# 1 Introduction

*[I want] minimum information given with maximum politeness.*

**Jacqueline Kennedy [Bartle]**

Jacqueline Kennedy would likely be dissatisfied if she were alive today. Information is proliferating wildly, and software is partly to blame. Sometimes the presentation of this vast information is none too polite, but hopefully this book is an exception.

The software industry has become critical. 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 non-technical professions such as management, economics, policy, and law. Obstacles 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 but 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 inherent (and sometimes almost accidental) nature of today's software technologies.

This book seeks to communicate to a broad audience, including non-technical as well as technical professions, the related characteristics of software technology, the business of software-creation, and the context within which software is deployed, operated and used. Given the growing importance of technology 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 is 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 described 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 impact 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 industries.

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, it is risky to invest in software creation, but unlike writing a novel it is essential to collaborate with the eventual users in defining its features. Like an organizational hierarchy, software applications

are often essential to running a business, but unlike an organization, software is often designed by outside vendors with (unfortunately) limited ability to adapt to special or changing needs. Although software is valued for what it does—like many material goods and even human employees--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' in Chapter 4), often supplied by different vendors and with different ownership. Even the term "ownership" has somewhat different connotations from material goods, because it is based on intellectual property laws rather than title and physical possession. Like 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 a copyright, but unlike information it can also incorporate patented inventions.

These examples, and others developed in the sequel, suggest that the software industry—as well as interested participants like the end-user, service provider, and regulatory authorities—confront 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 inter-working 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*

Although software is popping up everywhere, one of its primary impacts 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 is increasingly 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 is 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 the pattern of discourse and communication, and software today joins mass

transportation as a technology having profound impact 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 they write. 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. There is 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 as an important enabler of individuals, groups of individuals, organizations, communities, and society as a whole. We have to consider software as a human artifact or tool, one that can be almost infinitely flexible and that can bring greater benefits if only it can 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, but it is irrevocably changing the ways in which software is sold and involving 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 always 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 are a result 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.

Increasingly 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 connection to the reader. 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 software enabling good tools and rich graphics and sound, application programming increasingly assumes the flavor of an artistic rather than 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 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—a perspective advocated in Section 1.1.5—one great challenge is that responsible actions must be pre-defined 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; as the humans can be counted on to react intelligently to exceptional circumstances.

Although it wasn't always as true, post-Internet the software industry faces a monumental challenge in industry coordination. Infrastructure from many vendors has to work together, and applications have to 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 impacts. The potential deleterious impacts 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 impact 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 professions involved in software creation (Chapter 4) start (hopefully) with a good understanding of what users want to accomplish and end with working software code. Once the software has been created, *managers* (Chapter 5) must deal with a number of issues like acquiring the necessary infrastructure to run the software, the organization of 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 impacts of information technology, and resolving various conflicts among the industry participants. *Economists* (Chapter 9) offer many useful insights into the workings of the marketplace in software. The book concludes with a look into the future (Chapter 10).

There are many issues affecting more than one of these perspectives. In each case, we locate that issue in the chapter that seems most relevant, but also take care to relate it to the other perspectives as well.

A greater appreciation for the wealth of issues surrounding software and the interdependence of these issues 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 the network. Information is valued for how it teaches or influences us, while 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 (including the 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 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, and collaboration and coordination.

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, type of organization, or organizational function. Sociotechnical applications, one growth category, integrate 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 processes of capturing user needs, rethinking human organizations and the associated features and capabilities of software applications is increasingly an important design activity separate from the implementation of the necessary software.

A number of factors contribute value to the user, including specific features and capabilities, impact on an organization, the amount of usage, quality, performance, usability, and 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 a highly principled process due to the increasing size of investments in and complexity of software. These processes are increasingly 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 to modules that can be dealt with somewhat independently. This modularity has significant business implications, including open interfaces and API's, 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 the ability to distribute software dynamically as it is needed. Increasingly software applications are distributed across multiple computers and organizations.

The management of the entire software lifecycle is considered in Chapter 5. The major steps in a lifecycle include analysis of user needs, development of the software (including 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 the coexisting version of the software while it is in use. Distributed applications often cross organizational boundaries, and this leads to many interesting 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 is discussed in Chapter 6 and 7. The value chain in software can be partitioned into cooperative and competitive companies in many ways, and some of the most common and emerging ways are discussed in Chapter 6. There is a trend toward offering software as a service over the network, moving provisioning and operations from the end user organization to become the responsibility of a service provider.

One step in the value chain is software creation, and this is considered in more 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 many issues in the 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 increasing 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 the Internet.

Governments play 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 many challenges and 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 but also to enhance law enforcement and national security. Free speech and civil liberties are an area 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 insuring 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 from the economics profession on software are described in Chapter 9. On both the supply and demand side software displays a number of characteristics which, while not unique to software, and 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 goods in pricing options, and it can even be self-aware and autonomously initiate payment when it is used. The economic rationale for infrastructure includes not only sharing and economies of scale, including some special characteristics in mitigating congestion.

Some trends in software usage and markets that will have a marked impact 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 or most 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. Doubtless 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 considerable characteristics, or show major differences?
2. How well do you think we are doing in providing an appropriate education in software at the primary, secondary, and post-secondary 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 impact 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.