it.

5

The declining usefulness of mechanism has engendered a new phenomenon: charlatanism practised in the name of science or in the name of business. This charlatanism consists in the promise to solve a non-mechanistic problem with mechanistic methods. Since mechanism is universally accepted as "the method of science," we trust implicitly anyone who invokes the mechanistic principles.

Thus, once we decided to measure the value of an idea solely by its mechanistic qualities, it became impossible to distinguish between serious mechanistic ideas and mechanistic *delusions*.

Mechanistic delusions have always been part of our culture. Until recently, however, their harm was overshadowed by the mechanistic *successes*. Today, fewer and fewer problems have simple, mechanistic solutions, so the harm caused by delusions exceeds the benefits derived from successes.

Totalitarianism, in particular, is a mechanistic delusion. We like totalitarianism for the same reason we like all other mechanistic ideas: because it offers what appears to be simple solutions to difficult problems. However, while the pursuit of an ordinary mechanistic delusion means merely a waste of resources, the pursuit of totalitarianism can lead to the collapse of society. For, if the world is too complex to be improved mechanistically, the claimed benefits are a fantasy, while the price we pay for them is real. Our problems are getting bigger, while our minds are getting smaller: if we restrict ourselves to mechanistic thinking, we leave our non-mechanistic capabilities undeveloped; so we cope perhaps with the simple, mechanistic problems, but the complex, non-mechanistic ones remain unsolved, and may eventually destroy us.

In universities, the charlatanism is seen in the activity known as research. The rule is simple: any work that follows the mechanistic principles of reductionism and atomism is deemed scientific, and is therefore legitimate. Whether these principles are valid or not in a given field, or whether the resulting theories work or not, is immaterial. Thus, when faced with a problem in the human sciences, all one has to do is perceive it as a hierarchical structure. The problem can then be broken down into smaller and smaller parts, until reaching problems simple enough to describe with precision. But this method, borrowed from the exact sciences, fails when applied to human phenomena. It fails because human phenomena consist, not of one structure, but of multiple, interacting structures.

So the researchers are admired for the rigour with which they study those small problems, even while the real problem remains unsolved. Clearly, their only defence is that they are following the mechanistic principles. But why should principles that are useful in modeling the material world be accepted without reservation in the study of minds and societies? As soon as we question the value of mechanism in these fields, any research project grounded on mechanism changes from scientific pursuit to mechanistic fantasy. What stands between perceiving these academics as scientists or as charlatans, then, is only our blind acceptance of the mechanistic ideology.

In business, the charlatanism is seen in the activity known as marketing. The elites, we saw, tell us that our future must be based on an endless growth in production and consumption, and that this can only be achieved through

mechanistic methods. But if, in fact, there is less and less that *can* be discovered or improved mechanistically, the only way to attain the required growth is by replacing the making of useful things with the making of whatever can be made mechanistically (that is, efficiently and profitably). To put this differently, if the old experts – scientists, inventors, entrepreneurs – cannot keep up with our demand for growth, we must replace them with a new kind of experts: charlatans, who know how to make useless things appear important, and thereby help us to delude ourselves that our system is working just as it did in the past.

Thus, from its modest origin as a complement to trade, the process of selling has become more important than the merchandise itself. The fact that it is possible to cheat people, to persuade them to buy something that is not what it appears to be, is now the driving force of the economy. Deceptive advertising – messages purporting to inform while in reality exploiting human weaknesses and ignorance – is no longer limited to domains like fashion or cosmetics, but covers practically all products and services. Dishonest techniques (testimonials and success stories, background music, pictures of happy faces, and the like) are widely employed in order to influence, distract, and confuse. These techniques are logically equivalent to lying (they are needed precisely because the usefulness of those products and services cannot be proved), but we no longer notice this. Language itself has ceased to be a means of communication, and is used as a kind of weapon: words are carefully chosen, not to convey information, but to deceive and to manipulate.

Finally, and most disturbingly, the idea of "selling" has transcended the domain of commerce and is now found in every activity where there is an opportunity to influence people. From what we say in a résumé to what governments say in their policies, from business meetings to military decisions, from lectures and seminars to television news and documentaries, it is vital that we know how to *persuade* our audience; that is, how to *mislead* – how to use special effects so as to make unimportant things appear important, and important things unimportant.

The fact that we have to lie so much ought to worry us, ought to prompt us to doubt our system. We need more and more lies, obviously, because our *real* achievements do not fulfil our expectations. We have experienced continuous growth ever since the Scientific Revolution, and our world view has evolved accordingly: we have yet to accept the fact that there is a limit to discoveries and improvements. We are still making progress, of course, but at a slower and slower rate. Since the exponential growth that we are accustomed to cannot be sustained indefinitely, we are now supplementing the real growth with an imaginary one, based on fantasies. But instead of interpreting the perpetual increase in charlatanism as evidence that our system is failing, we perceive the

charlatanism as a new sort of science, or a new sort of business, and hence its increase as progress.

Much of the current growth, thus, is actually growth in delusions, and in the stupidity necessary in order to accept these delusions. It is as if, having realized that the human capacity for intelligence does not guarantee infinite growth, we are now trying to achieve the same growth by relying instead on the human capacity for stupidity. Like oil and minerals, we treat stupidity as a kind of resource, as something that we can exploit and benefit from. To make the most of this resource, though, human beings must be carefully indoctrinated, in order to neutralize their natural capacity for intelligence. The incessant lies and delusions, then, serve to replace the *reality* that surrounds us with the *fantasies* that – according to the elites – are the world we must strive to create instead.

❖

To summarize, the mechanistic myth has outlived its usefulness. What started as a good myth, helping us to expand our knowledge of the world, has become bad. The same qualities that make mechanism such a useful concept are now turning against us. For, mechanism can only explain *simple* phenomena – those that can be represented with *isolated* hierarchical structures; and in today's world we are facing more and more *complex* phenomena, which can only be represented with *systems* of structures. One reason for the complexity, thus, is that there are fewer and fewer mechanistic phenomena left to be explained. If we want to expand our knowledge today, we must increasingly deal with those phenomena that we chose to ignore in the past – when there were so many simple, mechanistic ones, waiting to be studied. Another reason for the complexity is that, as we keep expanding our knowledge, we are creating ourselves new, non-mechanistic phenomena (the software phenomena are an example).

So the mechanistic myth works against us because it restricts us to mechanistic thinking while our most important problems are non-mechanistic. The past successes of the mechanistic philosophy, together with its irresistible appeal, prevent us from noticing how limited mechanism really is. We are trying to explain everything mechanistically while less and less *is* mechanistic. As a result, we are wasting our resources on absurd ideas, neglecting the real problems. Only *minds* can process complex structures. So, to contend with our current problems, we must develop the highest intelligence and expertise that human minds are capable of. Instead, the mechanistic culture restricts us to novice levels: we are taught to treat every challenge as simple, isolated structures, so we are using only our mechanistic capabilities.

Along with the mechanistic myth, our elites too have turned from good to

bad. The elites defend the mechanistic myth because it is through this belief that they hold their privileged position. Thus, as long as we accept mechanism unquestioningly, all they have to do to gain our respect is practise mechanism. If we judged them instead by assessing the validity or usefulness of their ideas, we would realize how little of what they do is important. We would stop respecting them, and they would lose their elitist position.

So we shouldn't be surprised that our elites praise the mechanistic ideology and cover up the failure of the mechanistic ideas. In the past, when most mechanistic ideas were useful, the elites did not have to resort to lies and delusions; they gained our respect through real achievements. Today, the mechanistic ideas are becoming increasingly worthless; so the only way for the elites to maintain their position is through charlatanism, by *fooling* us into accepting mechanistic ideas.

Mechanism, moreover, has become totalitarian: We are asked now, not just to accept the mechanistic delusions promoted by the elites, but to become devoted mechanists ourselves. Like the elites, we must restrict ourselves to mechanistic thinking and adhere to this ideology regardless of whether our activities are successful or not.

Our totalitarianism, thus, is the ultimate mechanistic fantasy. For, if our problems stem from the declining usefulness of mechanism, it is absurd to attempt to solve them through totalitarianism, which only *adds* to our mechanistic practices. So, when listening to the elites, we are moving in the wrong direction: we are *aggravating* the problems. The elites tell us that totalitarianism is necessary in order to become more efficient. But if it is based on mechanism, and if mechanism itself is less and less useful, how can totalitarianism help us?

6

By way of conclusion, let us speculate on the alternatives to mechanism. We saw earlier that all human societies are founded on myths. For us, since the seventeenth century, the most important myth has been the mechanistic philosophy. Usually described as a shift from religion to science, the transition to mechanism was in fact a shift from religion myths to science myths: all we accomplished was to replace one kind of myths with another. Mechanism is not an ultimate concept, but merely an improvement, a better way to represent the world.

The usefulness of mechanism has been exhausted, however, and it can no longer function as myth: rather than helping us to advance our knowledge, it holds us back now, and allows evil elites to exploit us. There is an urgent need

to abandon it. But it is highly unlikely that, during the next few decades, we can achieve something that no human society ever could – learn to live without myths. The only practical alternative, therefore, is to replace mechanism with a different myth. We must effect, in our lifetime, the next transition: from this naive, seventeenth-century myth, to a modern one, adequate for our time. If we *must* believe in myths, we should at least choose one that can help us to solve *today's* problems.

We will continue to use mechanism, of course, but only where appropriate. What we want to avoid is the mechanistic *delusions*. In those fields where it works, mechanism remains the best method, the best way to represent the world. So what we must do is demote it: from its position as *myth*, to a more modest position, as *method*. Then, we must turn to the *new* myth for inspiration in solving our complex, non-mechanistic problems.

What is left is to decide what belief should replace mechanism as myth. It is obvious that the new myth must be more than just a more sophisticated variant of the mechanistic method. The greatest challenges we face today do not entail merely a larger number of mechanistic problems, or more involved mechanistic problems, but *non-mechanistic* problems. And there is only one way to solve this type of problems: by using our minds. As we will see in chapter 2, our minds excel at solving precisely the type of problems that mechanism leaves unsolved. In our infatuation with mechanism, we have been neglecting these problems. Moreover, we have been neglecting our own, non-mechanistic capabilities: we have been using only a fraction of the capacity of our minds, only what we need in order to think mechanistically.

The next myth, thus, must be a belief in *the unlimited potential of our minds*. Like all myths, this is a fantasy, since the potential of our minds is not unlimited. But we can *believe* that it is; and the very belief will inspire us. In fact, we are using now so little of this potential that, for all practical purposes, it *is* unlimited. Once accepted as myth, the new belief will motivate us to appreciate and to use our non-mechanistic capabilities. And with these capabilities we will accomplish more than we do now.

This process would be similar to the way mechanism itself functioned in the seventeenth century. As we will see in chapter 1, it was its role as myth, rather than its usefulness as method, that imparted to mechanism its strength. It was the *belief* that its potential is unlimited that inspired the seventeenth-century scientists. Had they perceived mechanism as just a new method of research, they would not have had the confidence to propose those radical theories, and the Scientific Revolution would not have happened. Today there are more mechanistic *delusions* than discoveries, so it is obvious that the potential of mechanism is not unlimited. But this fact did not detract from its value in the seventeenth century. All we have to do, then, is undergo a similar process with

the new myth. And this will help us to bring about advances of a different kind: in non-mechanistic knowledge.

If it seems improbable that we can start to believe now in a new myth, we must remember that human societies can adopt any myth. Thus, if we managed to believe for three hundred years that every phenomenon can be represented with a neat structure of things within things (an idea easily shown to be false, as we saw), it shouldn't be so difficult to believe now that the potential of our minds is unlimited.

But regardless of which myth we decide to adopt next, we *must* end our dependence on the mechanistic myth, and on the elites that profit from it. The blind belief in mechanism is destroying our minds, and is preventing us from dealing with our problems. The mechanistic *software* beliefs, in particular, have permitted a powerful *software* elite to arise. In just a few decades, organizations that have in fact little to offer us have attained so much power that they practically control society. As we will see in the course of this book, their power rests almost entirely on mechanistic software delusions, and on the stupidity engendered by these delusions.

Software, thus, has emerged as the most effective means of enforcing the mechanistic dogma. Software should have been our most modern pursuit; instead, degraded by the software elite, it is now merely the most modern way of pursuing a seventeenth-century myth.

