






Industry Profiles Collection
Donald N. Stengel, *Editor*

A Profile of the Software Industry

Emergence, Ascendance, Risks, and Rewards

Sandra A. Slaughter

businessexpert Press
www.businessexpertpress.com

A Profile of the Software Industry

A Profile of the Software Industry

Emergence, Ascendance, Risks, and Rewards

Sandra A. Slaughter

businessexpert Press

A Profile of the Software Industry: Emergence, Ascendance, Risks, and Rewards
Copyright © Business Expert Press, LLC, 2014.
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopy, recording, or any other except for brief quotations, not to exceed 400 words, without the prior permission of the publisher.

First published in 2014 by
Business Expert Press, LLC
222 East 46th Street, New York, NY 10017
www.businessexpertpress.com

ISBN-13: 978-1-60649-654-1 (paperback)
ISBN-13: 978-1-60649-655-8 (e-book)

Business Expert Press Industry Profiles Collection

Collection ISSN: 2331-0065 (print)
Collection ISSN: 2331-0073 (electronic)

Cover and interior design by Exeter Premedia Services Private Ltd., Chennai, India

First edition: 2014

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America.

Abstract

Software plays a critical role in today's global information economy. It runs the computers, networks, and devices that enable countless products and services. Software varies in size from vast enterprise and communications systems like SAP's enterprise resource planning system with hundreds of millions of lines of code to tiny apps like Angry Birds that run on mobile phones. Companies in the software industry produce and sell software products and related services. The industry is intensely competitive and has undergone a dramatic transformation from its roots in a handful of computer hardware mainframe producers in the 1960s to numerous large and small software companies today.

Understanding how the software industry works and how the industry is evolving are important. Since software runs the computers and networks that support the flow of information in the global economy, the software industry also affects companies in all other industries that use these products and services for their own competitive advantage.

This book offers a profile of the software industry and the companies in the industry. It describes the primary products and services produced in the industry; reviews the history of the industry; explains how the industry is structured; discusses its economics and competitive environment; and examines important trends and issues including globalization, workforce, regulation, and the emergence of new software business models.

Keywords

computers, information technology management, intellectual property, network effects, offshoring, open-source software, outsourcing, software, software as a service, software business model, software copyright, software development, software ecosystems, software industry, software patent, software piracy, software platforms, software product, software security, software standards

Contents

Acknowledgments ix
Preface xi

Chapter 1 Software Basics 1
Chapter 2 History of the Software Industry 29
Chapter 3 Structure and Competitive Dynamics of the Software Industry 53
Chapter 4 Globalization of the Software Industry 79
Chapter 5 Occupations and Workforce Issues and Trends in the Software Industry 127
Chapter 6 Regulation in the Software Industry 155
Chapter 7 Challenges, Opportunities, and Trends in the Software Industry 183

Suggestions for Further Reading 209
Glossary of Key Terms 217
Notes 227
References 245
Index 269

Acknowledgments

This book is dedicated to my husband, Ron, who is my strictest critic, but my staunchest support, my very best friend, and soul mate. It is also dedicated to my Mom and Dad, who are with me in spirit always. Finally, it is dedicated to Lila, Sammie, Lucy, Baxter, Bonnie, Buford, and Peaches, who have always loved me, no matter what!

My sincere thanks to Chris F. Kemerer and Michael D. Frutiger for their feedback on earlier drafts of this book. Their comments and suggestions have been invaluable. I also thank Rajiv D. Banker and Gordon B. Davis, my PhD co-advisors, for encouraging me to study the software industry and for their wise counsel and advice throughout my academic career.

Preface

Introduction

Welcome to the software industry. Software is a place where great ideas thrive, but so do angry birds and super monsters and worms, viruses, and spam zombies. Vast fortunes have been won (and lost) in the industry.

From its conceptual origins in the 19th century England and early manifestations in humble Silicon Valley garages, the software industry has grown to a behemoth that generates more than a half trillion dollars in revenues. Companies in the industry produce and sell software products and services. As an industry, the software industry has competitive dynamics that make it truly unique. Entry and exit barriers are low, marginal costs of production are minimal, and product innovation occurs rapidly and disruptively. It is an industry where a 12-year-old can create a popular software app for a mobile phone,[1] a college dropout can become a billionaire, and a developing country can become an economic powerhouse. On the other hand, software is also an industry where companies can spend more on litigating over intellectual property than on creating the intellectual property and where products can be used to attack individuals and organizations or to launch cyber warfare.

