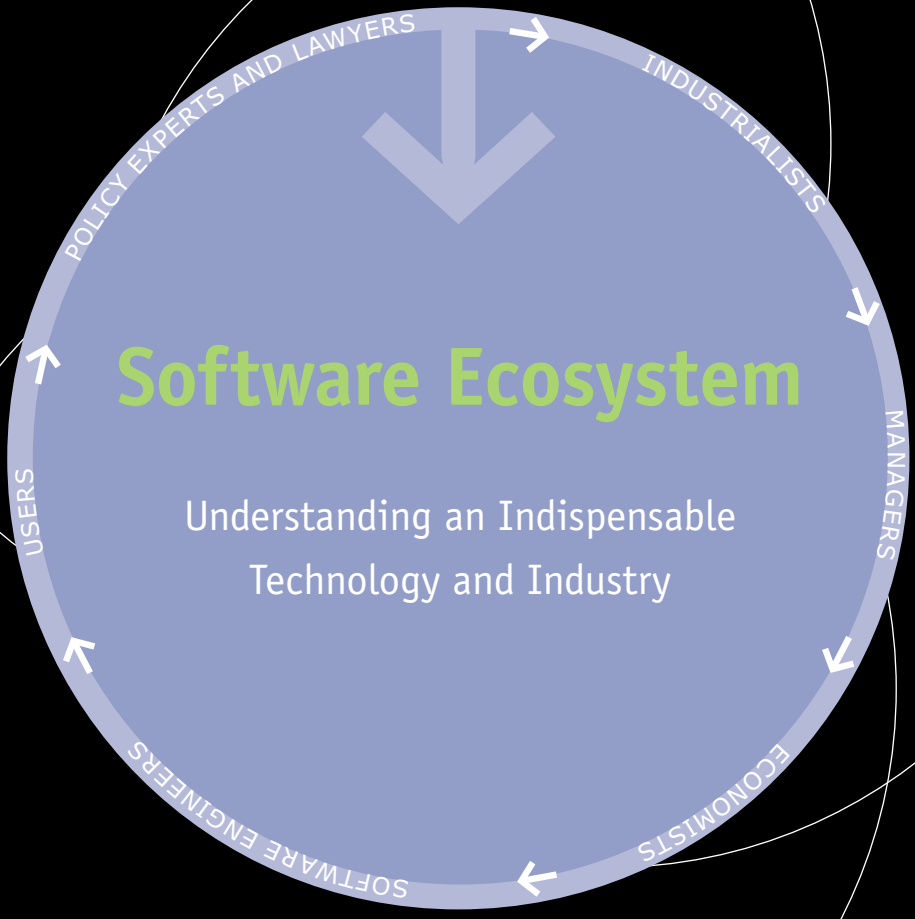


# Software Ecosystem

## Understanding an Indispensable Technology and Industry

David G. Messerschmitt and Clemens Szyperski

# Software Ecosystem

# Software Ecosystem

## Understanding an Indispensable Technology and Industry

David G. Messerschmitt and Clemens Szyperski

The MIT Press
Cambridge, Massachusetts
London, England

This book was set in Sabon by SNP Best-set Typesetter Ltd., Hong Kong
Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Messerschmitt, David G.
Software ecosystem : understanding an indispensable technology and industry / David G. Messerschmitt and Clemens Szyperski.
p. cm.
Includes bibliographical references and index.
ISBN 0-262-13432-2 (hc. : alk. paper)
1. Computer software. 2. Computer software—Development. 3. Computer software industry. I. Szyperski, Clemens. II. Title.

QA76.754.M47 2003
005.3—dc21 2002044404

10 9 8 7 6 5 4 3 2 1

To Dody and Laura
—D.G.M.

To Bianca, Leonie, Lennard, Amelie, and Luca
—C.A.S.

# Contents

# Preface

In this book, software technology and the myriad issues that surround its dissemination and use are examined from a number of relevant perspectives. This is especially timely as the importance of software in the overall industrial economy grows, and as the software industry undergoes important transformations. The tremendous success of the Internet is noteworthy in forever changing the structure of the industry, the applications of software, and software business models.

Most books on software focus on software technology or development methodologies. To be successful, software professionals need to appreciate the context of software in the real world. Other professionals like managers, lawyers, and economists with an interest in software and its uses and effects need to appreciate more deeply the technical aspects of software, at least those relevant to their own contexts. They need a vocabulary and a common frame of reference. This book addresses this need by explaining the characteristics of software in its context, emphasizing both technical and nontechnical issues, and relating the two.

The software industry cannot be fully appreciated outside the context of what precedes and what follows development. Conceptualizing what it does, the needs it serves, and its effects is an important prerequisite for understanding its development. Many issues follow software deployment, including provisioning, operations, use, maintenance and upgrade, customer service, and coordinated organizational changes and management challenges. Thus, an overall goal of this book is to integrate an explanation of pre- and post-development activities with a description of software development and technology.

The software industry is itself very complex, with many complementary products necessary to form a systems solution and complex alliances and standardization processes needed to meet the needs of numerous stakeholders. Together, the software suppliers, standardization bodies, content suppliers, service providers, and

end-user organizations form a complex web of relationships. The "ecosystem" metaphor is truly descriptive.

The growing importance and special characteristics of software increasingly make it an area of activity for government and an interesting topic of study for the social sciences, law, and business. We capture the most important issues raised by the growing importance of software in the management, policy, and legal arenas, and relate them to the characteristics of software technology and business models.

The overarching theme of the book is that software is different. It is different from information goods in terms of its economic properties: although it shares some characteristics with information goods, with material goods, and even with services, it mixes these characteristics in unique ways. It requires a different legal perspective, as evidenced by the increasing attention given to software-specific issues in legislation and the courts, including patents, copyrights, civil liberties, and antitrust. It is distinctive among technologies and in its industrial and business challenges in numerous ways. The single most prominent objective of this book is to explain how software is different.

In summary, software touches many professional lives. This leads to many interrelated and overlapping perspectives on software, technical and nontechnical. Following introductory chapters, the book is organized around the perspectives of the user, software engineer, manager, industrialist, lawyer or policy expert, and economist as well as the overlaps and relationships among these perspectives. The objective of the book is to capture the key software issues and concerns of each profession, to describe the characteristics of software ecosystem as well as the relevant business relationships and processes that surround it, and then to explain these characteristics of software in a way that readers can understand even if they are not technologists.

Each chapter is not written primarily to address the needs of the associated professions; indeed, that chapter is likely to be rather superficial to an expert. Rather, the chapters are intended to convey a comprehensive view of the software ecosystem for the benefit of practitioners of each and every perspective. The book is primarily about software, so we make no attempt to be comprehensive in our treatment of the surrounding issues; rather, we explain them in sufficient detail to be able to relate them substantively to software. We do attempt to be reasonably comprehensive in explaining software itself, and in an accessible way, while avoiding numerous detailed technical issues of keen interest to technologists but less relevant from other perspectives.

The software industry has always undergone rapid change, but arguably this change has been never more swift than today, principally because of the Internet. We firmly believe that software would benefit from more in-depth research in economics, management, policy, and legal disciplines. For example, the opportunities surrounding the growing role of the service provider as an intermediary between software and the user are poorly understood but are nevertheless being very actively pursued. If you are a scholar interested in studying software-related issues, we aim to empower you by explaining arcane but relevant characteristics of technology and processes. As additional aids, a list of specific research and discussion issues is included at the end of each chapter and a glossary of the most important terms at the end of the book.