The Software Myth

1

The software myth is the idea of software mechanism – the enactment of mechanistic beliefs through software. If traditional mechanism holds that every phenomenon can be represented with a hierarchical structure, software mechanism holds that every phenomenon can be represented with a hierarchical *software* structure. This is true because, once we reduce a phenomenon hierarchically to its simplest entities, these entities can be emulated by means of simple *software* entities. To represent the original phenomenon, all we have to do then is combine these entities hierarchically, and thereby generate a software structure that corresponds to the structure of entities that is the phenomenon itself.

In particular, the phenomena associated with human knowledge can be represented with software. Since any type of knowledge can be reduced hierarchically to simpler and simpler pieces down to some basic bits of knowledge, by incorporating these bits in a software device we can emulate the

original knowledge structure. Then, simply by operating the device, anyone will be able to perform the same tasks as a person who took the time to acquire the actual knowledge.

Software devices, thus, are perceived as substitutes for knowledge, skills, and experience. Whereas in the past we needed much learning and practice in order to attain expertise in a given field, all we need to know now, it seems, is how to operate software devices.

One type of knowledge that we have been trying especially hard to represent with software is *programming* knowledge. If software devices are only now gaining acceptance in our businesses and in our homes, their counterparts in the world of programming have existed since the 1960s. Thus, if the use of software devices as substitutes for expertise still sounds plausible for other types of knowledge, we have already had several decades to assess their value in *programming* work. And, as we will see in chapter 7, the claim that there exist substitutes for programming expertise has proved to be a fraud.

The study of software mechanism in the domain of programming can help us to understand, therefore, the delusion of software devices in general. For, it is the same myth that the elites invoke when promoting knowledge substitutes, whether they address programmers or other workers. Programming is the only domain in which we can, today, actually demonstrate the failure of software mechanism and the dishonesty of the software elites. Thus, we must make the most of this experience. If we understand how the software myth has destroyed the programming profession, we will be in a better position to recognize its dangers, and to prevent it perhaps from destroying other fields of knowledge.

2

The reason it is so tempting to think of software development as a mechanistic process is that software applications are indeed hierarchical structures – modules within modules. No matter how large or complex, it seems that an application can always be depicted as a neat structure of software entities, just as a manufactured object can be depicted as a neat structure of parts and subassemblies.

As we do in manufacturing, therefore, we should break down the process of software development into smaller and smaller parts, until we reach software entities that are easy to program. Then, as in manufacturing, we will be able to create applications of any size and complexity by employing inexperienced workers – workers who, individually, can only program small and simple pieces of software.

This idea, known as *software engineering*, is behind every programming theory of the last forty years. But the idea is wrong. We already saw that software applications are in fact *systems* of hierarchical structures, so the structure of modules that appears to represent an application is merely *one* of the structures that make it up. The software entities that constitute the application possess many attributes: they call subroutines, use database fields, reflect business practices, etc. Since each attribute gives rise to a structure, each structure represents a different aspect of the application: one subroutine and its calls, the uses of one database field, the implementation of one business practice, etc. But because they share their elements (the software entities that constitute the application), these structures are not independent. So the only way to develop applications is by dealing with several structures at the same time – something that only minds can do, and only after much practice.

Thus, while software engineering is said to turn programmers from old-fashioned artisans into modern professionals, its true purpose is the exact opposite: to eliminate the need for programming expertise. And this, the elites believe, can be accomplished by discovering scientific (i.e., mechanistic) programming theories, and by restricting programmers to methodologies and development systems based on these theories. The aim is to separate applications into their constituent structures, and further separate these structures into their constituent elements, at which point programmers will only need to deal with small, isolated software entities. For example, the theory of structured programming claims that the only important structure is the one that represents the application's flow of execution, and that this structure can be reduced to some simple, standard constructs; and the theory of object-oriented programming claims that we can treat each aspect of our affairs as a separate structure, which can then be assembled from some smaller, existing structures.

But each theory, while presented as a revolution in programming concepts, is in reality very similar to the others. This is true because they are all based on the same fallacy; namely, on the assumption that software and programming are mechanistic phenomena, and can be studied with the principles of reductionism and atomism. Ultimately, the naive idea of software engineering is a reflection of the ignorance that the academics and the practitioners suffer from. They remain ignorant because they waste their time with worthless theories: they are forever trying to explain the phenomena of software and programming through the mechanistic myth. It is not an exaggeration to say that, for the last forty years, their main preoccupation has been this absurd search for a way to reduce software to mechanics. The preoccupation is also reflected in their vocabulary: programmers call themselves "engineers," and refer to programming as "building" or "constructing" software.

The programming theories, thus, are mechanistic delusions, because they attempt to represent complex phenomena mechanistically. What is worse, instead of being abandoned when found to be useless, they are turned by their defenders into pseudosciences. Here is how: Since neither the academics nor the practitioners are willing to admit that their latest theory has failed, they continue to praise it even as they struggle against its deficiencies. They deny the endless falsifications, and keep modifying the theory in the hope of making it practical. While described as new features, the modifications serve in fact to mask the falsifications: they reinstate the traditional, *non-mechanistic* programming concepts – precisely those concepts that the theory had attempted to eliminate. In the end, the theory's exact, mechanistic principles are forgotten altogether. Its defenders, though, continue to promote it by invoking the benefits of mechanism. Then, after perpetrating this fraud for a number of years, another mechanistic theory is invented and the same process is repeated.

So the software workers are not the serious professionals they appear to be, but impostors. Whether they are academics who invent mechanistic theories, or software companies that create systems based on these theories, or programmers who rely on these systems, very little of what they do is genuine. They appear to be dealing with important issues, but most of these issues are senseless preoccupations engendered by their mechanistic delusions: since our problems rarely have simple, mechanistic answers, there is no limit to the specious activities that one can contrive when attempting to solve them mechanistically.

The mechanistic software ideology, thus, is the perfect medium for incompetents and charlatans, as it permits them to engage in modern, glamorous, and profitable activities while doing almost nothing useful. The software practitioners have become a powerful bureaucracy, exploiting society while appearing to serve it. Less than 10 percent (and often less than 1 percent) of their work has any value. Their main objective is not to help us solve our problems through software, but on the contrary, to create new, software-related problems; in other words, to make all human activities as complicated and inefficient as they have made their own, programming activities.

At the top of this bureaucracy are the software elites – the universities and the software companies. It is these elites that control, ultimately, our software-related affairs. And they do it by promoting mechanistic software concepts: since we believe in mechanism, and since their theories and systems are founded on mechanistic principles, we readily accept their elitist position. But if software mechanism is generally useless, their theories and systems are fraudulent, and their elitist position is unwarranted.

3

Three ingredients are needed to implement totalitarianism: a myth, an elite, and a bureaucracy. And the spread of totalitarianism is caused by an expansion of the bureaucracy: larger and larger portions of the population change from their role as citizens, or workers, to the role of bureaucrats; that is, from individuals who perform useful tasks to individuals whose chief responsibility is to practise the myth.

A characteristic of totalitarianism, thus, is this continuous increase in the number of people whose beliefs and acts are a reflection of the myth. Rather than relying on common sense, or logic, or some personal or professional values, people justify their activities by invoking the myth. Or, they justify them by pointing to certain ideas or theories, or to other activities; but if these in their turn can only be justified by invoking the myth, the original activities are specious.

A totalitarian bureaucracy can be seen as a pyramid that expands downward, at its base. The elite, which forms its apex, uses the myth to establish the system's ideology and to recruit the first bureaucrats – the first layer of the pyramid. Further layers are then added, and the pyramid becomes increasingly broad and deep, as more and more categories of people cease living a normal life and join the bureaucracy. Thus, as the pyramid expands, fewer and fewer people are left who perform useful activities; and the closer an individual is to the top of the pyramid, the greater the number of senseless, myth-related preoccupations that make up his life.

Since the lower layers support the higher ones, the model of a pyramid also explains how social power is distributed under totalitarianism: each layer exploits the layers that lie below it, and the elite, at the top of the pyramid, exploits the entire bureaucracy. Thus, the closer we get to the top, the more power, influence, and privileges we find. In addition, the bureaucracy as a whole exploits the rest of society – those individuals and institutions that have not yet joined it.

The totalitarian ideal is that *all* people in society join the bureaucracy and restrict themselves to myth-related activities. But this, clearly, cannot happen; for, who would support them all? In the initial stages of the expansion, when enough people are still engaged in useful activities, the elite and the bureaucrats can delude themselves that their ideology is working. As more and more people join the bureaucracy, however, the useful activities decline and the system becomes increasingly inefficient. Eventually, the inefficiency reaches a point where society can no longer function adequately, and collapses. It is

impossible to attain the totalitarian ideal – a bureaucracy that comprises the entire society.

It should be obvious, then, why the software myth can serve as the foundation of a totalitarian ideology. Since the essence of totalitarianism is endless expansion, the ideology must be based on an idea that appeals to every individual in society. And few ideas can match software in this respect. As we will see in chapter 4, software is comparable only to language in its versatility and potency. Thus, even when employed correctly, without falling prey to mechanistic delusions, software can benefit almost anyone. But when perceived as a *mechanistic* concept, its utopian promise becomes irresistible. The promise, we saw, is that software devices can act as substitutes for knowledge, skills, and experience. So, simply by operating a software device, we will be able to perform immediately tasks that would otherwise require special talents, or many years of study and practice. The promise of the software myth, thus, exceeds even the most extravagant promises made by the old political or religious myths. Consequently, an elite can dominate and exploit society through the software myth even more effectively than the political and religious elites did through the other myths, in the past.

The expansion of the software bureaucracy parallels the spread of computers; and even a brief analysis of this expansion (later in this section) will reveal the process whereby various categories of people are turned into bureaucrats. All it takes is a blind belief in the software myth – something that the elite is fostering through propaganda and indoctrination. Then, judged from the perspective of the myth, activities that are in fact illogical, or inefficient, or wasteful, are perceived as important and beneficial; and the incompetents who engage in these activities are perceived as professionals.

Ignorance, therefore, is what makes the belief in a myth, and hence the expansion of a bureaucracy, possible. An individual who took the time to develop expertise in a certain field cannot also develop irrational beliefs in the same field, as that would contradict his personal experience. Thus, in addition to its versatility and potency, it is its novelty that makes software such a good subject for myth. We allowed an elite to assume control of our software-related affairs without first giving ourselves the time to discover what is the true nature of software. And the elite saw software, not as a complex phenomenon, but as a mechanistic one; in other words, not as a phenomenon that demands the full capacity of the mind, but as one that requires only mechanistic thinking.

Because of this delusion, we have remained ignorant: we depend on software while lacking the skills to create and use software intelligently. Instead of

developing software expertise, we wasted the last forty years struggling with the worthless theories and methodologies promoted by the elite. Under these conditions, the emergence of irrational beliefs was inevitable. The software myth, thus, is a consequence of our mechanistic culture and our software ignorance.

4

The first workers to be turned into software bureaucrats were the programmers themselves. From the start, the theorists assumed that programming can be reduced to some simple and repetitive acts, similar to those performed by assembly-line workers in a factory. So, they concluded, programmers do not require lengthy education, training, and practice. If we develop software applications as we build appliances, all that programmers need to know is how to follow certain methods, and how to use certain aids – methods and aids based, like those in manufacturing, on the principles of reductionism and atomism. And to improve their performance later, all we need to do is improve the methods and aids.

Thus, instead of trying to understand the true nature of software and programming, the theorists *assumed* them to be mechanistic phenomena; and the programming profession was founded upon this assumption. Using the mechanistic myth as warrant, programming expertise was redefined as expertise in the use of theories, methodologies, and development aids; in other words, expertise in the use of substitutes for expertise. So what was required of programmers from then on was not programming skills, but merely familiarity with the latest substitutes.

If expertise is the highest level attainable by human minds in a given domain, and incompetence the lowest, programmers were neither expected nor permitted to attain a level much higher than incompetence. And, as society needed more and more software, everyone was convinced that what we needed was more and more of this kind of programmers. The alternative – promoting expertise and professionalism, allowing individuals to develop the highest possible skills – was never considered.

