SOFTWARE AND MIND

Andrei Sorin

EXTRACT

Introduction: *Belief and Software*
Section *Anthropology and Software*

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

This section explores the similarity of mechanistic software beliefs to primitive beliefs.

The entire book, each chapter separately, and also selected sections, can be viewed and downloaded free 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 (www.andsorbooks.com)
First edition 2013. Revised 2019.

For disclaimers see pp. vii, 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

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

This revised version (currently available only in digital format) incorporates many small changes made in the six years since the book was published. It is also an opportunity to expand on an issue that was mentioned only briefly in the original preface.

Software and Mind is, in effect, several books in one, and its size reflects this. Most chapters could form the basis of individual volumes. Their topics, however, are closely related and cannot be properly explained if separated. They support each other and contribute together to the book's main argument.

For example, the use of simple and complex structures to model mechanistic and non-mechanistic phenomena is explained in chapter 1; Popper's principles of demarcation between science and pseudoscience are explained in chapter 3; and these notions are used together throughout the book to show how the attempts to represent non-mechanistic phenomena mechanistically end up as worthless, pseudoscientific theories. Similarly, the non-mechanistic capabilities of the mind are explained in chapter 2; the non-mechanistic nature of software is explained in chapter 4; and these notions are used in chapter 7 to show that software engineering is a futile attempt to replace human programming expertise with mechanistic theories.

A second reason for the book's size is the detailed analysis of the various topics. This is necessary because most topics are new: they involve either

entirely new concepts, or the interpretation of concepts in ways that contradict the accepted views. Thorough and rigorous arguments are essential if the reader is to appreciate the significance of these concepts. Moreover, the book addresses a broad audience, people with different backgrounds and interests; so a safe assumption is that each reader needs detailed explanations in at least some areas.

There is some deliberate repetitiveness in the book, which adds only a little to its size but may be objectionable to some readers. For each important concept introduced somewhere in the book, there are summaries later, in various discussions where that concept is applied. This helps to make the individual chapters, and even the individual sections, reasonably independent: while the book is intended to be read from the beginning, a reader can select almost any portion and still follow the discussion. In addition, the summaries are tailored for each occasion, and this further explains that concept, by presenting it from different perspectives.

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.

The book's epigraph, about Newspeak, will become clear when we discuss the similarity of language and software (see, for example, pp. 409–411).

Throughout the book, the software-related arguments are also supported with ideas from other disciplines – from the philosophies of science, of mind, and of language, 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.

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, and their long history. 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.

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 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). The plural, "elites," is used when referring to several entities within such a body.

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 have published, in source form, some of the software 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.

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 (see pp. 2–4), 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. 497–498.)

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, then, software magic creates 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).