We are sensitive to the issue that since one co-author is an employee of Microsoft Corporation, the book may promulgate a Microsoft-centric view of the software world. As this is not our intention, we have given equal weight to numerous examples from across the software industry and have attempted to represent alternative viewpoints on many controversial issues rather than advocating a single viewpoint.

If you are a technical professional, you should also have a deeper appreciation and understanding of the many nontechnical processes, relationships, and issues that surround software. If you have an interest in molding the technology in ways that make it more useful and successful, enhance its positive outcomes, and mitigate its possible problems, you should find this book a helpful starting point.

This book is in part an outgrowth of a course taught at the University of California at Berkeley, "Strategic Computing and Communications Technology," to engineering, information management, and business administration students. It should prove useful in similar classes that examine the industrial and economic considerations of software: business, economics, and law courses, and courses for software professionals.

The book has an associated Web site, ⟨http://mitpress.mit.edu/softeco⟩, where supplementary materials like hyperlinks to resources referenced in the book, more detailed or analytical write-ups, and presentation slides are posted. This should be particularly valuable to instructors using this book.

This work was first published as a technical report (Messerschmitt and Szyperski 2001). Messerschmitt is indebted to Hal Varian, who co-taught the "Strategic Computing" course in 1997 and helped to refine and expand it in subsequent years, and to Carl Shapiro, who co-taught the course in 2001 and provided numerous helpful comments on the manuscript of this book. Others provided many useful comments on preliminary versions of the book, including Glenn Woroch of the University of

California at Berkeley, Leonard Cheng of the Hong Kong Institute of Science and Technology, Cuno Pfister of Oberon Microsystems, Zurich, and several anonymous reviewers. Messerschmitt has benefited from discussions with others with a deep and abiding interest in the software and communications industries, including Michael Katz and Joseph Farrell of Berkeley and Jean-Pierre Hubaux of the Swiss Federal Institute of Technology in Lausanne. Szyperski has benefited from discussions with Brad Cox, Ron Kay, and Norbert Szyperski.

# Software Ecosystem

# 1
# Introduction

[I want] minimum information given with maximum politeness.
Jacqueline Kennedy

Jacqueline Kennedy would likely be disappointed nowadays in her preference for receiving "minimum information." Information is proliferating wildly, and software is partly to blame. And sometimes the presentation of this vast information is none too polite, but hopefully this book is an exception.

The software industry has become crucial to the global economy. It is a large and rapidly growing industry in its own right, and its secondary impact on the remainder of the economy is disproportionate. In light of this, the software industry deserves much focus and attention by nontechnical professionals in fields such as management, economics, policy, and law. Obstacles to understanding include the often arcane and inaccessible nature of software technology, and a lack of visibility into the internal workings of the software industry and software in its context of uses. Individual productivity and Internet applications are familiar to us all, but the important role that software plays in organizations of all types is less visible though arguably far more influential. Controversial policy issues surrounding copyright enforcement and access to pornography by children are highly visible, but the public debate rarely takes adequate account of the opportunities and challenges afforded by the ubiquitous, embedded nature of today's software technologies.

This book seeks to explain to a broad technical and nontechnical audience the characteristics of software technology and of the business of software creation, and the context within which software is deployed, operated, and used. Given the growing importance of software and the rapid changes occurring in how it is developed, sold, and used, this is an opportune time for a comprehensive examination of the many facets of software.

## 1.1 What Makes Software Interesting?

Software has always been interesting to computer scientists, who appreciate its arcane details and deep technical challenges. But why should it be interesting to other professions? One reason has already been noted: It is the foundation of an important industry, and it is making an increasingly important contribution to the world's economy and raising a number of challenging policy and legal issues. Fortunately, there is much more to it than that—software is intellectually interesting from a wide variety of perspectives. The following are some interesting aspects of software; each is described here briefly and elaborated throughout the book.

### 1.1.1 Software Is Different

The software industry started from scratch only about fifty years ago, and has arguably become indispensable today, not only in its own economic effect but also in the efficient and effective management of many other industries. This journey from obscure to indispensable is by no means unique; for example, witness the earlier technological revolutions wrought by electrification or the automobile or communications. It is interesting to compare software to other industries, looking for parallels that might offer insights to the software industry. A basic thesis of the book is that software and the software industry are simply different; while such parallels are useful, they are inadequate to explain the many characteristics of the software industry.

Consider software as an economic good. It does have many characteristics that are individually familiar, but they are collected in unusual combinations. For example, like writing a novel, investment in software creation is risky, but unlike writing a novel, it requires collaboration with its users in defining its features. Like an organizational hierarchy, software applications are often essential to running a business, but unlike an organizational hierarchy, software is often designed by outside vendors with limited ability to adapt to special or changing needs. Although like many material goods and even human employees, software is valued for what it does, unlike material goods it has practically no unit manufacturing costs and is totally dependent on an infrastructure of equipment providing its execution environment. To a considerably greater degree than most material goods, a single software application and its supporting infrastructure are decomposed into many internal units, called modules (see chapter 4), which are often supplied by different vendors and with different ownership. Even the term *ownership* has somewhat different connotations than when used for material goods, because it is based on

intellectual property laws rather than on title and physical possession. As with many other goods, the use of software includes an important operational side, but these operations are often split across a multitude of administrative domains with a limited opportunity for coordination. Like information, software is usually protected by copyright, but unlike information it can also incorporate patented inventions.

These examples, and others developed later, suggest that the software industry—as well as interested participants like the end-user, service provider, and regulatory authorities—confronts unique challenges. In addressing these challenges, it is important to appreciate the many facets of software and how it is created and used in the real world.

### 1.1.2 Software Is Ubiquitous

Software is everywhere! Many of us use computers and software on a daily basis, often many hours per day. Many material products that have a behavior, that do something, have software embedded within. Software has become a part of our lifestyle, but even more fundamentally it has become integral to the operation of organizations of all types, including industry, education, and government. The operations of most organizations are as dependent on software as they are on their human workers, and dependent as well on the smooth interworking of software with workers. This puts a burden on many professions and disciplines whose work bears on individuals or organizations to develop and incorporate a deeper understanding of software.

### 1.1.3 Software Makes Our Environment Interactive

One of software's primary effects is to change our environment in fundamental ways. In the past, our environment was mostly built; we were surrounded by largely passive objects and artifacts, and most of our interaction was with other people. Increasingly, software is creating an environment in which we interact with inanimate objects in increasingly sophisticated ways.

### 1.1.4 Software Is Important

In our information age and information society, information has become an important commodity. More and more workers are information workers, who manipulate and manage information as their primary job function, or knowledge workers, who create and use new knowledge based on assembling and assimilating large bodies of information. Software has become a primary vehicle by which the most

mechanistic aspects of information acquisition, organization, manipulation, and access are realized, supplemented by the insights and actions of people. The growing importance of information and knowledge lead directly to the growing importance and role of software.

The addition of communication to the suite of information technologies (manifested primarily by the Internet) is dramatically increasing the importance of software to organizations and other social institutions. A primary distinguishing feature of groups, organizations, communities, and society is their patterns of discourse and communication, and software today joins mass transportation as a technology having a profound effect on those patterns. This is an evolution in its infancy, and software should only grow in significance over time.

### 1.1.5 Software Is about People

It is tempting to think of software in terms of instructions and bits and bytes, but fundamentally, from a user perspective, those technical constructs are largely irrelevant. Software is really the expression and representation of the wishes and actions of a human programmer in the context of an executing software program. As a useful thought model, consider a computer or material product with embedded software as having a person inside, that person responding to external stimulus and defining what actions to take next. In reality, that person is the programmer (or often a large team of software developers) who has anticipated all the possible external stimuli and expressed the resulting actions through the software she writes. Thus, software is really an expression of the behavior defined by a programmer in much the same way that a novel is the expression of plot and emotion.