The effects of this ideology can be seen in the large number of software failures: development projects abandoned after spending millions of dollars, critical business needs that remain unfulfilled, applications that are inadequate or unreliable, promises of increased savings or efficiency that do not materialize. Statistics unchanged since the 1970s show that less than 5 percent of programming projects result in adequate applications. What these statistics do not reveal is that even those applications that are adequate when new cannot

be kept up to date (because badly written and badly maintained), so they must be replaced after a few years. The statistics also do not reveal that, with inexperienced programmers, it costs far more than necessary to create even those applications that are successful. And if we remember also the cost of the additional hardware needed to run badly written applications, it is safe to say that, for over forty years, society has been paying in effect one hundred dollars for every dollar's worth of useful software.[1]

The conclusion ought to be that the mechanistic assumption is wrong: programming expertise is not the kind of knowledge that can be replaced with methods or devices, so personal skills and experience remain an important factor. The answer to the software failures is then simply to recognize that, as is the case in other difficult professions, to become a proficient programmer one needs many years of serious education, training, and practice.

In our mechanistic software culture, however, this idea is inadmissible; and someone who suggests it is accused of clinging to old-fashioned values, of resisting science and progress. The only accepted answer to the software failures is that we need, not better programmers, but better theories, methodologies, and development aids. If the previous ones failed, we are told, it is because they did not adhere faithfully enough to the mechanistic ideology; so the next ones must be even more mechanistic. In other words, the only permissible solutions to the problem of programming incompetence are those derived from the mechanistic myth – the same solutions that were tried in the past, and which *cause* in fact the incompetence. No matter how many failures we witness, the mechanistic ideology is never questioned.

The mechanistic software concepts cause incompetence because they are specifically intended as *substitutes* for programming expertise. Thus, it is not surprising that programmers who rely on these substitutes do not advance past the level of novices: they are *expected* to remain at this level.

So the incompetence of programmers, and the astronomic cost of software, are a direct consequence of the mechanistic myth. For the first time, a mechanistic delusion is powerful enough to affect the entire society. Previously, it was only in universities that individuals could pursue a mechanistic fantasy, in the guise of research; and the failure of their projects had little effect on the rest of society. Through software, however, the pursuit of mechanistic fantasies became possible everywhere. Unlike the mechanistic theories in

[1] It must be noted that software expenses, and computing expenses generally, are now usually called "investments." Useless concepts can only be promoted through deception: it is easier to make ignorant decision makers *invest*, than it is to make them *spend*, large amounts of money on dubious products and services. Thus, "investment" has joined the deceptive terms "solution" and "technology" in the promotion of software novelties in lectures, articles, advertising, and conversation.

psychology, sociology, or linguistics, the mechanistic *software* theories are not limited to academic research. Being applicable to business computing, they spread throughout society, and degraded the notions of expertise and responsibility in business just as mechanistic research had degraded these notions in universities. Just as the academics perceive their responsibility to be, not the discovery of useful theories but the pursuit of mechanistic ideas, programmers perceive their responsibility to be, not the creation of useful applications but the use of mechanistic software methods.

Millions of individuals are engaged, thus, not in programming but in the pursuit of mechanistic fantasies. Probably no more than 1 percent of the programming activities in society represent useful work; that is, work benefiting society in the way the work of doctors does. We find ourselves today in this incredible situation because programming is a new profession, without established standards of expertise. We allowed the software elite to persuade us that this profession must be based on mechanistic principles, so the standard of expertise became, simply, expertise in mechanistic software concepts. Had we tried first the alternative – giving programmers the time and opportunity to develop the highest knowledge and skills that human beings can attain in this new profession – we would easily recognize the absurdity of the mechanistic concepts, and the incompetence of those who restrict themselves to such concepts. It is only because we take software mechanism as unquestionable truth that we accept the current programming practices as a normal level of expertise. And if we consider this level normal, it is natural to accept also the resulting cost and the failures.

Also, with so many programmers around, new types of supervisors had to be created: more and more employees were turned into software bureaucrats – project managers, systems analysts, database administrators – to oversee the hordes of programmers who, everyone agreed, could not be trusted to develop applications on their own. Again, no one questioned this logic. If the programmers were deemed incompetent and irresponsible, the answer should have been to improve their training. Instead, it was decided to adopt, for software development, the assembly-line methods used in manufacturing; namely, to treat programmers as unskilled workers, and to develop applications by relying on management expertise rather than programming expertise.

So for every few programmers there was now a manager, and for every few managers a higher manager. But the manufacturing methods are inadequate for programming, because software applications are not neat hierarchical structures of subassemblies. Consequently, turning software development into factory-type work did not solve the problem of programming incompetence. It only increased the software bureaucracy, and hence the cost of software, and the failures. (Sociological studies of the programming profession, conducted

in the 1970s, show that the main goal of corporate management was not so much to improve programming practices, as to repress the programmers' attitudes and expectations. For example, the theory of structured programming was promoted as the means to turn programming into an exact activity, and programmers into skilled professionals, while its true purpose was the opposite: to *deskill* programmers; specifically, to eliminate the need and opportunity for programmers to make important decisions, and to give management complete control over their work.[2])

Finally, as the benefits expected from mechanistic software concepts are not materializing, new types of bureaucrats must be constantly invented as a solution to the incompetence of programmers. Thus, companies have now employees with absurd titles like architect, systems integrator, data analyst, business intelligence analyst, and report developer. While justified by invoking the growing complexity of business computing, and the growing importance of information technology, the task of these new bureaucrats is in reality to do what programmers should be doing; that is, create and maintain business applications. What masks this fact is that, instead of programming, they try to accomplish the same thing through various end-user tools, or by putting together ready-made pieces of software. But the idea that we can create useful applications in this fashion is based on the same delusions as the idea that programming expertise can be replaced with methods and aids. So it only adds to the complexity of business computing, while the real software problems remain unsolved. This is interpreted, though, as a need for even more of the new bureaucrats, in a process that feeds on itself.

❖

A major role in the spread of the software bureaucracy is played by the organizations that *create* the knowledge substitutes – the software companies. These companies form the elite, of course. But in addition to propagating the mechanistic software ideology, they function as employers; and in this capacity, they are turning millions of additional workers into software bureaucrats.

[2] See Philip Kraft, *Programmers and Managers: The Routinization of Computer Programming in the United States* (New York: Springer-Verlag, 1977); see also Joan M. Greenbaum, *In the Name of Efficiency: Management Theory and Shopfloor Practice in Data-Processing Work* (Philadelphia: Temple University Press, 1979). It must be noted, though, that, while ground-breaking and as important today as they were in the 1970s, these studies treat the deskilling of programmers as part of the traditional conflict between management and labour. Their authors were unaware of the fallacies of software mechanism, and that theories like structured programming do not, in fact, work. Thus, the delusion that programmers must be a kind of factory workers – because programming is a kind of manufacturing – constitutes a sociological phenomenon that has yet to be studied.

From just a handful in the 1960s, the software companies have grown in number and in size to become an important part of the economy. And they accomplished this simply by invoking the myth of software mechanism. For, their software and services can be justified only if we accept unquestioningly the mechanistic ideology. Thus, only if we agree that software development is a form of manufacturing will we accept the resulting incompetence, and hence the aids and substitutes supplied by these companies as the answer. Or, putting this in reverse, if we had professional programmers instead of the current practitioners, less than 1 percent of the software supplied by these companies would be needed at all.

What this means is that countless organizations, while operating as legitimate businesses under the banner "technology," are actually engaged in the making and marketing of mechanistic software fantasies. So their employees, no matter how good they may be in these activities – programming, research, management, administration, selling – are not performing work that is truly useful. They belong, therefore, to the software bureaucracy.

The programmers who work for these companies hold a special place in the bureaucracy. They are, in general, better prepared and more experienced than the application programmers. But if their job is to develop the useless systems sold by the software companies, their talents are wasted. These systems may appear impressive to their users, but they cannot replace good applications, nor the expertise needed to create good applications. So, if these systems cannot be a substitute for expertise, the work of those who create them is just as senseless as the work of those who use them. We are witnessing, therefore, this absurd situation: our better programmers are employed to create, not the custom applications that society needs, but some generic applications, or some substitutes for the knowledge required to create custom applications. Instead of helping to eradicate the software bureaucracy, our universities prepare programmers for the software companies, thereby *adding* to the bureaucracy. For, by catering to the needs of software bureaucrats, the system programmers are reduced to bureaucrats themselves.

A different kind of software companies are the enterprises run by the individuals known as industry experts, or gurus. Unlike the regular software companies, the gurus earn their fame personally – as theorists, lecturers, and writers. Their role, however, is similar: to promote the ideology of software mechanism. So they are part of the elite. Also like the software companies, their existence is predicated on widespread programming incompetence and an ever-growing bureaucracy.

Although they seldom have any real programming experience (that is, personally creating and maintaining serious business applications), the gurus confidently write papers and books on programming, publish newsletters, invent theories and methodologies, lecture, teach courses, and provide consulting services. Their popularity – the fact that programmers, analysts, and managers seek their advice – demonstrates, thus, the ignorance that pervades the world of programming. To appreciate the absurdity of this situation, imagine a similar situation in medicine: individuals known to have no medical training, and who never performed any surgery, would write and lecture on operating procedures; and real surgeons, from real hospitals, would read their books, attend their courses, and follow their methods.

While unthinkable in other professions, we accept this situation as a logical part of our *programming* culture. The reason it seems logical is that it can be justified by pointing to the software myth: if what we perceive as programming expertise is familiarity with theories, methodologies, and software devices, it is only natural to respect, and to seek the advice of, those who know the most in this area. So the gurus are popular because they always promote the latest programming fads – which, at any given time, are what ignorant practitioners believe to be the cure for their current difficulties.

Programming was only the first profession to be destroyed by the software myth. Once we agreed to treat programmers as mere bureaucrats, instead of insisting that they become proficient and responsible workers, the spread of the software bureaucracy was inevitable. Every aspect of the degradation that is currently occurring in other professions can be traced to the incompetence of programmers. For, as we increasingly depend on computers and need more and more software applications, if the programmers are unreliable we must find other means to develop these applications. We already saw how new types of managers, and new types of software workers, were invented to deal with the problem of programming incompetence. This did not help, however. So the problem spread beyond the data-processing departments, and is now affecting the activities of software *users*.

Little by little, to help users perform the work that should have been performed by programmers, various software aids have been introduced. They vary from simple programming environments derived from databases or spreadsheets (which promise users the power to implement their own applications) to ready-made applications (which promise users the power to eliminate programming altogether). These aids, however, are grounded on the same mechanistic principles as the development aids offered to programmers,

so they suffer from the same fallacies. If the substitutes for expertise cannot help programmers, we can hardly expect them to help amateurs, to create useful applications.

Workers everywhere, thus, are spending more and more of their time doing what only programmers had been doing before: pursuing mechanistic software fantasies. Increasingly, those who depend on computers must modify the way they work so as to fit within the mechanistic software ideology: they must depend on the inferior applications developed by inexperienced programmers, or on the childish applications they are developing themselves, or on the generic, inadequate applications supplied by software companies. The world of business is being degraded to match the world of programming: other workers are becoming as inefficient in their occupations as programmers are in theirs; like the programmers, they are wasting more and more of their time dealing with specious, software-related problems.

But we perceive this as a normal state of affairs, as an inevitable evolution of office work and business management. Because we believe that the only way to benefit from software is through the mechanistic ideology, we are now happy to adopt this ideology in our own work. As software users, we forget that the very reason we are preoccupied with software problems instead of our real problems is the incompetence and inefficiency caused by the mechanistic ideology in programming. So, by adopting the same ideology, we end up replicating the incompetence and inefficiency in other types of work. In other words, we become software bureaucrats ourselves.

Thus, because we do not have a true programming profession, workers with no knowledge of programming, or computers, or engineering, or science are increasingly involved in the design and creation of software applications. And, lacking the necessary skills, they are turning to the knowledge substitutes offered by the software companies – which substitutes address now all people, not just programmers. So, as millions of amateurs are joining the millions of inexperienced practitioners, the field of application development is becoming very similar to the field of consumer goods. A vast network of distribution and retail was set up to serve these software consumers, and a comprehensive system of public relations, marketing, and advertising has emerged to promote the knowledge substitutes: books, periodicals, brochures, catalogues, newsletters, trade shows, conventions, courses, seminars, and online sources.

The similarity to consumer goods is clearly seen in the editorial and advertising styles: childish publication covers; abundance of inane terms like "powerful," "easily," "solution," and "technology"; the use of testimonials to

demonstrate the benefits of a product; prices like $99.99; and so on. Thus, while discussing programming, business, efficiency, or productivity, the promotion of the software devices resembles the promotion of cosmetics, fitness gadgets, or money-making schemes. Also similar to consumer advertising are the deceptive claims; in particular, promising ignorant people the ability to perform a difficult task simply by buying something. The software market, thus, is now about the same as the traditional consumer market: charlatans selling useless things to dupes.