Software is called a place where "dreams are planted but nightmares harvested."[2] Although the software industry is a powerful wealth generator, producing billions in revenues, countless new companies, numerous billionaires and millionaires, and innovative business models, there is a high rate of failure. Companies' competencies and strategies are critical for competitive advantage and survival.

Consider the fortunes of McAfee, Inc. and its founder, John McAfee. In 1987, John McAfee (a former software engineer at NASA, UNIVAC, Xerox, Computer Sciences Corporation, and Lockheed) founded McAfee Associates, a software company that produced software antivirus and security products. Acquired by Intel in 2011 for $7.7 billion,[3] the company today is one of the largest software antivirus and security companies in

the world. It produces a wide range of software security products, such as antivirus software, firewalls, malware removal, software security products for cloud computing providers, data and network protection software, and parental software to monitor Internet use. In 2013, the company employed more than 7,600 and earned more than $2 billion in revenues.[4]

However, both McAfee (the founder) and McAfee (the company) have had numerous ups and downs over the course of their respective histories. With respect to the company, in 2006, the Securities and Exchange Commission filed a suit against McAfee, Inc. for accounting fraud (inflating revenues to investors from 1998 to 2000). The company paid a $50-million fine and settled a class action lawsuit brought by ex-employees to guarantee the value of their share options.[5] Subsequent problems have involved technical issues such as an erroneous virus definition file update by McAfee that caused millions of Windows XP computers worldwide to go offline; hackers identifying serious vulnerabilities in McAfee security software; and McAfee's home security software suffering outages and problems.[6]

With respect to its founder, after selling his stake in McAfee, Inc. in 1996, John McAfee became a multimillionaire. However, his personal fortune declined from $100 million to $4 million by 2009. In 2012, he was arrested (but released) for unlicensed drug manufacturing and possession of an unlicensed weapon in Belize.[7] He was later interrogated as a "person of interest" in the murder of his neighbor in Belize. He fled to Guatemala, seeking asylum, but his bid was denied. While awaiting deportation back to Belize he reportedly faked two heart attacks to delay his deportation, and he was ultimately sent back to the United States. He moved to Portland, Oregon, with his new 30-year-old wife (a former stripper) and is working on a biography.[8] Movies are being made about his life. He has been in the news recently on a number of fronts. He proposed a new device—called "Decentral"—that would allow Internet users to be anonymous and impervious to government surveillance and is working on putting together a start-up company to produce it.[9] And, he spurned a request from a U.S. House of Representatives committee to remedy HealthCare.gov because it "has no interest in fixing anything."[10]

McAfee is an example of the types of interesting companies and characters that populate the software industry.

This book will introduce you to the software industry. If you have an interest in the industry, but don't know much about it and you need a good book to read on a plane or train or in an afternoon, this is the book for you.

Even if you don't have an interest in the industry, you may still want to read this book. Why? Take a look around you … software is everywhere. Almost every man-made thing you see or interact with is powered by software or was made using software or both. Software helps stop and start your car, fly a plane, heat and air condition your home, power the stock market, entertain you on your cell phone, and run the businesses around you.

Understanding who creates software products and services and how they produce and market them is a key to understanding the software industry.

How This Book Is Organized

This book tells the story of a dynamic and complex industry. The book provides a profile of the software industry, describes how the industry works, and identifies the main players in the industry. Each chapter introduces a particular topic of importance to the industry and ends with a brief summary of key takeaways.

Chapter 1 explains the basics of software—what it is and how it is created. It also defines the two primary types of software that define the software industry. Finally, it distinguishes between the kinds of firms that are included in the industry (and part of this book) and the kinds of firms that are not. After describing the key building blocks of the software industry, the book then turns to the roots of the industry in Chapter 2.