Increasingly, software expresses and represents an information-based business process, along with a human organization and its participants. Software works in concert with a human organization, and the technological and social aspects of this combination are deeply interwoven.

### 1.1.6 Software Can Be Better

Software suffers from no significant practical physical limitations. Much more so than material goods, software can be molded pretty much as we choose. A lot of what we have today is the result of many arbitrary choices; the most significant limitation is conceptual bottlenecks and the remnants of history. We have an opportunity to make better choices in the future if we take account of the needs of users, end-user organizations, and societal needs. However, if this is to happen, we have to transcend the common perception of software as a manifestation of technology

and think more deeply and clearly about software an important enabler of individuals, groups of individuals, organizations, communities, and society as a whole. We have to consider software a human artifact or tool, one that can be almost infinitely flexible and that could bring greater benefits if only it could be molded more appropriately.

### 1.1.7 The Software Industry Is Undergoing Radical Change

A number of factors are converging to radically change the software industry, and underlying many of them is the astounding success of the Internet. The Internet is not only opening up entirely new categories of applications. It is irrevocably changing the ways in which software is sold and entangling software technologies in contentious debates over many public policy issues, including national sovereignty, national security and law enforcement, and limitations to information access.

The fundamental structure of the software-related industries has undergone the change characteristic of an immature industry, but arguably never has that change been more rapid or fundamental than now. This creates interesting opportunities to consider different models for the industry, and their relative merits and challenges.

### 1.1.8 Creating Software Is Social

We are familiar with the "nerd" image of social isolation associated with programmers. Like many such prejudices, there is some truth to it but also a substantial falsehood and oversimplification. Most of the desirable properties that make software successful are not a result of technical prowess but of a deep understanding of what the users of a program need and want, and how they can be accommodated in an efficient, natural, and even enjoyable way. The creators of the best and most successful application software must think much more like managers, psychologists, and sociologists than technicians. Good software cannot be created without a strong connection to all the stakeholders, which includes not only users but also managers, administrators, operators, and others. Taken in its entire context, identifying and analyzing needs through operations and use, creating and managing software is ultimately a highly social activity.

On the other hand, the programming phase of creating software requires spending a lot of time in front of a computer. In this respect, creating software is no different than creative writing, not generally considered a "nerdy" activity. How does programming differ from creative writing? On the one hand, software is almost always undertaken by large teams, an inherently social enterprise, and writing isn't. But the perceptual gap probably arises from the programmer's focus on technical

detail and the writer's greater attention to plot, to artistic expression, and emotional connections. This distinction is changing rapidly. Programming tools are reducing the technical content of application creation, especially as they allow the customization of existing applications. As software becomes more integral to the lifestyles of its users, and integral to the functioning of organizations, successful programmers form a strong emotional bond with the ultimate users. So, too, with expanding computer power, with software enabling good tools and rich graphics and sound, application programming assumes the flavor of an artistic rather than a technical endeavor.

### 1.1.9 Software Is Sophisticated and Complex

In contrast to every other economic good except information, software does not suffer from any important physical limits; it can be molded in almost any way we wish. For this reason, the limitations on what we can do with software are primarily social, economic, and conceptual, not physical. The sophistication and complexity of software grows to the point where it can no longer be managed because of conceptual limitations; no amount of money or resources can overcome these human limits. Understanding this unique aspect of software is essential to understanding the challenges faced by the software industry in moving forward.

If we think of an executing software program as a surrogate agent for its creators, one great challenge is that responsible actions must be predefined for all circumstances that may arise, not only the normal and expected but also the abnormal and exceptional. This makes accomplishing a complex task through software much more challenging than accomplishing the same task via a human organization because people can be counted on to react intelligently to exceptional circumstances.

Although it wasn't always true, post-Internet the software industry faces a monumental challenge in industry coordination. Infrastructure from many vendors must work together, and applications must work with infrastructure and work in complex ways across organizational boundaries. Other industries face coordination challenges—train locomotives have to fit with the tracks, and lubricants have to match the design of machinery—but arguably none face as wide-ranging and complex a coordination challenge as the software industry.

### 1.1.10 Software Can Be Tamed

Technologies are morally neutral; each can be used for good or for ill. We have a collective opportunity and responsibility to mold software in ways that give it higher utility and mitigate its possible negative effects. The potential deleterious effects of

software are numerous and well publicized. They include increasing the social disadvantage of poor or handicapped citizens, curtailment of free speech and civil liberties, stifling of innovation through excessive reliance on intellectual property, an overabundance of information and communication, and many others. Addressing these challenges is largely the role of public advocates, policy experts, politicians and lawmakers, but to do the job properly they must understand more fully the capabilities and limitations of software and the industries that surround software.

## 1.2 Organization and Summary

In the real world, software touches the personal and professional lives of most individuals. Of course, for most people this is in the sense of being a user and beneficiary of software. For a smaller (but still very significant) number of people, their professional lives are directly involved in facilitating the creation and use of software, or understanding its effects on individuals, organizations, or society. This book is primarily dedicated to capturing the perspective of these professionals.

This book is organized around the perspectives of different professions regarding software. Following an introductory chapter on information technology (chapter 2), the remaining chapters are organized around six such perspectives, and each chapter also relates its own perspective to the others. Users (chapter 3) are affected by what software does on their behalf. Software engineers and other professionals involved in software creation (chapter 4) start with a good understanding of what users want to accomplish and end by creating software code. Throughout the software lifecycle, managers (chapter 5) must deal with a number of issues, such as acquiring the necessary infrastructure to run the software, organizing the people and business processes that surround the software, and operating the software applications and infrastructure. Industrialists (chapters 6 and 7) organize themselves into companies to create and manage software and are concerned about ownership and business relationships and bringing together the complementary infrastructure and functions that together make the benefits of software available to users. Policy experts and lawyers (chapter 8) are concerned with ensuring a healthy and innovative industry, mitigating possible negative effects of information technology, and resolving various conflicts among the industry participants. Economists (chapter 9) offer many useful insights into the workings of the software marketplace. The book concludes with a look into the future (chapter 10).

Many issues affect more than one of these perspectives. In each case, we locate that issue in the chapter that seems most relevant but also relate it to the other perspectives as well.

A greater appreciation for the variety of issues surrounding software and their interdependence can be facilitated by starting with an overview of the book. Software works in conjunction with information content and the information technologies, as described in chapter 2. All information, as well as software itself, can be represented by data (collections of bits), manipulated by a processor, stored for future access, and communicated over a network. Information is valued for how it teaches or influences us, whereas software is valued for what it does, that is, its behavior. Many legal, economic, and business considerations for information and software are influenced by the simplicity of creating perfect replicas of information and software. Software depends on a complementary material technological infrastructure for its execution, including both hardware (processor) and infrastructure software (e.g., operating system). The software industry has benefited from several decades of geometric advance in the performance per unit cost in processing, storage, and communication technologies, an observation known as Moore's law. To understand the future of software, it is important to understand among other things the driving forces behind Moore's law and how it may evolve in the future.