Again, to appreciate the absurdity of this situation, all we have to do is compare the field of programming with a field like medicine. There is no equivalent, in medicine, of this transformation of a difficult profession into a consumer market. We don't find any advertisers or retailers offering knowledge substitutes to lay people and inexperienced practitioners who are asked to replace the professionals.

This transformation, then, has forced countless additional workers to join the software bureaucracy. For, if what they help to sell is based on the idea that software devices can replace expertise, and if this idea stems from the belief in software mechanism, all those involved in marketing the knowledge substitutes are engaged in senseless activities.

❖

Finally, let us recall that it is precisely those institutions which ought to encourage rationality – our universities – that beget the software delusions. Because they teach and promote only *mechanistic* software concepts, the universities are, ultimately, responsible for the widespread programming incompetence and the resulting corruption.

In the same category are the many associations and institutes that represent the world of programming. The ACM and the IEEE Computer Society, in particular – the oldest and most important – are not at all the scientific and educational organizations they appear to be. For, while promoting professionalism in the use of computers, and excellence in programming, their idea of professionalism and excellence is simply adherence to the mechanistic ideology. Thus, because they advocate the same concepts as the universities and the software companies, these organizations serve the interests of the elite, not society.

If this sounds improbable, consider their record: They praise every programming novelty, without seriously verifying it or confirming its usefulness. At any given time, they proselytize the latest programming "revolution," urging practitioners to join it: being familiar with the current software concepts, they tell us, is essential for advancement. In particular, they endorsed the three

pseudoscientific theories we examine in chapter 7, and conferred awards on scientists who upheld them. As we will see, not only are these theories fallacious and worthless, but the scientists used dishonest means to defend them; for example, they claimed that the theories benefit from the rigour and precision of mathematics, while this is easily shown to be untrue. Thus, instead of exposing the software frauds, the ACM and the IEEE Computer Society help to propagate them.

What these organizations are saying, then, is exactly what every software guru and every software company is saying. So, if they promote the same values as the *commercial* enterprises, they are not responsible organizations. Like the universities, their aim is not science and education, but propaganda and indoctrination. They may be sincere when using terms like "professionalism" and "expertise," but if they equate these terms with software mechanism, what they do in reality is turn programmers into bureaucrats, and help the elite to exploit society.

5

The foregoing analysis has shown that our mechanistic software culture is indeed a social phenomenon that is causing the spread of a bureaucracy, and hence the spread of totalitarianism. Every one of the activities we analyzed can be justified only through the software myth – or through another activity, which in its turn can be justified only through the myth or through another activity, and so on. The programmers, the managers, the academics, the gurus, the publishers, the advertisers, the retailers, the employees of software companies, and increasingly every computer user – their software-related activities seem logical only if we blindly accept the myth. As soon as we question the myth, we recognize these activities as what they actually are: the pursuit of mechanistic fantasies.

So the expansion of software-related activities that we are witnessing is not the expansion of some useful preoccupations, but the expansion of delusions. It is not a process of collective progress in a new field of knowledge – what our software-related affairs *should* have been – but a process of degradation: more and more people are shifting their attention from their former, serious concerns, to some senseless pursuits.

In chapter 8 we will study the link between the mechanistic ideology and the notion of individual responsibility; and we will see that a mechanistic culture leads inevitably to a society where people are no longer considered responsible for their acts. The road from mechanism to irresponsibility is short. The belief in mechanism tempts us to neglect the natural capabilities of our minds, and

to rely instead on inferior substitutes: rather than acquiring knowledge, we acquire devices that promise to replace the need for knowledge. We accomplish by means of devices less than we could with our own minds, and the devices may even be wrong or harmful, but no one blames us. Our responsibility, everyone agrees, is limited to knowing how to operate the devices.

Today, the incompetence and irresponsibility are obvious in our software-related activities, because these activities are dominated by mechanistic beliefs. But if we continue to embrace software mechanism, we should expect the incompetence and irresponsibility to spread to other fields of knowledge, and to other professions, as our dependence on computers is growing.

A society where all activities are as inefficient as are our software-related activities cannot actually exist. We can afford perhaps to have a few million people engaged in mechanistic fantasies, in the same way that we can afford to have an entertainment industry, and to spend a portion of our time with idle amusements. But we cannot, all of us, devote ourselves to the pursuit of fantasies. Thus, if the spread of software mechanism is causing an ever-growing number of people to cease performing useful work and to pursue fantasies instead, it is safe to predict that, at some point in the future, our society will collapse.

To avert this, we must learn all we can from the past: we must study the harm that has *already* been caused by software mechanism, in the domain of programming. In programming we have been trying for forty years to find substitutes for expertise, so we have enough evidence to *demonstrate* the absurdity of this idea, and the dishonesty of those who advocate it.

Despite its failure in programming, it is the same idea – replacing minds with software – that is now being promoted in other domains. And it is the same myth, software mechanism, that is invoked as justification, and the same elites that are perpetrating the fraud. So, in a few years, we should expect to see in other domains the same corruption we see today in programming, the same incompetence and irresponsibility. One by one, all workers will be reduced, as programmers have been, to software bureaucrats. As it has been in programming, the notion of expertise will be redefined everywhere to mean expertise in the use of substitutes for expertise. As programmers are today, we will all be restricted to the methods and devices supplied by an elite, and prevented from developing our minds.

Thus, if we understand how the mechanistic delusions have caused the incompetence and irresponsibility found today in the domain of programming, we will be able perhaps to prevent the spread of these delusions, and the resulting corruption, in other domains.

Anthropology and Software

If the theories of software engineering are founded on a myth, it is not surprising that they do not work. The software practitioners, though, continue to believe in software mechanism, and this prevents them from gaining knowledge and experience. Thus, because of their ignorance, the world of programming resembles a primitive society. Also, as other professions increasingly depend on computers, and hence on the mechanistic software myth, the *users* of software are now prevented from gaining knowledge and experience. So the whole world resembles, increasingly, a primitive society. We can learn a great deal about our software delusions, therefore, by comparing the attitudes of programmers and users with those of the primitives.

Let us turn, then, to the field of social anthropology. In the first subsection, we will study the practice of magic as a complement to proven knowledge. And in the second subsection, we will study the invocation of supernatural powers in general.

Software Magic

1

When analyzing the names of software products,[1] we cannot help noticing the large number of names that evoke magic practices. For example, a popular database management system is called Oracle, a word meaning "prophet" and "prophecy" in antiquity. An application development system is called Delphi, after the location of a temple in ancient Greece where oracles were issued. A network system is called Pathworks; pathworking is a form of group visualization practised by those who believe in the occult. One utility is called Genie Backup Manager; others are called Clipboard Genie and Startup Genie. We also have Install Wizard, Disk Clean Wizard, Search Wizard, Web Wizard, PC Wizard, Registry Wizard, Barcode Wizard, etc. To back up drivers we could use Driver Magician, and to create help files Help Magician. A catalogue of hardware and software products describes certain entries as "magic solutions," and offers discounts on other entries to help us "get more magic for less."[2]

[1] As we will see later, the belief that software is a kind of product is one of the fallacies of the software myth. So I use the term "software product" only when I want to stress the absurdity of this concept (as in the present section).

[2] IBM *RS/6000* catalogue (spring 2000), pp. 8, 2.

But to leave no doubt as to the supernatural qualities of their products, many software companies include the word "magic" in the product's name: Network Magic, CADmagic, Barcode Magic, Label Magic, vCard Magic, Brochure Magic, Magic eContact, Image Gallery Magic, Magic Transfer, QS Flash Magic Menu Builder, Screenshot Magic, Magic Styles, Web Design Magic, SCP Button Magic, Magic Internet Kit, Magic Recovery Professional, MagicTracer, Magic Xchange, Macro Magic, AttributeMagic Pro, Color Magic Deluxe, Magic Photo Editor, Magic Speed, Magic Separator, Magic/400, Clipboard Magic, Magic Flash Decompiler, Order Page Magic, MagicWeb, MagicFlare, Magic Window Hider, ZipMagic, Magic TSR Toolkit, Antechinus Draw Magic, Slideshow Magic, Magic Folders, Magic Connection, Magic Mail Monitor, Magic ASCII Studio, Raxso Drive Magic, Magic Writer, File Magic, Magic Blog, Magic Cap, Magic Inventory Management, Magic Calendar Maker, Developer Magic, Magic Link, Magic C++, Spectramagic NX, Magic Net Trace, Exposure Magic, Magic Audio Recorder, MAGic, Word Magic, Voice Magic, Focus Magic, Magic ScreenSaver, Magic Memory Optimizer, Monitor Magic, Pad Magic, PartitionMagic, ClipMagic, SupportMagic, Magic DVD Copier, Backup Magic, SpeechMagic, Video Edit Magic, MagicISO, etc.

Or, software companies adopt the word "magic" for their own name: Computer Magic Inc., InfoMagic Ltd., General Magic Inc., Magic Multimedia Inc., Design Magic Ltd., PC-Magic Software, NeoMagic Corp., Inmagic Inc., Software Magic Inc., Magic Software Enterprises Ltd., Magic Solutions Ltd., PlanMagic Corp., WebMagic Inc., TeleMagic Inc., Imagic Inc., Viewmagic Inc., Geomagic Inc., etc.

In an industry famous for its preoccupation with the latest technological advances, at a time when all we hear is proclamations about progress and the future, one would expect vendors to take special care in *avoiding* terms associated with primitive beliefs, as these associations could hurt their credibility. The opposite is the case, however: the ignorance that pervades the world of software has created an environment where primitive beliefs are again an important factor, so the software vendors *deliberately* employ terms that evoke magic powers.

To those who lack knowledge, the world appears as a mysterious place, full of uncertainties and unexplained events. Superstitions and magic systems are then an effective way of coping with situations that would otherwise cause great anxiety. Irrational beliefs, held by most people in a repressed form even in our modern world, can become dominant and can easily be exploited when ignorance renders rational thinking impossible. And so it is how our society, which is increasingly dominated by software and hence by ignorant software practitioners and users, increasingly resembles the ancient and primitive societies, where priests, magicians, shamans, and prophets were consulted in

all important affairs. Far from *avoiding* associations with supernatural forces, software vendors and gurus – today's priests and prophets – know that for ignorant programmers and users it is precisely these associations that matter.

❖

Magic – a pseudoscience – claims that certain objects, spells, or acts have the power to influence persons and events, although this power cannot be explained. Magic theories appear to provide important benefits, but persons who believe in magic must accept these theories without proof. For this reason, magic beliefs tend to manifest themselves as wishful thinking. Magic systems have existed as long as human societies, so they have always reflected our current preoccupations, fears, and desires. Thus, we have had magic systems to help us win battles, attract mates, predict the future, lose weight, and create software applications without programming.

The person who believes in magic refuses to face reality: he clings to his beliefs and disregards all evidence of their falsity. The validity of most magic theories can easily be determined – by carefully monitoring the successes and failures, for example. But the believer never bothers with such details, and is annoyed when someone suggests it. He already *knows* that the theory works. He enthusiastically accepts any success as verification of the theory, while dismissing major failures as insignificant exceptions.

The problem with magic thinking, then, is not so much one of ignorance as one of method. Even when we are ignorant, logical methods of inquiry enable us to test hypotheses, and hence to adopt only those theories that work. We favour theories that promise simple solutions to difficult problems, naturally; but it is precisely these theories that are most likely to be false. The most important advantage we have over primitive societies is not our scientific and technological knowledge, but our logical methods of inquiry. Our capabilities, which had grown only slowly throughout the centuries, have been growing exponentially since we adopted these methods. Those content to invoke specious explanations when reaching the limits of their understanding, instead of seeking to expand their knowledge, are condemned to intellectual stagnation. Their knowledge grows very slowly, or not at all.

Given the success that science had in explaining nature and extending our knowledge, it is not surprising that, until recently, magic practices were considered to be a vestige of our primitive past. All human societies, it was believed, start with magic, and when sufficiently advanced, replace it with science. No society can possibly continue to practise magic once the benefits of scientific thinking are revealed to it. Magic thinking, it was thought, is simply prescientific thinking.