Chapter 2 describes the history of the software industry. It goes back more than 150 years to the first concepts of software and then moves to the start of the industry in the 1960s. The history of the software industry is intertwined with that of the computer hardware industry and the early days in the 1970s are often referred to as the mainframe era. The introduction of personal computers in the 1980s dramatically transformed the software industry, and the chapter tells the story of the intense competition for dominance in that era and the emergence of key firms such

as Microsoft, Oracle, and SAP. The Internet came into prominence in the 1990s and 2000s and has dramatically changed the software industry again. Chapter 2 describes the innovative software products, services, pricing, and distribution methods enabled by the Internet.

After providing a historical perspective of the software industry in Chapter 2, Chapter 3 outlines the structure and competitive dynamics of the software industry today. It defines the different sectors and segments of the industry and profiles the major companies in the industry. The chapter then describes the software value chain: the different activities that are needed to produce and market software products. It concludes by identifying the unique economics of software and describes how these economics influence competitive dynamics in the industry.

Chapter 3 describes the market structure of the industry on a national scale, whereas Chapter 4 provides a global perspective of the software industry. Although the United States has historically dominated the industry, that situation is changing. In the 1990s software outsourcing and offshoring emerged and stimulated the growth of the industry around the globe. Vibrant software products and services industries have emerged in Europe, Asia, Australia, Africa, and Latin America. The chapter provides an overview of the software industry and profiles companies in key countries in these regions.

After describing the structure of the software industry in Chapters 3 and 4, Chapters 5 through 7 explore important trends, opportunities, and issues in the software industry including those relating to workforce, regulation, security, and the emergence of new software business models.

Chapter 5 discusses occupations in the software industry and workforce issues and trends. In contrast to other industries that manufacture a physical product, in the software industry, labor is the primary factor of production. The industry depends critically on human talent. This chapter identifies the major occupations in the software industry. It then discusses important workforce issues such as labor shortages, lack of gender and ethnic diversity in the industry, the global workforce, debates about the impact of the global workforce on the U.S. workforce, and other software workforce trends.

Chapter 6 examines regulation in the software industry. Given the unique economics of the industry, it is prone to monopolistic behavior.

This chapter describes the major regulations governing the industry. It also identifies the different types of intellectual property protections available to companies in the industry. The chapter concludes by considering problems with infringement of intellectual property in the form of software piracy and by examining the intensification of battles over software intellectual property being waged by companies in the industry.

Chapter 7 identifies key trends in the software industry. The chapter starts with a discussion of software security and privacy issues that are of increasing concern given the frequency and scope of attacks and the vulnerability and ubiquity of software. The chapter then reveals emerging software business models and innovative pricing and delivery schemes such as software as a service, open-source software, service-oriented architectures, and software platform ecosystems. The chapter concludes by envisioning the possible future of the software industry. It considers the possibility that companies in the software industry may be shifting to producing services rather than products and projects what that shift may mean for the structure and competitive dynamics in the industry.

For those who would like to learn more about the software industry, resources for further study on the software industry as well as a glossary of key terms and a complete set of references are provided.

CHAPTER 1

Software Basics

This book focuses on companies that produce and sell software—either for hire or for the mass market. Before you can truly understand the software industry and how it works, you really need to understand its key product: *software*.

This chapter is a bit technical, but important. It describes what *software* is and how it is created. It then describes the basic types of software products and related services that companies in the software industry produce and describes what types of companies the software industry includes (as the focus of this book) and what types of companies it does not.

What Is Software

Computer software or *software* refers to computer instructions that control the functioning of computer hardware and direct its operations. Software includes two types of instructions: machine instructions (the binary code that turns certain electronic pulses on and off to communicate with the computer processor) and source code (more human-understandable instructions).

Since a computer can only understand machine instructions, software source code must be translated into machine code. This is accomplished using something called a *compiler*. For a hypothetical piece of code, Figure 1.1 visually illustrates the process by which a compiler translates human-understandable software source code into machine-understandable object code that can be executed by the computer processor.