The primary value of software is in the features and capabilities it provides the end-user, as described in chapter 3. Historically, applications have been largely driven by the addition, one after the other, of processing, storage, and communication to the suite of information technologies. The major classes of applications include scientific modeling and simulation, databases and transaction processing, information publication and access, collaboration and coordination, and software embedded in larger systems. Today, many applications strongly exploit all three technologies. Many of the most valuable generic applications supporting large numbers of users have doubtless been invented and refined, so today applications are becoming increasingly specialized to meet the needs of particular vertical industries, types of organization, or organizational functions. Sociotechnical applications, one growth category, emphasize the integration of a human organization with information, information technology, and software. As a result of the deeper integration of software applications into human and organizational activities, the process of understanding user needs and how the features and capabilities of software applications can meet them has become an important design activity separate from technical software implementation.

A number of factors contribute value to the user of software, including specific features and capabilities, effects on an organization, amount of use, quality, performance, usability, flexibility, and extensibility. One consideration that recurs throughout the book is network effects, wherein the value to the user depends on the number of other adopters of particular software.

The process of creating software, and some of the technical characteristics of software most relevant to business models and economic properties of software, is considered in chapter 4. A major issue in creating software is the software development process, which deserves careful attention and must be highly principled because of the increasing size of investments in and the complexity of software. The development process has come to be focused on iterative refinement, agility to meet changing needs and requirements, and ways to bring the user experience more directly to bear. For some types of infrastructure software, the market is experimenting with community-based models in which multiple individuals and companies participate in advancing and maintaining the software.

Software architecture is the first stage of software implementation. It decomposes and partitions a software system into modules that can be dealt with somewhat independently. This modularity has significant business implications, including open interfaces and application programming interfaces (APIs), system integration, and the composition of different applications and infrastructure.

Program distribution and execution options bring to the fore many business and legal issues, including whether the customer is offered source code, whether the software is interpreted or compiled, and whether software is dynamically distributed as it is needed. Increasingly software applications are distributed across multiple computers and organizations.

The management of the entire software life cycle is considered in chapter 5. The major steps in its life cycle include analysis of user needs, development of the software (architecture design, programming, testing, integration, and other functions), provisioning of the equipment and software in the user environment (including user training and often organizational changes), operation and administration of the software, and use. These steps form a value chain, where each step depends upon the successful completion of previous steps and adds value to them. The entire cycle is typically repeated multiple times, and the software supplier faces interesting challenges in managing the process of maintenance and upgrade of coexisting versions of the software while it is in use. Distributed applications often cross organizational boundaries, and this leads to many management challenges and also increases the importance of standardization (discussed in chapter 7). The management of

security and privacy is used as an example of the types of issues that arise in a distributed administrative environment.

The software industry past and present and how it is changing are discussed in chapters 6 and 7. The value chain in software can be partitioned into cooperative and competitive companies; some of the most common ways of doing this are discussed in chapter 6. There is a trend toward offering software as a service over a network, moving responsibility for provisioning and operations from end-user organizations to service providers.

One step in the value chain is software creation, and this is considered in depth in chapter 7. A typical software application draws upon equipment and software from at least a few, if not many, individual firms, leading to issues about business relationships within the software industry as well as between software and end-user firms. Coordination among firms is essential to getting operational software into the hands of the users. Standardization is one way in which software suppliers coordinate themselves, and it is growing in importance as distributed applications move across administrative and organizational boundaries.

Driven largely by the rapid success of the Internet and distributed applications and new requirements for mixing different information media within a single application, the industry organization has been moving from a vertical to a horizontal structure, the latter consonant with an architecture concept called layering. Layering provides a relatively flexible way to partition an infrastructure to provide a wealth of services, with the possibility of adding additional features and capabilities by building layers on top of an existing infrastructure.

There is growing interest in software reuse and component software as a way of improving the productivity of development organizations, addressing complexity, and improving quality. The idea is to assemble applications largely from existing components, which are modules created independently of any particular system context and constructed for multiple uses. To derive the full benefit, component software will require a marketplace for buying and selling components, which would in turn result in marked changes in the organization of the software industry. A related direction is Web services, in which applications are assembled from services made available over a network such as the Internet.

Government plays many roles in the software industry, as considered in chapter 8. A successful industry depends on government-sanctioned and -enforced intellectual property rights, but information and software present many special challenges that remain to be fully explored and settled. Software technologies can help in enforcing intellectual property rights, for example, through copy protection,

although this presents some controversies. Some aspects of the software market are regulated by government, for better or worse. Security and privacy are areas of regulation, both to protect the rights of citizens and to enhance law enforcement and national security. Free speech and civil liberties are areas of controversy, and software plays multiple roles in potentially restricting access to offending materials, and collecting and amassing personal information that may potentially violate individual privacy. Government regulation attempts to insure a competitive software industry and fair business practices through antitrust laws. Government also plays a major role in ensuring an adequate workforce in the software industries through its direct support of education and publicly funded research and through immigration laws.

Some insights and perspectives on software from the economics profession are described in chapter 9. On both the supply and demand sides, software displays a number of characteristics that, while not unique to software, are mixed in unusual ways. Software shares characteristics with many other types of goods and services, including information, services, material goods, and even plans for a manufacturing factory. Insights that are relevant to understanding business strategies and relationships in the software industry include network effects, lock-in, economies of scale, and risk. Software is arguably the most flexible of any economic good in pricing options, and it can even be self-aware and autonomously initiate payment. The economic rationale for infrastructure includes sharing and economies of scale as well as some special characteristics in mitigating congestion.

Some trends in software use and markets that will have a marked effect on the future of the industry are discussed in chapter 10. These include information appliances, nomadic and mobile users and applications, and pervasive computing (software embedded in many everyday material products).

## 1.3 Research and Discussion Issues

In each chapter, some interesting and important issues that deserve further discussion and investigation are listed. These are not comprehensive but rather illustrate the possibilities for further study.

1. If we look at software as an economic good, it has some distinctive differences from other technologies. How does the evolution of the software industry compare to earlier technologies, such as mass transportation, electrification, and radio? Does it share a considerable number of similarities, or show major differences? Did it develop faster or slower?

2. How well do you think we are doing in providing an appropriate education in software at the primary, secondary, and postsecondary levels? What base level of understanding and specific skills should all students possess?

3. Starting with the observation that products of many types become increasingly interactive, in what ways does the design of these products and how they interact with their owners and users change?

4. If you were to compare computers and software to earlier technological advances like mass transportation, electrification, and radio/television, how would you compare the effects on individuals, organizations, and society in general? Has software been more or less important in changing everyday lives?

5. As you are using some of your favorite software applications, think about how your experience relates directly to the programmers who created them. Do you see the personality or culture of the programmer shining through? Or do you feel really isolated from them even when you try to connect? Why?

6. As you are using those same applications, step back and ask how they might be much better. Do you think they are about as good as they can be, or can they be significantly improved? What do we mean by "good"?

7. Consider how the Internet has changed the software applications you use. What very specific observations can you make?

8. As the software industry has matured, for example, evolving from back office functions to individual productivity to serving as the foundation of many organizational processes, consider how the challenges facing the creator of software have changed.

# References


Abraham, T., S. Ahlawat, and S. Ahlawat. 1998. The India option: Perceptions of Indian software solutions. *International Journal of Technology Management* 15 (6/7): 605–621.

Allen, M. W. 1971. History and applications of computers. In *Information, computers, machines, and man*, ed. A. E. Karbowiak and R. M. Huey. New York: Wiley.

Anderson, R. 2001. *Security engineering: A guide to building dependable distributed systems*. New York: Wiley.

Ang, P.-H., and B. Nadarajan. 1996. Censorship and the Internet: A Singapore perspective. *Communications of the ACM* 39 (6): 72–78.

Baker, A. L., and S. H. Zweben. 1979. The use of software science in evaluating modularity concepts. *IEEE Transactions on Software Engineering* TSE-5 (2): 110–120.