Like the theory of myth, however, the theory of magic has undergone a dramatic shift in the last one hundred years. Far from being a vestige of the past, far from being automatically displaced by science, we understand now that magic beliefs affect a modern society just as much as they do a primitive one. All that has happened is a change in theories. We may no longer believe that weather rituals can bring rain, but we accept many other theories – in economics, linguistics, psychology, sociology, programming – which are, in fact, as scientific as rain magic.

Our reevaluation of the role of magic in society started following the work of anthropologist Bronislaw Malinowski.[3] Malinowski, who studied in great detail the life of primitive peoples, was struck by the continual blending of magic thinking and rational thinking. To a casual observer, the primitives appear to merely add some spurious ceremonies to all their activities. Careful study, however, reveals a surprisingly logical pattern. Magic is not practised at will. For each activity, tradition dictates whether magic is required at all, which magic formula must be used, at what point it should be applied, and which magician is qualified to perform the ritual. The ritual, which may be quite lengthy and elaborate, must be performed with great precision, since any deviation from the formula is believed to weaken its efficacy.

The pattern Malinowski observed is this: when the activity can be performed with confidence, when the primitives expect a certain and easy success, no magic is employed; but when the activity entails a significant degree of uncertainty or danger, magic is deemed necessary. Also, just as one would expect, the greater the uncertainty or danger, the more elaborate the magic employed. This is how Malinowski puts it: "We find magic wherever the elements of chance and accident, and the emotional play between hope and fear have a wide and extensive range. We do not find magic wherever the pursuit is certain, reliable, and well under the control of rational methods and technological processes. Further, we find magic where the element of danger is conspicuous. We do not find it wherever absolute safety eliminates any elements of foreboding."[4]

Primitive people employ magic, then, as an *extension* to their knowledge and capabilities. When they feel that skills and labour alone will allow them to complete a given task, their actions are totally rational. But when they know from experience that despite their skills and labour they may still fail, they resort to magic. This happens in activities like agriculture, hunting, and fishing, which depend on factors that are unpredictable and beyond their

[3] See, especially, his *Coral Gardens and Their Magic* (New York: Dover, 1978), and *Argonauts of the Western Pacific* (New York: Dutton, 1961).

[4] Bronislaw Malinowski, *Magic, Science and Religion, and Other Essays* (Garden City, NY: Doubleday Anchor, 1954), pp. 139–140.

control. They also use magic to complement their rational efforts in matters like health or social relations, which also contain much uncertainty.

2

Programming and software use are saturated with magic practices, but we fail to notice this fact. The reason we fail to notice it is the uncanny similarity between magic practices and rational behaviour: "Magic is akin to science in that it always has a definite aim intimately associated with human instincts, needs, and pursuits. The magic art is directed towards the attainment of practical aims. Like the other arts and crafts, it is also governed by a theory, by a system of principles which dictate the manner in which the act has to be performed in order to be effective."[5]

If we watch the activity of a person while being unfamiliar with the scientific principles underlying that activity, we cannot distinguish between rational and magic practices. Only if our knowledge *exceeds* his, can we recognize which acts contribute to his success and which ones are spurious. Primitive people, when engaged in pursuits like agriculture, feel that technical knowledge and magic rituals are equally important. We, watching them from our position in an advanced society, can recognize that only their technical knowledge contributes to their success, and that their rituals are spurious. At the same time, we ourselves engage in spurious activities in our *software* pursuits, convinced that they are as important as our technical expertise. Thus, only a person with superior programming knowledge can recognize the absurdity of such concepts as structured programming and object-oriented programming.

So it is the similarity of our rational and our irrational acts that we must study if we want to uncover the absurdities in today's software practices. But how can we study this similarity? We are convinced that everything we do is rational – we never perform foolish acts deliberately – so we will always fail to distinguish between the rational and the irrational in our own life. One way, we will see later in this book, is to approach any software concept, product, or theory with due skepticism. As in other disciplines, we can apply logical methods of inquiry to confirm or refute any software claim. Besides, as these methods are universal, they can be used even by those with limited programming knowledge. And when doing this, we discover that most software claims are associated with pseudoscientific theories, propaganda, and charlatanism.

Another way is to study the blending of the rational with the irrational in the lives of primitive people, which, in turn, will help us to recognize the same

[5] Ibid., p. 86.

conduct in our own life. For this purpose, we can find no better examples than the garden and canoe magic systems used in the Trobriand islands of eastern New Guinea, which were so thoroughly documented by Malinowski.

❖

The natives display great agricultural expertise in tending their plantations. They understand, for instance, the properties of the different types of soil, and they know which crops are best suited for each type; they are familiar with the principles of fertilization; and they can identify hundreds of varieties and types of plants. In addition, they are conscientious workers, and they perform skilfully such tasks as preparing the garden, planting the seeds, protecting the growing crops, and harvesting them.

This expertise, however, is always supplemented with magic. The natives can explain, for example, why no crops can thrive in certain areas of their island "in perfectly reasonable, almost scientific language.… At the same time they attribute the supreme fertility of some districts … to the superiority of one magical system over another."[6] They devise clever ways to protect their crops from pests, and "these practical devices they handle rationally and according to sound empirical rules."[7] At the same time, they build and deploy various structures and objects in their gardens, which, they clearly explain, have no other purpose but magic.

The natives do not use magic because they confuse it with practical work. They realize that invoking magic powers is an entirely different type of act, but they believe it to be just as important: "The two ways, the way of magic and the way of garden work … are inseparable. They are never confused, nor is one of them ever allowed to supersede the other."[8] The natives know which tasks they must perform through their own skills and work, and they never attempt to use magic as a substitute. Thus, they "will never try to clean the soil by magic, to erect a fence or yam support by a rite.… They also know that no work can be skimped without danger to the crops, nor do they ever assume that by an overdose of magic you can make good any deficiencies in work.… Moreover, they are able to express this knowledge clearly and to formulate it in a number of principles and causal relations."[9]

Malinowski includes two diagrams showing stages in the growth of one of the local crops, drawn from information provided by the natives themselves.[10] It seems that the natives have greater knowledge about their crops than some

[6] Bronislaw Malinowski, *Coral Gardens and Their Magic*, vol. 1 (New York: Dover, 1978), p. 75. [7] Ibid., p. 77. [8] Ibid., p. 76. [9] Ibid.
[10] Ibid., pp. 140–141.

modern farmers have about theirs. They can describe in great detail the entire development process, from the time the seed is placed in the ground until the plant matures. There are more than twenty native terms in these diagrams – for various parts of the seed, roots, branches, etc. – showing their keen interest in the botanic aspects of their work.

At the same time, the natives have elaborate systems of magic, which they apply scrupulously throughout the growth process. The magic varies from specialized spells and charms addressing individual parts of the plant, to rituals for their tools and for the whole garden. Most of this magic is performed by professional magicians, who receive fees for their services. There are several magic systems in use, and the natives discuss their relative merits with the same seriousness as programmers discussing their application development systems. Some magic systems are owned by individuals, families, or clans, and in this case others must pay for their use – a practice not unlike our patents and copyrights.

We discover a similar combination of rational and irrational acts in canoe building and the associated fishing and trading activities.[11] The natives build sturdy and attractive craft, their size and design matching their intended use: a simple type for coastal transport, a more elaborate type for fishing, and a relatively large and complex type, carrying more than a dozen men, for long sea voyages. Limited to primitive tools, the building of a dugout canoe is a major construction project for them, demanding coordinated team work and timely contribution from specialists. But they are capable of accurate planning and efficient labour organization. Also, they are familiar with the principles of buoyancy and stability, sailing and navigation. They understand, for example, why the outrigger must have a certain, optimal span, measured as a fraction of the canoe's length: a larger span offers greater stability, but at the same time it weakens the outrigger. And they can explain clearly why one canoe is faster than another, or why, in a storm, they must follow one procedure rather than another. "They have," Malinowski points out, "a whole system of principles of sailing, embodied in a complex and rich terminology, traditionally handed on and obeyed as rationally and consistently as is modern science by modern sailors."[12]

Despite these skills, however, every stage in the building of the canoe is accompanied by a magic ritual, deemed necessary to ensure a fast and safe craft. To pick just one example – which also demonstrates the importance of details in magic – a ritual performed before painting the canoe involves burning under its bottom a mixture of such substances as the wings of a bat,

[11] Bronislaw Malinowski, *Argonauts of the Western Pacific* (New York: Dutton, 1961), esp. chs. IV–VI. [12] Malinowski, *Magic, Science and Religion*, p. 30.

the nest of a small bird, cotton fluff, and grass. "The smoke is supposed to exercise a speed-giving and cleansing influence.… All the substances are associated with flying and lightness. The wood used for kindling the fire is that of the light-timbered mimosa tree. The twigs have to be obtained by throwing at the tree a piece of wood (never a stone), and when the broken-off twig falls, it must be caught by hand, and not allowed to touch the ground."[13] Malinowski describes dozens of additional rites, spells, and ritual performances.

❖

What are we to make of this? How is it possible for people to be so rational, and yet so irrational, at the same time? To answer this, we must start by noting that people appear irrational only when judged from *outside* their system of belief. Judged from *within* that system, their conduct is logical and consistent. All it takes is one unproven concept, one false assumption. An entire system can then be built around it, and even if every theory and method in the system is logically derived, that one assumption will render the system nonsensical.

In the case of magic, the false assumption is that certain objects, spells, and ritual performances have the power to influence people's lives, or the forces of nature, or the course of events. In the case of programming, the false assumption is that software applications are akin to the appliances we build in a factory, so programming is akin to manufacturing; that, like appliances, we can separate an application into independent modules, each module into simpler ones, and so on, down to some small parts; that all we need to know is how to program these small parts, because there exist methods and devices which allow us to build applications from software parts just as we build appliances from physical parts; and that, moreover, we can complete our software manufacturing projects even faster if we start with prefabricated subassemblies – large modules that already contain many parts.

In programming as in magic, many principles and methods have been invented, and organized into logical systems. There isn't much that can be criticized when studying such a system from within itself; that is, when using as criteria of validity only concepts that are part of the system. This is what believers are doing, and why the system appears sound to them.

Thus, an individual who believes in magic will always use magic systems; then, *within a magic system*, his conduct will always be logical. Similarly, theorists and practitioners who assume that programming is similar to manufacturing will always pursue *mechanistic* software ideas; then, *within the mechanistic ideology*, their decisions and acts will always be logical.

[13] Malinowski, *Argonauts*, p. 140.

But the validity of each part of the system depends ultimately on the validity of that one fundamental assumption, which may well be the only concept linking the system to the real world. If that concept is wrong, the entire system, no matter how logical, becomes worthless. Believers never question that concept. The larger the system they build around it, the smaller and less important the concept appears to be. Eventually, they forget altogether that the concept was never anything but an assumption.

3

We are now in a position to explain the blending of rational and irrational behaviour. Primitive societies are closed societies. Their members follow elaborate traditions – rigid patterns of thought and conduct – in all their activities. The traditions derive from ancient myths, which are the charter and the foundation of their culture.

Among other things, tradition establishes for each activity what is within the power of the individual and what is beyond his power. For the part that is within his power, the individual is expected to act rationally and to display expertise, initiative, and creativity. But what is he expected to do when something is believed to lie beyond his power? Recall Malinowski's critical observation that magic is employed only when the outcome of an activity has a great degree of uncertainty, when the primitives know that their skills alone cannot ensure success. Because their social system does not permit them to acquire skills beyond the boundaries determined by tradition, it must provide them with other means to cope with the more difficult tasks. This is the purpose of magic. Simply by accepting one unproven theory, they gain access to a multitude of new possibilities.

If we divide the world of primitive people into fields they understand and control, and fields that lie beyond their knowledge and capabilities, what magic does is bring the latter into the same category as the former. Magic assures them that the methods they use successfully in those fields they understand can be used equally in fields where their knowledge is inadequate.

The primitives know perfectly well when it is *skills* that they rely on and when it is *magic*. When sailing, for example, if the wind suddenly changes they use a spell to persuade it to return to its original direction. We, with our knowledge and computers, are content to try to *predict* the weather; through magic, however, the primitives believe they can *control* it. But their behaviour is quite logical: they make use of their sailing methods as long as they work, and turn to magic precisely because they realize that adjusting their sails would be ineffective, that it is the wind they must now adjust rather than the sails.