A group or sequence of instructions is called a software *program*, which is written to perform a specific task with a computer. For example, a program could look up a customer number in a database to find the customer's past purchases. Another program could simply obtain and report

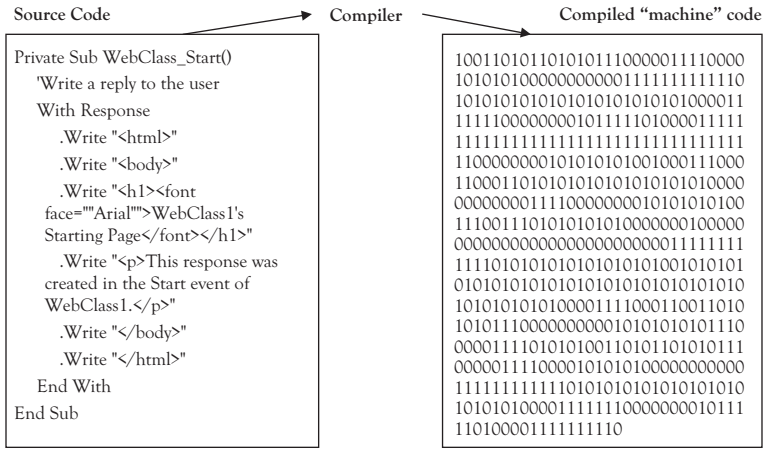


Figure 1.1 Translation of software source code into machine code by a compiler

Source: Freetutes.com (2011).
Note: The compiled machine code is not directly obtained from this code—it is illustrative only.

the current date and time. A grouping of related or connected software programs is called a *software system*. For example, a customer relationship management system includes programs to track, look up, and analyze customer purchasing history.

Software programs are typically written by computer programmers using programming languages, utilities, and other tools. A programming language is a specific set of notations used to write a software program. Examples of programming languages include Assembly, Java, COBOL, FORTRAN, and C++.

Programming languages are often described in terms of *levels* or *generations*.[1] Early or first-generation programming languages in the 1950s used machine language instructions and could only be executed on particular computers. The next- or second-generation programming languages were still machine specific, but were *mnemonic* and were known as assembly languages or *assembler*. Figure 1.2 shows an example of an assembly software code snippet that creates a clock to dynamically obtain and show the current time.[2] At the end of the 1950s, the language ALGOL was introduced and has influenced many other programming languages (such as Pascal, C, and Python).

```
Clock

CGROUP      GROUP VECTOR,CODESEG
VECTOR      SEGMENT AT 0H
            DB 6CH DUP(?)          ;FILLER
TIME_LO DW ?                       ;DOS TIME
TIME_HI DW ?                       ;DOS TIME
VEC_IP DW                          ;CLOCK UPDATE VECTOR IP
VEC_CS DW                          ;CLOCK UPDATE VECTOR CS
VECTOR      ENDS

CODESEG SEGMENT PARA
            ASSUME CS:CODESEG,DS:CGROUP
            ORG 100H
CLK         PROC FAR
            JMP SETUP              ;ATTACH TO DOS
INTRPT      LABEL DWORD
INT_IP      DW 0                   ;OLD UPDATE VECTOR IP
INT_CS      DW 0                   ;OLD UPDATE VECROR CS
TICKS       DW 0                   ;TICK COUNTER
SCR_OFF DB 0,0                     ;SCREEN OFFSET IN BUFFER
CRT_PORT DW 0                      ;SCREEN STATUS PORT
flag        db 0
TIME        DB 8 DUP(':',0BH)      ;TIME SAVE AREA
CLK_INT LABEL NEAR
            PUSH AX                ;SAVE REGISTERS
            PUSH CX
            PUSH DI
            PUSH SI
            PUSH DS
            PUSH ES
            PUSHF                  ;AND FLAGS
            CALL CS:[INTRPT]       ;DO OLD UPDATE INTERRUPT
            MOV CX,0040H           ;GET SEGMENT OF DOS TABLE
            MOV DS,CX              ;PUT IN DS
            MOV CX,CS:TICKS        ;GET TICK COUNT
            INC CX                 ;INCREMENT IT
            CMP CX,20              ;01F4H    ;HAS A MINUTE GONE BY?
            JB NO_MINUTE           ;NO, MOVE ON
            CALL UPDATE            ;YES, UPDATE CLOCK AND
            MOV CX,0               ; RESET TICK COUNTER
```