Balasubramanyam, V. N., and A. Balasubramanyam. 1997. International trade in services: The case of India's computer software. *World Economy* 20 (6): 829–843.

Baldwin, C. Y., and K. B. Clark. 1997. Managing in an age of modularity. *Harvard Business Review* 75 (5): 84–93.

Bass, L., P. Clements, and R. Kazman. 1998. *Software architecture in practice*. New York: Addison-Wesley.

Beck, K. 1999. *Extreme programming explained: Embrace change*. Reading, Mass.: Addison-Wesley.

Berztiss, A. 1996. *Software methods for business reengineering*. New York: Springer-Verlag.

Boehm, B.W. 1981. *Software engineering economics*. Englewood Cliffs, N.J.: Prentice-Hall.

———. 1984. Software engineering economics. *IEEE Transactions on Software Engineering* TSE-10 (1): 4–21.

———. 1988. A spiral model of software development and enhancement. *IEEE Computer* 21 (5): 61–72.

Boehm, B.W., and P. Bose. 1994. A collaborative spiral software process model based on theory W. In *Proceedings, 3d International IEEE Conference on the Software Process, Reston, Va.*

Boehm, B. W., A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy. 1998. Using the WinWin spiral model: a case study. *IEEE Computer* 31 (7): 33–44.

Boehm, B. W., and K. Sullivan. 1999. Software economics: status and prospects. *Information and Software Technology* 41 (14): 937–946.

———. 2000. Software economics: A roadmap. In *The future of software engineering*, ed. A. Finkelstein. 22d International Conference on Software Engineering.

Bosch, J. 2000. *Design and use of software architectures*. Reading, Mass.: Addison-Wesley.

Brooks, F. Jr. 1975. *The mythical man-month*. Reading, Mass.: Addison-Wesley. Anniversary edition 1995.

Brown, W. J., R. C. Malveau, C. Raphael, W. H. Brown, W. H. McCormick, W. Hays III, and T. J. Mowbray. 1998. *Antipatterns: Refactoring software, architectures, and projects in crisis*. New York: Wiley.

Broy, M., A. Deimel, J. Henn, K. Koskimies, F. Plasil, G. Pomberger, W. Pree, M. Stal, C. Szyperski, and M. Broy. 1998. What characterizes a (software) component? *Software Concepts and Tools* 19 (no. 1): 49–59.

Bulkeley, W. M. 2000. Ozzie to unveil Napster-style networking. *Wall Street Journal Online*, October 24. ⟨http://www.zdnet.com/zdnn/stories/news/0,4586,2644020,00.html⟩.

Cartwright, S. D. 2000. Napster: A business in search of a viable model. *Journal of Business Strategy* 21 (5): 28–32.

Chang, W.-T., and D. Messerschmitt. 1996. Dynamic deployment of peer-to-peer networked applications to existing World-Wide Web browsers. In *Proceedings, Conference on Telecommunications Information Network Architecture (TINA), Heidelberg*.

Chavez, A., C. Tornabene, and G. Wiederhold. 1998. Software component licensing: a primer. *IEEE Software* 15 (5): 47–53.

Church, J., and N. Gandal. 1992. Network effects, software provision, and standardization. *Journal of Industrial Economics* 40 (1): 85–103.

Ciarletta, L. P., and A. A. Dima. 2000. A conceptual model for pervasive computing. In *Proceedings, 29th International Conference on Parallel Computing, Toronto*.

Clark, J. R., and L. S. Levy. 1993. Software economics: An application of price theory to the development of expert systems. *Journal of Applied Business Research* 9 (2): 14–18.

Conway, M. E. 1968. How do committees invent? *Datamation* 14 (4): 28–31, cited in Brooks 1975, *The mythical man-month*, anniversary edition (1995), 111.

Coplien, J. O. 1995. A generative development-process pattern language. In *Pattern Languages of Program Design*, ed. J. Coplien and D. Schmidt. Reading, Mass.: Addison-Wesley.

Covisint establishes corporate entity: Automotive e-business exchange becomes LLC. 2000. ⟨http://www.covisint.com/about/pressroom/pr/covisint_becomes.shtml⟩.

Cox, B. 1996. *Superdistribution: Objects as property on the electronic frontier.* ⟨http://virtualschool.edu/mon/TTEF.html⟩.

Dam, K. W. 1994. The economic underpinnings of patent law. *Journal of Legal Studies* 23 (1, pt. 1): 247–271.

———. 1995. Some economic considerations in the intellectual property protection of software. *Journal of Legal Studies* 24 (2): 321–377.

Dam, K. W., and H. S. Lin. 1996. National cryptography policy for the information age. *Issues in Science and Technology* 12 (4): 33.

———. 1997. Deciphering cryptography policy. *Issues in Science and Technology* 13 (4): 20.

David, P. A. 1990. The dynamo and the computer: An historical perspective on the modern productivity paradox. *American Economic Review* 80 (2): 355–361.

David, P. A., and S. Greenstein. 1990. The economics of compatibility standards: An introduction to recent research. *Economics of Innovation and New Technology* 1 (1/2): 3–41.

Denning, D. E. 1995. Key escrow encryption: the third paradigm. *Computer Security Journal* 11 (1): 43–52.

Denning, D. E., and D. K. Branstad. 1996. A taxonomy for key escrow encryption systems. *Communications of the ACM* 39 (3): 34–39.

Denning, D. E., and M. Smid. 1994. Key escrowing today. *IEEE Communications Magazine* 32 (9): 58–68.

Devanbu, P., P. Fong, and S. Stubblebine. 1998. Techniques for trusted software engineering. In *Proceedings, 20th International Conference on Software Engineering, Kyoto.* ⟨http://seclab.cs.ucdavis.edu/~devanbu/icse98.ps⟩.

ECMA. 1999. *ECMA-262: ECMAScript language specification.* ⟨http://www.ecma.ch/⟩.

———. 2001a. *ECMA-334: C# (C sharp) language specification.* ⟨http://www.ecma.ch/⟩.

———. 2001b. *ECMA-335: Common language infrastructure.* ⟨http://www.ecma.ch/⟩.

Economides, N. 1996. The economics of networks. *International Journal of Industrial Organization* 16 (4): 673–699. ⟨http://www.stern.nyu.edu/networks/top.html⟩.

Ellram, L. M. 1994. A taxonomy of total cost of ownership models. *Journal of Business Logistics* 15 (1): 171–191.

Ellram, L. M., and S. P. Siferd. 1998. Total cost of ownership: A key concept in strategic cost management decisions. *Journal of Business Logistics* 19 (1): 55–84.

Emigh, J. 1999. Total cost of ownership. *Computerworld* 33 (51): 52.

Farrell, J., and C. Shapiro. 1988. Dynamic competition with switching costs. *Rand Journal of Economics* 19 (1): 123–137.

———. 2000. Scale economies and synergies in horizontal merger analysis. ⟨http://www.haas.berkeley.edu/~shapiro/mergers.pdf⟩.

Ferguson, C. H., and C. R. Morris. 1994. *Computer wars: The fall of IBM and the future of global technology.* New York: Times Books.

Firesmith, D. G., and B. Henderson-Sellers. 2001. *The OPEN process framework: An introduction.* New York: Addison-Wesley. ⟨http://www.donald-firesmith.com/⟩.

Fowler, M., and J. Highsmith. 2001. The agile manifesto. *Software Development*, August. ⟨http://www.sdmagazine.com/documents/s=844/sdm0108a/0108.htm⟩.

Frakes, W. B., and P. B. Gandel. 1990. Representing reusable software. *Information and Software Technology* 32 (10): 653–664.