Instructing the wind to change direction appears silly only if we reject the theory that the weather can be controlled. They *accept* this theory; so they apply methods that involve the weather, in the same way they apply methods that involve the sails. Both types of methods appear to them equally rational and effective. *Magic practice is an attempt to use our current capabilities to accomplish tasks that require, in fact, greater capabilities.*

It is important to remember that magic does not ask us to accept a *different* mistaken theory every time. All magic practices are based on *the same* mistaken theory. Besides, this theory is plausible: all it asks us to believe is that we can influence events by means of spells or objects. Magic, thus, makes processes that are impossible appear like a logical extension of processes that are familiar and effective. After all, we do influence the world around us with spoken words, with our bodies, with objects and tools. This is why it is so easy for us to believe in magic, and so difficult to distinguish between our magic activities and our rational ones. We may think that we are performing the same kind of acts, but these acts can have a real and verifiable effect one moment and an illusory effect the next.

❖

And the same is true of *software* magic. In chapter 7 we will see that the mechanistic software theories do not promise any benefits that could not be gained simply through good programming. What the software elites are seeking, therefore, is a substitute for programming knowledge: by incorporating various principles into a methodology, or into a development environment, they hope to get inexperienced programmers to accomplish tasks that require, in fact, great expertise. Following rules and methods, or using built-in features and operations, is easier than acquiring knowledge and skills, and is within the capabilities of inexperienced programmers. Programming systems, thus, are perceived as magic systems: they assure programmers that they can accomplish a difficult task with their current knowledge alone.

Software development has become the most elaborate type of magic ever created by man, but this escapes our notice if we watch only superficially the activities of programmers. For, in their activities, as in those of primitive people, the rational and the irrational blend and overlap continually. We already saw that one can distinguish irrationality only by stepping outside the system of belief that fosters it, so we must also do this for software.

Each software activity appears logical, urgently needed, and perfectly justified – *if* studied in the context of other, similar activities. This is because most software activities are engendered by some previous software activities. We may even be impressed by the incessant changes and innovations, the

endless theories, languages, methodologies, and development tools, the thousands of courses, exhibitions, conventions, newspapers, magazines, books, brochures, and newsletters, and the astronomic amounts of money spent by corporations and governments. But if we study these activities, we notice that they only make sense if we accept the unproven theory that software development is akin to manufacturing. This absurd theory has been accepted for so long that it is now routinely invoked as the ideological justification for every software concept, when there is no evidence, much less a scientific foundation, to support it. We saw that with magic, by accepting just one unproven theory, the primitives gain the confidence to handle tasks that lie beyond their capabilities. Similarly, by accepting just one unproven *software* theory, inexperienced programmers can confidently engage in activities that lie beyond *their* capabilities.

Like magic in primitive societies, software magic is quite plausible. After all, we build *physical* structures by assembling standard parts and prefabricated modules, and computer programs appear to have their own kind of parts and modules. We improve our manufacturing methods and tools continually, and programming also appears to involve methods and tools. Moreover, programming methods based on the principles of manufacturing seem to work in simple cases – in the examples found in textbooks, for instance. Thus, extending these methods to the large and complex applications we need in the real world appears to be a logical step, whose validity is guaranteed by the fact that large *manufacturing* projects appear to use the same methods as the small ones; they merely involve more parts and subassemblies.

Also like primitive magic, software magic does not ask us to have faith in a *different* unproven theory for each new concept. All programming methods and systems are based on *the same* theory – the similarity of software development to manufacturing – and this makes its fallaciousness harder to detect. These concepts have become a self-perpetuating belief system: a system that uses its own growth as confirmation of validity. No one seems to remember that the entire system, despite its enormous size and complexity, is based ultimately on a theory that was never proved. (See pp. 511–512.)

Unlike other disciplines, where mechanical analogies may lurk behind a theory but are seldom avowed, the software practitioners are quite outspoken about their attempt to reduce software to mechanics. We *must* make programming like manufacturing, they say. They proudly add mechanical metaphors to their software jargon, and take this as a sign of expertise: we are finally turning software into a professional activity, like engineering. But there is no evidence

that programming can be based on manufacturing methods. So, even if programmers actually had the training and experience of engineers (rather than merely *calling* themselves engineers, and using engineering *metaphors*), these skills alone would be of little benefit.

Their claim to expertise through mechanical metaphors is especially amusing, as the belief in software mechanics makes their activities look less and less like expert programming and increasingly like primitive magic. Malinowski called this verbal pattern "the creative metaphor of magic":[14] "It is the essence of magic that, by the affirmation of a condition which is desired but not yet fulfilled, this condition is brought about."[15] The verbal part of a magic formula is typically an elaborate and picturesque series of statements describing the *desired* state of affairs, which, of course, is very different from reality. The person performing the ritual asks, as it were, the forces of nature, or certain objects, to behave in a different manner, or to possess different qualities: "The repetitive statement of certain words is believed to produce the reality stated.… The essence of verbal magic, then, consists in a statement which is untrue, which stands in direct opposition to the context of reality. But the belief in magic inspires man with the conviction that his untrue statement must become true."[16]

So when programmers call themselves "engineers," when they talk about "software engineering" and "building" programs from software "components," they are practising in effect software magic: they are making statements they know to be untrue (or, at least, know to be unproven), hoping that, through their repeated assertion, software phenomena may be persuaded to be like the phenomena we see in manufacturing.

4

Let us return to the blending of the rational and the irrational in software activities. Programmers act quite rationally when working on small and isolated pieces of an application. They know, for example, the importance of expressing correctly the conditions for an iterative statement, and they don't expect their development tools to do it for them. They never question the need to specify certain operations in the proper sequence, or to assign correct values to variables, or to access the right database records. And if the resulting program does not work as expected, it is their own logic that they suspect, not the computer.

[14] Malinowski, *Coral Gardens*, vol. 2, pp. 70, 238.
[15] Ibid., p. 70.
[16] Ibid., pp. 238–239.

But this is where their rationality ends. We all know that the difficulties encountered in large and complex applications are not simply the accumulation of a large number of small problems. When a software project fails, or when an application does not provide the solution everyone expected, it is not an individual statement or condition that must be corrected, or the subtotals in a report that are wrong, or a data entry field that is missing – nor even a hundred such problems. Isolated deficiencies may well contribute to the failure of the application, but even when we manage to identify and resolve them, the application remains inadequate. The reason is that applications are systems of interacting structures. And the most serious software deficiencies are those caused by the interactions: we overlooked or misjudged some of the links between structures.

Applications, then, are more than the simple hierarchical structures we wish them to be, more than the neat modules and relations we see in diagrams. All programming theories are based on the idea that we must reduce the application to one structure, and thereby *eliminate* the interactions. This is what we do in manufacturing, the theorists say, so this must also be the answer to our programming difficulties. But it is precisely the interactions that make software such a versatile concept: it is the very fact that we can implement interacting structures through software that lets software adapt so well to our needs. The reason we don't seem to be able to eliminate the interactions, no matter what theory we follow, is that we need these interactions if software is to mirror our affairs accurately.

Only minds can process interacting structures, so the answer to our programming difficulties is programming expertise: the skills attained by working for many years on large and complex applications, and on diverse types of software. In our culture, however, programmers are restricted to simple and isolated tasks. Like the members of a primitive society, they are expected to display knowledge and creativity in those activities deemed to be within their power: programming small parts of an application. Hard work may be required, but the success of these activities is assured. Tradition does not permit them to acquire the higher skills needed to design, program, and maintain whole applications. This is a difficult task, full of uncertainties, for which tradition prescribes the use of magic: methodologies, development tools and environments, database systems, and the like. These aids encourage programmers to think of the application as a system of independent structures and parts, thus reassuring them that their current knowledge suffices. Like primitive magic, software magic creates for programmers the illusion that the difficult and unpredictable tasks are of the same kind as the simple ones: the methodology, the development tools, or the database system will somehow turn those independent structures and parts into a useful application.

It takes an experienced person to recognize how little of what programmers do is rational, and how much effort they waste on spurious activities. Neither the programmers themselves nor a lay person watching them can see this, because irrational programming activities are almost identical to rational ones. Thus, a programmer may spend much time mastering the complexities of a particular development system, and even more time later programming in that system, convinced that this is the only way to enhance his capabilities. If asked to demonstrate the benefits of the system, the only thing he can do is point to its popularity, or describe a particular function that was easy to implement. But he cannot *prove* the need for that system. In reality, the most important factor is his skills. Whatever he managed to accomplish with that system he would have accomplished with any other system, or with no system at all (that is, with a traditional programming language, perhaps supplemented with libraries of subroutines). Like the primitives, though, the programmer remains convinced that his technical knowledge and the magic system are equally important.[17]

Since no one can prove the need for a particular development system, all related activities are specious. But there is nothing to betray their irrationality. Studying reference manuals, attending courses, discussing problems and solutions – all these activities are important, all can be justified. They can be justified, however, only in the context of that development system, only if we do not question the need for it.

As a result, even when they get to know a development system well, programmers are no better off than before. Their programming skills did not improve. They wasted their time acquiring worthless knowledge about yet another methodology, yet another language, yet another theory, instead of improving their skills simply by programming. All they did was learn how to use a new magic system.

It is easy to see that, no matter how many years of practice these programmers have behind them, their real programming experience stays at the level it was after the first year or two. They may be familiar with many magic systems, but they have no skills beyond what the software tradition permits them to acquire. Just like the primitives, they do not confuse programming with magic. They know perfectly well what they can accomplish with their own skills, and

[17] The benefits of a system or method can be determined only by way of controlled experiments; that is, experiments designed to isolate and measure a specific variable while eliminating all others, including human factors. Such experiments are practically impossible, and this is one reason why the only meaningful way to determine the value of a system or method is by studying the *failures*, not the successes. (We will discuss this problem in "Popper's Principles of Demarcation" in chapter 3.) Thus, any attempt to defend or promote a concept by pointing to individual successes turns it into a pseudoscience, a fraud.

they turn to magic for the more difficult tasks precisely because they are aware of their limited capabilities.

I have described the rational and irrational activities of programmers, but, increasingly, a similar blend can be seen in the activities of software *users*. They too believe that the only way to improve their performance, or to solve difficult problems, is by relying on software devices. Like the programmers, though, whatever they manage to accomplish is due almost exclusively to their skills, not to those devices. To advance, therefore, they must *avoid* the devices, and practise their profession instead, in order to further improve their skills.

How, then, can we detect irrational activities in our software pursuits? We must beware of those activities that can only be justified if judged from within the software culture. We must not be impressed by how important or urgent these activities seem to be, or how expertly the individual performs them. Instead, we must search for evidence. Any attempt to prove the validity of an irrational act will lead to that unproven theory – the theory that forms the foundation of our software culture. The theory is that there exist systems which help us to break down software-related tasks into smaller and smaller parts, so all we need to know is how to use these systems and how to solve simple problems. This is what we do in manufacturing, and software is no different.

❖

Software propaganda has succeeded in shifting our definition of programming expertise from its traditional, commonsensical meaning – the skills needed to solve a difficult problem, or to complete an important task – to its modern meaning: familiarity with the latest theories and methodologies, avoiding programming and using instead ready-made pieces of software, etc. We are expected to measure the expertise of software practitioners, not by assessing their real contribution, but by how many development tools they have tried, how many courses they have attended, how many computer magazines they are reading, and how acquainted they are with the latest "solutions" and "technologies" – the latest ideas, products, announcements, and rumours.

Companies need programmers, but one wouldn't think so just by reading job offer advertisements. For, the required qualifications we see in these advertisements are not what one would think is expected of programmers; namely, proven expertise in solving a company's business problems with software. Depending on the current fad, the requirements are for experience with object-oriented systems, or 4GL systems, or client-server systems, or relational database systems, or CASE tools, or a particular language or development aid or environment; that is, knowledge of one magic system or another. Companies are looking for magicians, not programmers.

Software Power

1

The term *mana*, which comes from Melanesian, was introduced in anthropology at the end of the nineteenth century by R. H. Codrington. This term, usually translated as *power*, denotes a supernatural force, a mythical essence, "an atmosphere of potency that permeates everything."[1] Since then, it has been found that archaic peoples throughout the world believe in its existence. Although we now refer to this concept as mana, it has equivalent terms in many languages: for some peoples of India it is *sakti* or *barkat*, for the African Pygmies *megbe*, for the Iroquois *orenda*, for the Hurons *oki*, for the Dakota *wakan*, for the Sioux *wakanda*, for the Algonquins *manito*.[2] It is believed that this force exists everywhere in the universe, and that any person can use it to accomplish tasks he would otherwise find impossible. The force is said to derive from a number of sources, such as ghosts, spirits, and gods.