Figure 1.2 Example of assembly code that obtains and shows the current time

Source: Happy Codings (2013a).

```
NO_MINUTE:
        MOV CS:TICKS,CX     ;SAVE UPDATED TICK COUNT
        MOV CX,0B000H       ;GET VIDEO SEGMENT
        MOV ES,CX           ;PUT IN ES
        MOV DX,CS:CRT_PORT      ;GET CRT STATUS PORT ADDR
        MOV DI,WORD PTR CS:SCR_OFF ;GET SCREEN BUFFER OFFSET
        LEA SI,CS:TIME      ;GET DOS TIME
        MOV CX,16           ;SET UP TO MOVE 10 BYTES
        CLI                 ;DISABLE OTHER INTERRUPTS
WAIT1:  IN AL,DX            ;READ CRT STATUS
        TEST AL,1           ;CHECK FOR VERTICAL RETRACE
        JNZ WAIT1           ;WAIT FOR RETRACE LOW
        MOV AH,CS:[SI]      ;GET FIRST BYTE TO MOVE
WAIT2:  IN AL,DX            ;GET CRT STATUS
        TEST AL,1           ;CHECK FOR VERTICAL RETRACE
        JZ WAIT2            ;WAIT FOR RETRACE HIGH
        MOV ES:[DI],AH      ;MOVE BYTE TO SCREEN
        INC DI              ;INCREMENT INDEX
        INC SI
        LOOP WAIT1          ;MOVE NEXT BYTE
        STI                 ;ENABLE INTERRUPTS
        POP ES              ;RESTORE REGISTERS
        POP DS
        POP SI
        POP DI
        POP CX
        POP AX
        IRET                ;RETURN FROM INTERRUPT
CLK     ENDP
UPDATE  PROC NEAR
        PUSH AX             ;SAVE REGISTERS
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH DS
        MOV AX,0040H        ;GET ADDRESS OF DOS TABLE
        MOV DS,AX           ;PUT IN DS
        MOV AX,TIME_HI      ;GET HIGH BYTE OF DOS TIME
        mov flag,0          ;am flag
HOUR:   CMP AX,0CH          ;CONVERT TO HOURS
        JLE H1
        mov flag,1          ;set to pm
```

Figure 1.2 (continued)

```
        SUB AX,0CH
        JMP HOUR
H1:     AAM                 ;CONVERT TO ASCII
        ADD AX,3030H
        LEA BX,CS:TIME      ;GET ADDRESS OF TIME AREA
        MOV CS:[BX],AH      ;SAVE HOURS FIRST DIGIT
        MOV CS:[BX+2],AL    ;SAVE HOURS SECOND DIGIT
        MOV AX,TIME_LO      ;GET DOS TIME LOW BYTE
        MOV CX,8H           ;CONVERT TO MINUTES
        SHR AX,CL
        MOV DX,3CH
        MUL DL
        SHR AX,CL
        AAM                 ;CONVERT TO ASCII
        ADD AX,3030H
        MOV CS:[BX+6],AH    ;SAVE MINUTES FIRST DIGIT
        MOV CS:[BX+8],AL    ;SAVE MINUTES SECOND DIGIT
        MOV BYTE PTR CS:[BX+12],'A'
        CMP FLAG,0          ;IS IT AM?
        JZ GOAHEAD
        MOV BYTE PTR CS:[BX+12],'P'
GOAHEAD:
        MOV BYTE PTR CS:[BX+14],'M'
        POP DS              ;RESTORE REGISTERS
        POP DX
        POP CX
        POP BX
        POP AX
        RET
UPDATE  ENDP
SETUP:  MOV AX,0            ;GET ADDRESS OF VECTOR TABLE
        MOV DS,AX           ;PUT IN DS
        CLI                 ;DISABLE FURTHER INTERRUPTS
        MOV AX,[VEC_IP]     ;GET ADDRESS OF OLD UPDATE IP
        MOV CS:[INT_IP],AX  ;SAVE IT
        MOV AX,[VEC_CS]     ;GET ADDRESS OF OLD UPDATE CS
        MOV CS:[INT_CS],AX  ;SAVE IT
        MOV VEC_IP,OFFSET CLK_INT ;PUT ADDRESS OF CLK IN
        VECTOR IP
        MOV VEC_CS,CS       ;PUT CS OF CLK IN VECTOR CS
        STI                 ;ENABLE INTERRUPTS
        MOV AH,0FH          ;READ VIDEO STATUS
```