Frank, D. J., R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur, and H. P. Wong. 2001. Device scaling limits of Si MOSFETs and their application dependencies. *Proceedings of the IEEE* 89 (3): 259–288.

Frazier, T. P., and J. W. Bailey. 1996. The costs and benefits of domain-oriented software reuse: Evidence from the STARS demonstration projects. Paper P-3191. Institute for Defense Analysis.

Furse, M. 1998. United States vs. Microsoft: Ill-considered antitrust. *International Review of Law, Computers, and Technology* 12 (1): 99–120.

———. 1999. United States vs. Microsoft: High-tech antitrust. *International Review of Law, Computers, and Technology* 13 (2): 237–253.

Gaffney, J. E. Jr., and T. A. Durek. 1989. Software reuse—Key to enhanced productivity: Some quantitative models. *Information and Software Technology* 31 (5): 258–267.

Garcia, M. 1998. Institutional development in the software industry: Intellectual property protection. Ph.D. diss., University of Southern California, Los Angeles, CA 90089.

Gardner, T. 2001. An introduction to Web services. *Ariadne,* October 2 (no. 29). ⟨http://www.ariadne.ac.uk/issue29/gardner/⟩.

Garone, S., and S. Cusack. 1999. *Components, objects, and development environments: 1999 worldwide markets and trends*. International Data Corporation.

Gaskins, D. 1971. Dynamic limit pricing: Optimal pricing under threat of entry. *Journal of Economic Theory* (2): 306–322.

Gerlach, J. H., and F.-Y. Kuo. 1991. Understanding human-computer interaction for information systems design. *MIS Quarterly* 15 (4): 526–549.

Goertzel, B., and P. Pritchard. 2002. The Internet economy as a complex system. ⟨http://www.goertzel.org/papers/ecommerce.html⟩.

Gray, J., and P. Shenoy. 2000. Rules of thumb in data engineering. In *Proceedings, 16th IEEE International Conference on Data Engineering, San Diego.*

*Great Books Online*. 2002. ⟨http://www.bartleby.com/⟩.

Griffith, D. 1998. Total cost of ownership: The hidden margin-eating monster. *Computer Technology Review* 18 (6): 54–56.

Gulledge, T. R., and W. P. Hutzler, eds. 1993. *Analytical methods in software engineering economics*. New York: Springer-Verlag.

Hartford, K. 2000. Cyberspace with Chinese characteristics. *Current History*, September, 255. ⟨http://www.pollycyber.com/pubs/ch/home.htm⟩.

Heer, D. N., and D. P. Maher. 1995. The heart of the new information appliance. *IEEE Transactions on Consumer Electronics* 41 (3): 869–874.

Heineman, G. T., and W. T. Councill. 2001. *Component-based software engineering: Putting the pieces together.* New York: Addison-Wesley.

Hiles, A. 1993. *Service-level agreements: Managing cost and quality in service relationships*. London: Chapman and Hall.

Hopkins, J. 2000. Component primer. *Communications of the ACM* 43 (10): 27–30.

Howard, J. D. 1997. An analysis of security incidents on the Internet. Ph.D. diss., Carnegie Mellon University, Pittsburgh, PA 15213.
⟨http://www.cert.org/research/JHThesis/ Start.html⟩.

*IBM Systems Journal*. 1999. Vol. 38, no. 4. Special issue on pervasive computing. ⟨http://www.research.ibm.com/journal/sj38-4.html⟩.

International technology roadmap for semiconductors. 2002. Technology Working Groups of International SEMATECH. ⟨http://public.itrs.net/⟩.

Jacobson, I., M. Griss, and P. Jönsson. 1997. *Software reuse: Architecture, process and organization for business success*. Reading, Mass.: Addison-Wesley.

Jung, H.-W., and B. Choi. 1999. Optimization models for quality and cost of modular software systems. *European Journal of Operational Research* 112 (3): 613–619.

Kang, K. C., and L. S. Levy. 1989. Software methodology in the harsh light of economics. *Information and Software Technology* 31 (5): 239–250.

Katz, M. L., and C. Shapiro. 1985. Network externalities, competition, and compatibility. *American Economic Review* 75 (3): 424–440.

———. 1986a. How to license intangible property. *Quarterly Journal of Economics* 101 (3): 567–589.

———. 1986b. Technology adoption in the presence of network externalities. *Journal of Political Economy* 94 (4): 822–841.

———. 1999a. Antitrust in software markets. In *Competition, innovation and the Microsoft monopoly: Antitrust in the digital marketplace*, ed. J. A. Eisenach and T. M. Lenard, 29–81. Proceedings of a Conference of the Progress and Freedom Foundation, Washington, D.C. Boston: Kluwer. ⟨http://www.haas.berkeley.edu/~shapiro/software.pdf⟩.

———. 1999b. Competition policy in the information economy. ⟨http://www.haas.berkeley.edu/~shapiro/software.pdf⟩.

Keliher, M. J. 1980. Computer security and privacy: A systems approach is needed. *Vital Speeches* 46 (21): 662–666.

Kemerer, C. F. 1998. Progress, obstacles, and opportunities in software engineering economics. *Communications of the ACM* 41 (8): 63–66.

Keyes, R. W. 2001. Fundamental limits of silicon technology. *Proceedings of the IEEE* 89 (3): 227–239.

Koch, C. 1998. Service level agreements: put IT in writing. *CIO Magazine*, November 15. ⟨http://www.cio.com/archive/111598_sla.html⟩.

Kovacic, W., and C. Shapiro. 1999. Antitrust policy: A century of economic and legal thinking. ⟨http://www.haas.berkeley.edu/~shapiro/century.pdf⟩.

Kruchten, P. 2000. *The rational unified process: An introduction*. 2d ed. New York: Addison-Wesley.

Langelaar, G. C., I. Setyawan, and R. L. Lagendijk. 2000. Watermarking digital image and video data. A state-of-the-art overview. *IEEE Signal Processing Magazine* 17 (5): 20–46.

Langlois, R. N. 1992. External economies and economic progress: The case of the microcomputer industry. *Business History Review* 66 (1): 1–50.

———. 1999. Modularity in technology, organization, and society. University of Connecticut, Department of Economics, Working Paper 1999-05.

Langlois, R. N., and P. L. Robertson. 1992. Networks and innovation in a modular system: Lessons from the microcomputer and stereo component industries. *Research Policy* 21 (4): 297–313.

Laudon, K. C. and J. P. Laudon. 2001. *Management Information Systems: Managing the Digital Firm*. Englewood Cliffs, N.J.: Prentice Hall.

Lee, P., and M. Leone. 1996. Optimizing ML with run-time code generation. *ACM SIGPLAN Notices* 31 (5): 137–148.

Lehman, M. M., and J. F. Ramil. 2000. Software evolution in the age of component-based software engineering. *IEE Proceedings Software* 147 (6) 249–255.

Lehman, M. M., J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski. 1997. Metrics and laws of software evolution: The nineties view. In *Proceedings, Fourth International Software Metrics Conference, Los Alamitos, Calif.*, 20–32.

Levin, S. I. 1999. Who are we protecting? A critical evaluation of United States encryption technology export controls. *Law and Policy in International Business* 30 (3): 529–552.

Levy, D., and S. Welzer. 1985. System error: How the IBM antitrust suit raised computer prices. *Regulation* 9 (5): 27–30.

Levy, L. S. 1987. *Taming the tiger: Software engineering and software economics*. New York: Springer-Verlag.

Lewis, T. 1997. *Friction-free economy: Strategies for success in a wired world*. New York: Harper Business.