Mana can reveal itself in almost anything: a tree, a stone, an animal, and even in such things as a gesture, a sign, a colour, and a season of the year.[3] A typical use of mana may be as follows:[4] An individual would go alone to some isolated spot, where, after fasting, prayer, and exposure to the elements, a spirit might come and point to him a plant. That plant would then become a source of good luck, and the individual would employ this power to ensure success in his endeavours. He might carry with him at all times something symbolizing the plant, and perhaps also offer it to others.

Mana is different from magic. Mana is a universal force available to anyone at any time, and to be used in any way the individual desires; magic, on the other hand, requires formal practice: its power is in the spell and ritual, and magic formulas have an exact significance. Mana exists in nature and can manifest itself in objects, acts, or ideas; magic power resides in man, and magic formulas can only be transmitted from one person to another.[5] Thus, while primitive man may use both magic and mana, most anthropologists agree that, despite their similarity – the belief in supernatural powers that can enhance a person's limited capabilities – they form two different concepts. Sometimes,

[1] Ernst Cassirer, *Language and Myth* (New York: Dover, 1953), p. 63.

[2] Mircea Eliade, *Myths, Dreams, and Mysteries: The Encounter between Contemporary Faiths and Archaic Realities* (New York: Harper and Row, 1975), ch. VI passim.

[3] Ibid., p. 132.

[4] Guy E. Swanson, *The Birth of the Gods: The Origin of Primitive Beliefs* (Ann Arbor: University of Michigan Press, 1964), p. 7.

[5] Bronislaw Malinowski, *Magic, Science and Religion, and Other Essays* (Garden City, NY: Doubleday Anchor, 1954), p. 77.

mana is taken as the general concept, and magic as one particular application of it. As we will see in this subsection, software practitioners and users, too, consider mana a more general concept than their formal magic systems.

❖

The words "power," "powerful," "empower," etc., are so common in computer-related discourse that it is almost impossible to describe a new product without the use of them. We have come to expect them, and we doubt the efficacy of the product if these words are missing. After all, we already have thousands of software and hardware products, so the only justification for a new one is that it is more "powerful." An analysis of these words, however, reveals that the power of a product is usually perceived, not as certain qualities, but in the sense of mana – as *supernatural* power.

From the many meanings the dictionary offers for the word "power," it is obvious that the one current in computer matters is *the capability to effect something*. We can immediately divide this function into two kinds. First, "power" can simply stand for a list of qualities. For example, if one computer is faster than another, or if one text editor has better editing features than another, we may say that they are more powerful. When used in this sense, "power" is an abbreviation: an abstract term we can employ without fear of confusion, since we all know what it stands for. If asked, we could readily describe the superior features we subsumed under "power."

Even a casual examination of books, articles, advertising, or conversations makes it clear, however, that "power" is hardly ever used in this precise sense. In its more common sense, "power" is still used as an abstract term, but *without being defined*. Abstract terms are so common in everyday discourse that we seldom stop to think whether we know what they stand for. So, when encountering an undefined abstract term, we tend to assume that it stands for the list of things we *expected*, or *wished*, to see at that point. When encountering "power" without an explanation, then, we assume that it means what it would mean if used legitimately, although now it is just a slogan.

Here are some typical uses of "power" and its derivatives in computer-related discourse: "Powerful software solutions for midsize companies."[6] "Discover the power of Primus Internet services."[7] "Empowering the Internet generation."[8] "Empowered with these capabilities, your company can charge ahead intelligently and efficiently ...."[9] "Power tools for power applications."[10] "Powering comprehensive unified communications solutions."[11] "Wireless

[6] http://whitepapers.techrepublic.com.com/.
[7] Primus Canada, adv. pamphlet.
[8] Cisco Systems, adv.
[9] http://www.jda.com/.
[10] Microsoft Visual Basic 2.0, adv. pamphlet.
[11] http://www.myt3.com/.

inventory systems give you the power to have accurate information in real time ….”[12] “Open source empowers the user more than proprietary software can.”[13] “Empowering Software Development Environments by Automatic Software Measurement.”[14] “Business innovation powered by technology.”[15]

When it does not describe precise and verifiable capabilities, “power” is intended to convey something mysterious, supernatural – mana. For the primitives, the belief in mana, like the belief in magic, is a substitute for personal knowledge: “Mana is a substance or essence which gives one the ability to perform tasks or achieve ends otherwise impossible.”[16] Similarly, modern individuals believe that a given product or concept has the power to enhance their capabilities, but they don't feel they have to understand how this power acts.

Now, products of all kinds promise us power – weight-loss gadgets, money-making schemes, self-help instructions, and so forth. But in no other field is the promise of power as widespread as in software-related matters. We can see this not only in the frequent use of “power,” “powerful,” “empower,” etc., but also in the long list of software products whose *name* includes “power” (this use of “power,” needless to say, is always in an undefined sense): PowerEncoder, Power Keeper, PowerCrypt, PowerPoint, PowerGraphs Toolkit, NXPowerLite, PowerShadow, PowerOLAP, Power Booleans, IT PowerPAC, Power Edit, PDF Power Brand, PowerShop ERP, PowerGREP, RoutePower 32, Animation Power, PowerCinema, PowerPassword, PowerPulse, Bill Power, PowerBackup, HTML PowerSpell, PowerExchange, PowerPressed, Power Office, PowerKey Pro, PowerConvert, HedgePower, PowerBuilder, PowerDesk Pro, PowerDraw, Power Translators, PowerDirector, PowerProducer, Power Solids, Power Print, EMail Developer's Power Suite, PowerUpdate, PowerERP, Power Accounting, OptionPower, Power LogOn, Powerpak, PowerPack, PowerGEM, PowerTerm, PowerChain, PowerBSORT, PowerTCP Emulation, PowerSuite, PowerRecon, ELX Power Desktop, PowerTicker, PowerAnalyzer, Power Broker, Jobpower, PowerBASIC, Powershell, PowerWebBuilder, PowerWEB, PowerPlan, ES Power PDF Creator, PowerToys, PowerMerge, PowerCOBOL, PowerCenter, DQpowersuite, PowerPath, PowerVideoMaker, SQL Power Architect, Power Sound Editor, PowerBoot, PowerISO, etc.

We can account for the abundance of “power” names in software products only if we remember the ignorance that software practitioners and users suffer from, the limited skills that our software culture permits them to acquire. Faced with the difficult problem of developing, using, and maintaining serious

[12] http://findmysoftware.com/. [13] http://www.netc.org/.
[14] Book title, 11th IEEE International Software Metrics Symposium.
[15] Front cover banner, *Information Week* (1999–2007).
[16] Swanson, *Birth of the Gods*, p. 6.

applications, modern people, like the primitives, end up seeking aid from the only source they believe to be available – supernatural forces.

Few people, of course, would admit that they are using a software product because its name includes "power." But the software vendors know better. The ability of a product's name to influence a buying decision, and the associations created in a person's mind between the product and the idea conveyed by its name, are well understood in advertising. The software vendors are simply exploiting the belief in the supernatural, which has been retained, in a repressed form, even by modern man. This belief surfaces in moments of insecurity, or anxiety, or fear, when, like our ancestors, we feel impotent against some great perils. Since ignorance is a major source of insecurity, the large number of products with "power" names merely reflects the large number of difficult situations that ignorant programmers and users are facing.

Similarly, the phrase "power tools" is often used by software vendors to name sets of software devices: LG Power Tools, Engineering Power Tools, SQL Power Tools, HTML PowerTools, Windows Powertools, PowerTools PRO for AOL, TBox Power Tools, jv16 Power Tools, Rizone's Power Tools, Creative Element Power Tools, Nemx Power Tools, Power Tools for ArcGIS, Rix2k Extreme Power Tools, CodeSite Power Tools, etc.

The phrase is also popular in book titles: Java Power Tools, Unix Power Tools, Linux Power Tools, Mac OS X Power Tools, DOS Power Tools, Scripting VMware Power Tools, Windows Developer Power Tools, LEGO Software Power Tools, AutoCad Power Tools, Windows XP Power Tools, Netcat Power Tools, Wordperfect 6 Power Tools, Foxpro 2.0 Power Tools, Visual Basic .NET Power Tools, Novell Netware Power Tools, etc.

The vendors, clearly, want us to associate a software utility, or the information found in a book, with the efficacy of electricity; that is, with the kind of energy used by real power tools like drills and saws. But, without an actual explanation, the meaning of this "power" remains vague, just like the "power" in a name. So, in the end, we perceive it the same way – as mana.

2

Much has been learned about the way the primitives interpret mana, from linguistic and ethnological analyses of the archaic languages. The conclusion has been that "mana" is not simply a word, like "power." We must use a multitude of concepts to convey in a modern language its full meaning: "sacred, strange, important, marvellous, extraordinary";[17] also "remarkable,

[17] Paul Radin, quoted in Eliade, *Myths, Dreams, and Mysteries*, p. 129.

very strong, very great, very old, strong in magic, wise in magic, supernatural, divine – or in a substantive sense … power, magic, sorcery, fortune, success, godhead, delight."[18]

Cassirer notes that "the idea of mana and the various conceptions related to it are not bound to a particular realm of *objects* (animate or inanimate, physical or spiritual), but that they should rather be said to indicate a certain 'character,' which may be attributed to the most diverse objects and events, if only these evoke mythic 'wonder' and stand forth from the ordinary background of familiar, mundane existence.… It is not a matter of 'what,' but of 'how'; not the object of attention, but the sort of attention directed to it, is the crucial factor here. Mana and its several equivalents do not denote a single, definite predicate; but in all of them we find a peculiar and consistent *form of predication*. This predication may indeed be designated as the primeval mythico-religious predication, since it expresses the spiritual 'crisis' whereby the holy is divided from the profane."[19]

The idea of the *sacred*, especially in its sense as the opposite of the profane, expresses even better, therefore, how the primitives perceive mana. This is significant, if we want to understand the belief in *software* power. Like mana, software power is a potency that can manifest itself in diverse concepts and entities, so it does not describe their *type* but their *character*. By asserting that a thing has power, the believer says, in effect, that he perceives it as belonging in the domain of the sacred rather than the ordinary.

So the belief in software power, like the primitive beliefs, is a belief in the existence of miraculous capabilities – capabilities which cannot and need not be explained. In the following passage, Eliade describes the concept of mana, but this can just as easily describe the concept of *software* power: "Among the 'primitives' as among the moderns, the sacred is manifested in a multitude of forms and variants, but … all these hierophanies are charged with *power*. The sacred is strong, powerful, because it is *real*; it is efficacious and durable. The opposition between sacred and profane is often expressed as an opposition between the *real* and the *unreal* or pseudo-real. Power means *reality* and, at the same time, *lastingness* and *efficiency*."[20]

Software power, then, is the modern counterpart of mana. We can confirm this by noting the many similarities between the two beliefs. First, and most

[18] Nathan Söderblom, quoted in Cassirer, *Language and Myth*, p. 66.

[19] Cassirer, *Language and Myth*, pp. 65–66.

[20] Eliade, *Myths, Dreams, and Mysteries*, p. 130. (The term *hierophany* was coined by Eliade to denote any manifestation of the sacred.)

significantly, everyone realizes that supernatural power acts like a tool, or like an appliance: we can benefit from it directly, without having to gain new knowledge. Thus, the primitives understand that "mana is an object, not a body of skills and abilities which are obtained through learning. Access to it is acquired, in the sense that a house or a wife or a spear is acquired, that is as a gift, as a purchase, or through the performance of appropriate acts."[21] Similarly, the believers in *software* power do not expect to acquire any skills by using software devices. They understand that this power is a *substitute* for knowledge and experience. Vendors, in fact, make this point the main attraction of software devices: simply by purchasing one, you gain access to a power that will allow you to accomplish your tasks immediately.

Second, supernatural power is perceived by everyone as a truly general potency. For the primitives, mana "is not so much the idea of … particular embodiments, as the notion of a 'power' in general, able to appear now in this form, now in that, to enter into one object and then into another."[22] Similarly, the great variety of means by which we can acquire *software* power shows that believers do not associate it with *specific* things – a company, a product, a function – but with a universal potency that can materialize in any software-related concept. It can appear in development environments as well as in applications, in database systems as well as in utilities, in user interface as well as in computations.