Figure 1.2 (continued)

```
        INT 10H
        SUB AH,8                ;SUBTRACT 8 CHAR TIME FROM NCOLS
        SHL AH,1                ;MULTIPLY BY 2 FOR ATTRIBUTE
        MOV CS:SCR_OFF,AH       ;SAVE SCREEN TIME LOCATION
        MOV WORD PTR CS:CRT_PORT,03BAH ;SAVE MONO STATUS
        PORT ADDR
        TEST AL,4               ;CHECK FOR COLOR MONITOR
        JNZ MONO                ;IF MONO, MOVE ON
        ADD WORD PTR CS:SCR_OFF,8000H ;ADD COLOR OFFSET TO
        TIME OFFSET
        MOV WORD PTR CS:CRT_PORT,03DAH ;SAVE COLOR STATUS
        PORT ADDR
MONO:   CALL UPDATE                 ;DO FIRST UPDATE & PRINT TIME
        MOV DX,OFFSET SETUP         ;GET END ADDRESS OF NEW
        INTERRUPT
        INT 27H                 ;TERMINATE AND REMAIN RESIDENT
        DB 117 DUP(0)           ;FILLER
CODESEG ENDS
        END CLK
```

Figure 1.2 (continued)

Third-generation programming languages (3GLs) were more *programmer friendly*, more abstract, and with instructions that were more understandable by humans. Examples of prominent 3GLs include FORTRAN, LISP, and COBOL. 3GLs were *portable* and could be implemented similarly on different types of computers that did not have the same machine code. Today, updated versions of these 3GLs are still used. Figure 1.3 shows an example of a COBOL software code snippet in which the current date and time are obtained and displayed.[3]

Subsequent languages (such as FOCUS and FOXPRO) called fourth- or fifth-generation programming languages (4GLs or 5GLs) were designed to reduce programming effort, the time it takes to develop software, and the cost of software development, by allowing such features as automatic code generation from a few higher level constructs.

The rise of the Internet has also inspired the development or refinement of programming languages such as Java and C++. Figure 1.4 provides an example of a software code snippet written in Java to display the current date and time.[4]

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.dat.

ENVIRONMENT DIVISION.
  SOURCE-COMPUTER. IBM-PC.
  OBJECT-COMPUTER. IBM-PC.

DATA DIVISION.
  WORKING-STORAGE SECTION.
  01 cr-date1.
  05 yr pic 99.
  05 FILLER PIC X VALUE "/".
  05 mnth pic 99.
  05 FILLER PIC X VALUE "/".
  05 date1 pic 99 value 10.

  01 cr-date2.
  05 yr1 pic 99.
  05 mnth1 pic 99.
  05 date11 pic 99 value 10.

  77 jul-dy pic 9(6).
  01 tm.
  05 hh1 pic 99.
  05 mm1 pic 9.
  05 ss1 pic 99.
  05 fs1 pic 9.
  01 tm1 pic 9(10).
  77 dy-of-wk pic 9.

PROCEDURE DIVISION.
  ACCEPT cr-date2 FROM DATE.
  MOVE yr1 TO yr.
  MOVE mnth1 TO mnth.
  MOVE date11 TO date1.

DISPLAY cr-date1.
  DISPLAY "YEAR:" yr.
  DISPLAY "month:" mnth.
  DISPLAY "date:" date1.
```

Figure 1.3 Example of COBOL code that obtains and shows the current date and time

Source: Daily Free Code (2012)

```
        DISPLAY cr-date2.
        DISPLAY "YEAR:" yr1.
        DISPLAY "month:" mnth1.
        DISPLAY "date:" date11.

        ACCEPT jul-dy FROM DAY.

        ACCEPT tm FROM TIME.
        ACCEPT tm1 FROM TIME.
        ACCEPT dy-of-wk FROM DAY-OF-WEEK.

        DISPLAY "julian" jul-dy.
        DISPLAY "time" tm.
        DISPLAY "HH" hh1.
        DISPLAY "MM" mm1.
        DISPLAY "SS" ss1.
        DISPLAY "TM1" tm1.
        DISPLAY " day of week" dy-of-wk.