Liebowitz, S., and S. Margolis. 1999. *Winners, losers and Microsoft: Competition and antitrust in high technology*. Oakland, Calif.: Independent Institute.

Lixin, T. 2001. Shifting paradigms with the application service provider model. *IEEE Computer* 34 (10): 32–39.

Luecke, R. W., D. T. Meeting, and R. G. Klingshirn. 1999. New AICPA standards aid accounting for the costs of internal-use software. *Healthcare Financial Management* 53 (5): 40–46.

Lyman, P., and H. R. Varian. 2000. How much information? ⟨http://www.sims.berkeley.edu/ how-much-info⟩.

Macker, J. P., V. D. Park, and M. S. Corson. 2001. Mobile and wireless Internet services: putting the pieces together. *IEEE Communications Magazine* 39 (6): 148–155.

MacKie-Mason, J. K., and H. R. Varian. 1995. Pricing congestible network resources. *IEEE Journal on Selected Areas in Communications* 13 (7): 1141–1149.

Maes, M., T. Kalker, J.-P. Linnartz, J. Talstra, F. G. Depovere, and J. Haitsma. 2000. Digital watermarking for DVD video copy protection. *IEEE Signal Processing Magazine* 17 (5): 47–57.

Makulowich, J. 1999. Pervasive computing: "The next big thing." *Washington Technology Online* 14 (8). ⟨http://www.wtonline.com/vol14_no8/cover/652-1.html⟩.

Markoff, J. 1999a. Growing compatibility issue: Computers and user privacy. *New York Times*, March 3.

———. 1999b. When privacy is more perilous than the lack of it. *New York Times*, April 4.

Marsh, G. 1987. Impacts of the new computer crime law on computer security operations. *CPA Journal* 57 (8): 106–107.

Marshall, A. 1890. *Principles of economics*. Reprinted: Prometheus Books, 1997.

Messerschmitt, D. 1999a. *Networked applications: A guide to the new computing infrastructure*. San Francisco: Morgan Kaufmann.

———. 1999b. The prospects for computing-communications convergence. In *Proceedings, MÜNCHNER KREIS, Conference VISION 21: Perspectives for Information and Communication Technology, Munich*.
⟨http://www.eecs.berkeley.edu/~messer/PAPERS/99/Munich. PDF⟩.

———. 1999c. *Understanding networked applications: A first course*. San Francisco: Morgan Kaufmann.

Messerschmitt, D., and C. Szyperski. 2001. Industrial and economic properties of software: Technology, processes, and value. University of California Berkeley Computer Science Division Technical Report UCB//CSD-01-1130 and Microsoft Corporation Technical Report MSR-TR-2001-11.

Meyers, J. 1993. A short history of the computer. ⟨http://ww.softlord.com/comp/⟩.

Microsoft Corporation. 2000. Universal plug and play device architecture.
⟨http://www.upnp.org/download/UPnPDA10_20000613.htm⟩.

Moore, G. E. 1965. Cramming more components onto integrated circuits. *Electronics* 38 (8): 114–117.

Mowery, D. C., and R. N. Langlois. 1994. Spinning off and spinning on: The federal government role in the development of the U.S. computer software industry. *Research Policy* 25: 947–966.

Mowery, D. C., and R. R. Nelson, eds. 1999. *Sources of industrial leadership: Studies of seven industries*. New York: Cambridge University Press.

Munter, P. 1999. Accounting for software development costs. *CPA Journal* 69 (2): 42–45.

NRC (National Research Council). 1994. *Rights and responsibilities of participants in networked communities*. Washington, D.C.: National Academies Press.

———. 1995. *Evolving the high performance computing and communications initiative to support the nation's information infrastructure*. Washington, D.C.: National Academies Press.

———. 1996. *Cryptography's role in securing the information society*. Washington, D.C.: National Academies Press.

———. 1999a. *Being fluent with information technology.* Washington, D.C.: National Academies Press.

———. 1999b. *Funding a revolution: Government support for computing research*. Washington, D.C.: National Academies Press.

———. 2000a. *The digital dilemma: Intellectual property rights in the digital age*. Washington, D.C.: National Academies Press.

———. 2000b. *Making IT better: Expanding information technology research to meet society's needs*. Washington, D.C.: National Academies Press.

———. 2001. *Building a workforce for the information rconomy*. Washington, D.C.: National Academies Press.

———. 2002. *Technically speaking: Why all Americans should know more about technology*. National Academies Press.

NSF (National Science Foundation). 2001. White paper on an NSF ANIR middleware initiative. CISE Advisory Committee, Subcommittee on the Middleware Infrastructure. ⟨http://www.cise.nsf.gov/anir/mwir_whiteppr.htm#1⟩.

Netpliance. ⟨http://www.netpliance.com/⟩.

Neumann, P. G. 1999. Robust open-source software. *Communications of the ACM* 42 (2): 128.

Nielsen, J. 1993. Noncommand user interfaces. *Communications of the ACM* 36 (4): 83–99. ⟨http://www.useit.com/papers/noncommand.html⟩.

———. 2000. *Designing Web usability: The practice of simplicity*. Indianapolis: New Riders Publishing.

Oberndorf, P. A. 1997. Facilitating component-based software engineering: COTS and open systems. In *Proceedings, Fifth International Symposium on Assessment of Software Tools and Technologies, Pittsburgh*.

Open Source Initiative. ⟨http://www.opensource.org/⟩.

O'Reilly, T. 1999. Lessons from open-source software development. *Communications of the ACM* 42 (4): 32–37.

Palm, Inc. 2002. The philosophy behind the Palm OS. ⟨http://www.palmos.com/platform/ philosophy.html⟩.

Parnas, D. L. 1972. On the criteria for decomposing systems into modules. *Communications of the ACM* 15 (12): 1053–1058.

Petreley, N. 1999. Total cost of ownership reduction may just be another impossible dream. *InfoWorld* 21 (40): 126.

Pettersson, M. 1996. *Complexity and evolution*. New York: Cambridge University Press.

Pfleeger, C. P. 1997. *Security in computing*. 2d ed. Englewood Cliffs, N.J.: Prentice-Hall.

Podilchuk, C. I., and E. J. Delp. 2001. Digital watermarking: algorithms and applications. *IEEE Signal Processing Magazine* 18 (4): 33–46.

Portero-Sánchez, L. M. 1999. Competition in the software industry: The interface between antitrust and intellectual property law. *Communications and Strategies* 35 (3d quarter): 45–79.

Pour, G. 1998. Moving toward the component-based software development approach. In *Proceedings, Conference on Technology of Object-Oriented Languages (TOOLS 27), Beijing*.

Pressman, R. S. 2000. *Software engineering: A practitioner's approach*. 5th ed. New York: McGraw-Hill.

Raymond, E. 1996. *The new hacker's dictionary.* 3d ed. Cambridge, Mass.: MIT Press.

Resnick, P., and H. R. Varian. 1997. Recommender systems. *Communications of the ACM* 40 (3).

Roberts, L. G. 1969. Data processing technology forecast. Advanced Research Projects Agency Internal Memorandum. ⟨http://www.ziplink.net/~lroberts/Forecast69.htm⟩.

———. 2000. Beyond Moore's law: Internet growth trends. *IEEE Computer* 33 (1): 117–119.

Robertson, P. L., and R. N. Langlois. 1995. Innovation, networks, and vertical integration. *Research Policy* 24 (4): 543–562.

Romero, S. 2001. Location devices' use rises, prompting privacy concerns. *New York Times*, March 4.

Royce, W. W. 1970. Managing the development of large software systems. In *Proceedings, IEEE WESCON*.