And, although we are discussing *software* power, we must note that this universal potency can materialize in anything else associated with computers. Thus, it can appear in whole computers (Power Mac, PowerBook, PowerEdge, AcerPower, Power Spec, Prime Power), and also in the *parts* of a computer, and in related devices: in a monitor ("empower your business with advanced display technology,"[23] "… these stylish, powerful and efficient monitors improve the atmosphere of any desktop"[24]), a graphics card ("Radeon 7500 is a powerful and versatile graphic solution,"[25] "GeForce GTX 480 powers interactive raytracing"[26]), a hard drive ("fast performance and huge capacity to power today's storage-hungry applications"[27]), a motherboard ("empowered by integrated graphics and Intel Hyper-Threading Technology …,"[28] "it delivers awesome power …"[29]), a scanner ("empower your information management with digital technology"[30]), a network device (PowerConnect switch), a mouse

[21] Swanson, *Birth of the Gods*, p. 6.
[22] Cassirer, *Language and Myth*, p. 63.
[23] NEC Corp., adv.
[24] http://www.samsung.com/.
[25] http://ati.amd.com/.
[26] http://www.nvidia.com/.
[27] Seagate ST3160316AS Barracuda 7200.12, http://www.tigerdirect.ca/.
[28] Asus P4V8X-MX motherboard, http://ca.asus.com/.
[29] Gigabyte GA-X58A-UD3R motherboard, http://www.acousticpc.com/.
[30] Ricoh Aficio scanners, *The Ricoh Report* (Nov. 2000).

(Power Wheelmouse, PowerScroll), a storage system (PowerVault), a CD device (PowerCD, PowerDisc), a processor (PowerPC), a camera (PowerShot), or a microphone (PowerMic). And it can appear even in such concepts as a newsletter (IBM PowerTalk, APC PowerNews), a business relationship (Samsung Power Partner program), a panel discussion (Power Panels[31]), a trade show ("over 700 high-powered exhibits"[32]), or a course ("a powerful 3-day course"[33]).

Lastly, the term "power," like "mana," is employed in a variety of grammatical roles. Analyzing the ways in which the Sioux use "wakanda," McGee notes that "the term was applied to all sorts of entities and ideas, and was used (with or without inflectional variations) indiscriminately as substantive and adjective, and with slight modification as verb and adverb."[34] Similarly, through its derivatives, "power" is used indiscriminately as noun, adjective, verb, and adverb. Let us see some examples.

As noun: "Discover the power of MetaFrame and WinFrame software."[35] "Relational database power made easy."[36] "The power to build a better business Internet."[37] "This empowerment is most visible in backend solutions like servers and networks."[38] "Experience the power of software instrumentation."[39] "SaaS Business Empowerment programs are designed to help Progress' SaaS partners focus on the early-stage fundamentals …."[40] "Accrisoft Freedom web empowerment software provides all the tools you need …."[41] "… AutoPlay Media Studio gives you the power to quickly create just about any software application you can dream up."[42] "IT empowerment with ITSM education from Hewlett-Packard."[43] "Enjoy visual power."[44]

As adjective: "Powerful network storage software with built-in intelligence and automation …."[45] "Discover hundreds of new uses for this empowering tool."[46] "Visual Two-Way-Tools for power programming."[47] "Powerful software for solving LP, NLP, MLP and CGE models."[48] "Control your duplicate files with this powerful utility."[49] "This powerful feature allows affiliates to

[31] Comdex Canada Exhibition (1995), adv. pamphlet.
[32] Database and Client/Server World Exposition (1994), adv.
[33] Global Knowledge, adv. pamphlet.
[34] William McGee, quoted in Cassirer, *Language and Myth*, p. 68.
[35] Citrix Systems, Inc., adv. pamphlet.
[36] Borland Paradox for Windows, adv. pamphlet.
[37] Oracle Corp. iDevelop 2000 event, adv. pamphlet.
[38] http://www.netc.org/.
[39] http://www.ocsystems.com/.
[40] http://web.progress.com/.
[41] http://accrisoft.org/.
[42] http://www.indigorose.com/.
[43] Hewlett-Packard Company, adv.
[44] Microsoft Visual Basic 2.0, adv. pamphlet.
[45] http://www.compellent.com/.
[46] http://www.indigorose.com/.
[47] Borland Delphi, adv. pamphlet.
[48] http://web.uvic.ca/.
[49] http://www.kewlit.com/.

create advertising channels."[50] "Simple, useful and powerful software tools."[51] "Powerful database design made simple."[52] "A powerful software tool to tweak, optimize, maintain and tune up your Windows XP …."[53] "Develop powerful Internet applications."[54] "Create powerful, dynamic Windows programs."[55] "A powerful, easy-to-use process improvement tool."[56]

As verb: "Oracle software powers the Internet."[57] "We can power you, too."[58] "Empowered by innovation."[59] "MV Software has been powering business solutions for over two decades."[60] "Empower employees to collaborate and innovate."[61] "Windows Principles: … empowering choice, opportunity, and interoperability."[62] "XML: powering next-generation business applications."[63] "Learning powered by technology."[64] "Utoolbox.com … is powered by a dedicated team of professionals."[65] "Empowering software engineers in human-centered design."[66] "Empowering software debugging through architectural support for program rollback."[67] "Powering the lean, consumer-driven supply chain for manufacturers worldwide."[68] "We can empower your organization through adoption of IT Service Management …."[69] "Data Query empowers the end user to create reports …."[70] "Empowering software maintainers with semantic web technologies."[71] "Powering on demand applications."[72] "Powering the digital age."[73]

As adverb: "Accurate Shutdown is a powerfully automatic software that turns off your computer at the user-specified time."[74] "RSConnect Suite corporate management software: … powerfully simple, powerfully quick."[75] "QSR software … provides a sophisticated workspace that enables you to work through your information efficiently and powerfully."[76] "XP Picture Manager can correct your photos powerfully and quickly."[77] "The building blocks of

[50] http://www.qualityunit.com/.
[51] http://www.utoolbox.com/.
[52] SDP Technologies S-Designor, adv. pamphlet.
[53] http://www.freedownloadscenter.com/.
[54] Microsoft Visual Studio 6.0, adv. pamphlet.
[55] Borland Turbo Pascal for Windows 1.5, adv. pamphlet.
[56] IEEE Computer Society Press, *LearnerFirst Process Management*, adv. pamphlet.
[57] Oracle Corp., adv.
[58] Dell Computers, adv.
[59] http://www.nec.com/.
[60] http://www.mvsoftware.com/.
[61] Cisco Systems, adv.
[62] http://www.microsoft.com/.
[63] http://www.dbmag.intelligententerprise.com/.
[64] Brochure subtitle, U.S. Dept. of Education, *Transforming American Education* (2010).
[65] http://www.utoolbox.com/.
[66] http://portal.acm.org/.
[67] http://iacoma.cs.uiuc.edu/.
[68] http://www.jda.com/.
[69] Global Knowledge, *IT and Management Training* catalogue (Dec. 2006), p. 12.
[70] Oracle Discoverer/2000, adv. pamphlet.
[71] http://www.rene-witte.net/.
[72] https://www-304.ibm.com/.
[73] http://www.swiftdisc.com/.
[74] http://www.accuratesolution.net/.
[75] http://www.necpos.com/.
[76] http://www.qsrinternational.com/.
[77] http://www.softtester.com/.

virtual instrumentation include powerfully productive software ….”[78] “HP StorageWorks Command View EVA software provides you with a powerfully simple storage management experience ….”[79] “The intelligent technology in our electrical calculation software powerfully calculates and performs your electrical calculations and designs ….”[80] “Powerfully advanced mailing software.”[81]

In addition, the phrase “powered by” is commonly used in promotional slogans to mention a given product, in place of a phrase like “made by,” “works with,” or “employs.” Some examples of this practice: “powered by Google,” “powered by IBM,” “powered by Sun,” “powered by AOL Mail,” “powered by Microsoft Access,” “powered by XMB,” “powered by Cognos,” “powered by FIS,” “powered by Mozilla,” “powered by HitsLink,” “powered by PayPal,” “powered by WebsiteBaker,” “powered by Trac,” “powered by ATI,” “powered by Merril Lynch,” “powered by Geeklog,” “powered by vBulletin,” “powered by eBay Turbo Lister,” “powered by GetSimple,” “powered by TAXWIZ,” “powered by nexImage,” “powered by MindTouch,” “powered by Joomla,” “powered by ShopFactory,” “powered by Network Solutions,” “powered by Sothink.”

3

As programmers and as users, we wish to benefit from the power of software, but without taking the time to develop software expertise. Consequently, we have come to regard this power as the kind of power that we can *acquire*. And it is through the devices supplied by software companies that we hope to acquire it. So, when describing their devices as powerful, the software companies are simply exploiting this belief.

Like all beliefs we carry from our primitive past, the belief that certain devices possess a mysterious power can only be dispelled through learning. As in other domains, once we possess the necessary skills in software-related matters, we can easily recognize which devices are helpful and which ones are fraudulent. In a rational society, this education would be the responsibility of the software elites – the universities, in particular. In our society, however, the opposite is taking place: since the elites can profit far more by exploiting society than by educating it, ignorance and primitive beliefs serve their interests. Thus, only if we remain ignorant will we believe that their devices, which are based on mechanistic concepts, can solve our complex problems. So the elites are doing all they can to *prevent* us from developing software knowledge.

[78] http://www.scientific-computing.com/.
[79] https://ads.jiwire.com/.
[80] http://solutionselectricalsoftware.com/.
[81] http://www.satorisoftware.co.uk/.

Software devices can replace expertise only in solving *mechanistic* problems; that is, problems which can be broken down into simpler and simpler ones, and hence modeled with isolated hierarchical structures. Most problems we want to solve with software, however, are non-mechanistic. They can only be represented as systems of interacting structures, so they require a human mind, and expertise. The problems associated with programming, particularly, are of this kind. In the end, less than 1 percent of the software devices we are offered are genuine, beneficial tools; the rest are fraudulent. What distinguishes the latter is their claim to solve complex, non-mechanistic problems; in other words, to act as substitutes for minds. They address naive programmers and users, promising them the power to accomplish tasks that require, in fact, much knowledge and experience.

So the software elites are not responsible organizations, but charlatans. They present their devices as the software counterpart of the traditional tools and instruments, but at the same time they invoke the notions of magic and supernatural power. They tell us that we need these devices in the same way that engineers and doctors need theirs. But the tools and instruments we use in engineering and in medicine are promoted on the basis of real qualities, and provide real benefits. Their vendors do not exploit our ignorance and irrationality when persuading us to use them. Clearly, then, if *software* devices must be promoted in this fashion, it is because they are generally useless, because the possession of an imaginary power is their only quality. To put it differently, if software devices were promoted by demonstrating their *real* benefits, we would use only the few that are truly useful.

The harm caused by this charlatanism extends, however, beyond the waste of time and resources. For, when restricted to the mechanistic knowledge required to operate devices, we forgo all opportunities to develop complex, non-mechanistic knowledge. Without this knowledge we cannot solve our complex problems. But if we believe that it is only through devices that we can solve them, we continue to depend on devices, and hence to restrict ourselves to mechanistic knowledge, in a process that feeds on itself. The only way to escape from this vicious circle is by expanding our knowledge, so as to exceed the mechanistic capabilities of devices. And we cannot do this as long as we agree to depend on them. Thus, by enticing us with software devices, the elites ensure our perpetual ignorance. They prevent us from gaining knowledge and also from solving our problems.

The propaganda depicts the software elites as enligthened leaders who are creating a new world for us – a world with higher and higher levels of efficiency. But now we see that the reality is very different: they are fostering ignorance and irrational beliefs, so they are creating a *less* efficient world. When presenting their devices as magic systems or as sources of supernatural

power, they are encouraging us to behave like the primitives. This degradation started with the software practitioners, in their programming activities. Now, as our dependence on computers is spreading, it is increasingly affecting everyone, in every activity.

Bear in mind, though, that it is not software or programming that causes this degradation, but *mechanistic* software and programming, the kind promoted by the software elites. Mechanistic software-related activities restrict us to mechanistic thinking, thereby preventing us from using our natural, non-mechanistic capabilities. Left alone, without software elites and the mechanistic dogma, human beings would learn to develop and use software as effectively as their minds permit them. Complex software phenomena, and complex software knowledge, would then join the many other complex structures that make up human existence. Our software-related activities would then *enhance* our minds, as do other complex phenomena (the use of language, for instance).