———. 1990. TRW's Ada process model for incremental development of large software systems. In *Proceedings, International Conference on Software Engineering, Nice, France*.

Saltzer, J. H., D. P. Reed, and D. Clark. 1984. End-to-end arguments in system design. *ACM Transactions on Computer Systems* 2 (4): 277–288. An earlier version appeared in *Proceedings, Second International Conference on Distributed Computing Systems* (April 1981), 509–512.

Samuelson, P. 1999. Why the anticircumvention regulations need revision. *Communications of the ACM* 42 (9): 17–21.

Sánchez, R., and J. T. Mahoney. 1996. Modularity, flexibility, and knowledge management in product and organization design. *Strategic Management Journal* (winter): 1763–1776.

Schaller, R. R. 1997. Moore's law: past, present and future. *IEEE Spectrum* 34 (6): 52–59.

Schattke, R. 1988. Accounting for computer software: The revenue side of the coin. *Journal of Accountancy* 165 (1): 58–70.

Shannon, C. E., and W. Weaver. 1949. *The mathematical theory of communication*. Urbana: University of Illinois Press.

Shapiro, C. 2000. Setting compatibility standards: Cooperation or collusion? ⟨http://www.haas.berkeley.edu/~shapiro/standards.pdf⟩.

———. 2001a. Antitrust limits to patent settlements. ⟨http://www.haas.berkeley.edu/ ~shapiro/settle.pdf⟩.

———. 2001b. Navigating the patent thicket: Cross licenses, patent pools, and standard setting. In *Innovation policy and the economy*, ed. A. Jaffe, J. Lerner, and S. Stern. ⟨http://www.haas.berkeley.edu/~shapiro/thicket.pdf⟩.

Shapiro, C., and H. R. Varian. 1998. Versioning: The smart way to sell information. *Harvard Business Review* 76 (6): 106–114.

———. 1999a. The art of standard wars. *California Management Review* 41 (2): 8–32.

———. 1999b. *Information rules: A strategic guide to the network economy*. Boston: Harvard Business School Press.

Shy, O. 2001. *The economics of network industries*. New York: Cambridge University Press.

Shy, O., and J. Thisse. 1999. A strategic approach to software protection. *Journal of Economics and Management Strategy* 8 (2): 163–190.

Silvestre, J. 1987. Economies and Diseconomies of scale. In *The new Palgrave: A dictionary of economics*, ed. J. Eatwell, M. Milgate, and P. Newman, 80–83. London: Macmillan.

Slaughter, S. A., D. E. Harter, and M. S. Krishnan. 1998. Evaluating the cost of software quality. *Communications of the ACM* 41 (8): 67–73.

Smith, R., B. Meyer, C. Szyperski, and G. Pour. 2000. Component-based development? Refining the blueprint. In *Proceedings, 34th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS 34), Santa Barbara, Calif*.

Software in India: Bangalore bytes? 1996. *Economist*, March 23.

Software Productivity Consortium. 1994. Process Engineering with the Evolutionary Spiral Process Model SPC-93098-CMC, version 01.00.06. Herndon, Va.

Stawlings, W. 1999. *Cryptography and network security: Principles and practice*. 2d ed. New York: Prentice-Hall.

Stokes, D. E. 1997. *Pasteur's quadrant: Basic science and technological innovation*. Washington, D.C.: Brookings Institution Press.

Suganuma, T., T. Ogasawara, M. Takeuchi, T. Yasue, M. Kawahito, K. Ishizaki, H. Komatsu, and T. Nakatani. 2000. Overview of the IBM Java just-in-time compiler. *IBM Systems Journal* 39 (1): 175–193.

Sullivan, J. 1999. Napster: Music is for sharing. *Wired News*, November 1. ⟨http://www.wired.com/news/print/0,1294,32151,00.html⟩.

Sun Microsystems. 1999a. The Java Hotspot performance engine architecture: A white book about Sun's second-generation performance technology. ⟨http://java.sun.com/products/hotspot/whitebook.html⟩.

———. 1999b. Jini technology architectural overview. ⟨http://www.sun.com/jini/whitepapers/architecture.html⟩.

Szyperski, C. 1998. Emerging component software technologies: A strategic comparison. *Software Concepts and Tools* 19 (1): 2–10.

———. 2000. Rethinking our trade and science: From developing components to component-based development. *Modular Programming Languages. Joint Modular Languages Conference, JMLC 2000, Zurich*.

———. 2001. Components and Web services. *Software Development* 9 (8).

———. 2002a. *Component software: Beyond object-oriented programming*. 2d ed. New York: Addison-Wesley.

———. 2002b. Services rendered. *Software Development* 10 (1).

Thebaut, S. M., and V. Y. Shen. 1984. An analytic resource model for large-scale software development. *Information Processing and Management* 20 (1/2): 293–315.

Thompson, C., ed. 1998. Workshop Report. OMG DARPA Workshop on Compositional Software Architectures. ⟨http://www.objs.com/workshops/ws9801/report.html⟩.

Torrisi, S. 1998. *Industrial organization and innovation: An international study of the software industry*. London: Edward Elgar Publishers.

Touretzky, D. S. 2001. Free speech rights for programmers. *Communications of the ACM* 44 (8): 23–25.

———. 2002. Gallery of CSS descramblers. ⟨http://www.cs.cmu.edu/~dst/DeCSS/Gallery⟩.

Traw, C. B. S. 2001. Protecting digital content within the home. *IEEE Computer* 34 (10): 42–47.

Ueda, K. 2001. Synthesis and emergence: Research overview. *Artificial Intelligence in Engineering* 15 (4).

Upton, D. M. 1992. A flexible structure for computer-controlled manufacturing systems. *Manufacturing Review* 5 (1): 58–74. ⟨http://www.people.hbs.edu/dupton/papers/organic/ WorkingPaper.html⟩.

U.S. Department of Justice. 1998. *The United States of America vs. Microsoft Corporation*. Civil Action No. 98-1232. Complaint. May 18. ⟨http://www.usdoj.gov/atr/cases/ f1700/1763.htm⟩.

———. 1999. *The United States of America vs. Microsoft Corporation*, Civil Action No. 98-1232. Findings of Fact. November 5. ⟨http://www.usdoj.gov/atr/cases/f3800/msjudgex.htm⟩.

Usability Professionals Association. ⟨http://www.upassoc.org/⟩.

Vacca, J. 1993. Tapping a gold mine of software assets. *Software Magazine* 13 (16): 57–67.

Varian, H. R. 1992. *Microeconomic analysis*. 3d ed. New York: W.W. Norton.

———. 1993. Economic incentives in software design. *Computational Economics* 6 (3/4): 201–217.

———. 2001. Versioning information goods. In *Internet publishing and beyond: The economics of digital information and intellectual property*, ed. B. Kahin and H. R. Varian. Cambridge, Mass.: MIT Press.

Veryard, R., ed. 1991. *The economics of information systems and software*. Boston: Butterworth-Heinemann.

Wallach, D. S. 2001. Copy protection technology is doomed. *IEEE Computer* 34 (10): 48–49.

Ward, E. 2000. Viral marketing involves serendipity, not planning. *B to B* 85 (10): 26.

Weiser, M. 1991. The computer for the 21st century. *Scientific American* 265 (3): 94–104.

*White book: How DIGITAL FX!32 works*. 2002. ⟨http://www.support.compaq.com/ amt/fx32/fx-white.html⟩.

World Wide Web Consortium. 2002. A little history of the World Wide Web. ⟨http://www.w3.org/History.html⟩.