      STOP RUN.
```

Figure 1.3 (continued)

```
GetTheCurrentTime

package com.ack.j2se.date;

import java.util.Calendar;
import java.util.Date;

public class GetTheCurrentTime {
public static void main( String[] args ) {
            // one way
            long currentTimeInMillis = System.currentTimeMillis();
            Date today = new Date( currentTimeInMillis );
            System.out.println( today );

            // another way
            Calendar cal = Calendar.getInstance();
            today = cal.getTime();
            System.out.println( today );  }}   }
```

Figure 1.4 Example of Java code to obtain and show the current date and time

Source: Happy Codings (2013a).

It is important to understand programming languages for several reasons. One is that a programming language provides a standard way of describing the functions performed by the computer in a way that can be read, understood, and updated by other programmers. Another reason it is important to understand programming languages is that the effort (cost) to create or maintain software varies significantly by the type of programming language used to write the software code.[5] The earlier programming languages, especially those that are more machine specific, are very difficult and expensive to code.

Table 1.1 lists selected programming languages along with their estimated productivity rates (hours required to produce the same unit of functionality). The productivity rates are provided for a *function point*—a unit of measurement that expresses the amount of functionality provided to a user by the software.[6] Clearly, there are considerable differences in the effort to code in different programming languages. Thus, programming language is a critical factor determining the development and lifecycle cost for a software system.

To put things into perspective, it might help to understand just how large software systems are today. Since some of the first software systems were used to perform repetitive calculations in World War II, software systems have become increasingly complex.

Table 1.1 Effort (hours) to code the same function in selected programming languages

Language	Lines of code for one function point	Hours to create one function point
Assembly	340	79
COBOL	110	27
Pascal	40	10
C	130	32
C++	50	12
Java	55	13
Visual Basic	30	7

Sources: Numbers in this table were created based upon Jones (1996); Boehm (1981).
Note: Capers Jones (1996) provides the lines of code for different programming languages corresponding to a function point. Boehm's (1981) COCOMO model for software effort estimation provides a formula to convert software lines of code to *man-months* of effort. Man months were converted into hours by multiplying by 22 work days and 8 hours per day.

For example, the first edition of the Unix operating system released in 1971 had 4,200 lines of code,[7] but the MS Windows XP operating system released in 2001 has more than 45 million lines of code.[8] The code for Visicalc (the first electronic spreadsheet program introduced in 1981) could fit in 27,520 bytes of storage,[9] whereas the code for the latest version of Microsoft's Excel requires orders of magnitude more storage. The *flame* computer virus that attacked Iranian computers in 2012 contained more than 250,000 lines of LUA code[10]—the same programming language used to create the computer game called *Angry Birds*. LUA is a Portuguese word that means "moon"; the programming language was created by developers in Brazil. SAP's enterprise resource system is believed to have 250 million lines of code.[11] A typical new automobile averages 100 million lines of code.[12]

Undoubtedly, today's software systems such as business-to-business electronic procurement systems are increasingly sophisticated and must meet more demanding requirements than in the past.

How Software Is Created

The practice of developing software has also undergone major transformations as the nature of software systems has evolved. Originally, computer programs were written by scratch, ad hoc, and the early programmers had to use paper tape or punch cards to load the code onto the computer. Today, more sophisticated approaches and tools for developing software have emerged.

Developing software involves gathering requirements, analyzing and designing the system based on the requirements, creating the software source code for the system, testing the system to make sure it works, deploying the system, and supporting and maintaining it.

The first step in developing a software system involves collecting requirements. Requirements refer to the features and functions desired by current or potential customers. For example, the ability to compute an economic order quantity could be a functional requirement for an inventory management system. The ability to compute estimated sales tax could be a functional requirement for an order entry system. Vivid

animation and quick response time could be technical requirements for a video game. Requirements can be gathered in a variety of ways, such as via customer focus groups, interviews, surveys, and observation. Sometimes, software developers create scenarios or user stories (use cases) to document requirements.

The software system's features, functions, and *look and feel* are designed based on its requirements. Software analysis and design involves creating rules and definitions (also called *specifications*) for how a software system should function. Software designs can be represented in abstract diagrams and models. An example of a software design diagram for a hypothetical online shopping system is shown in Figure 1.5. The diagram is called a

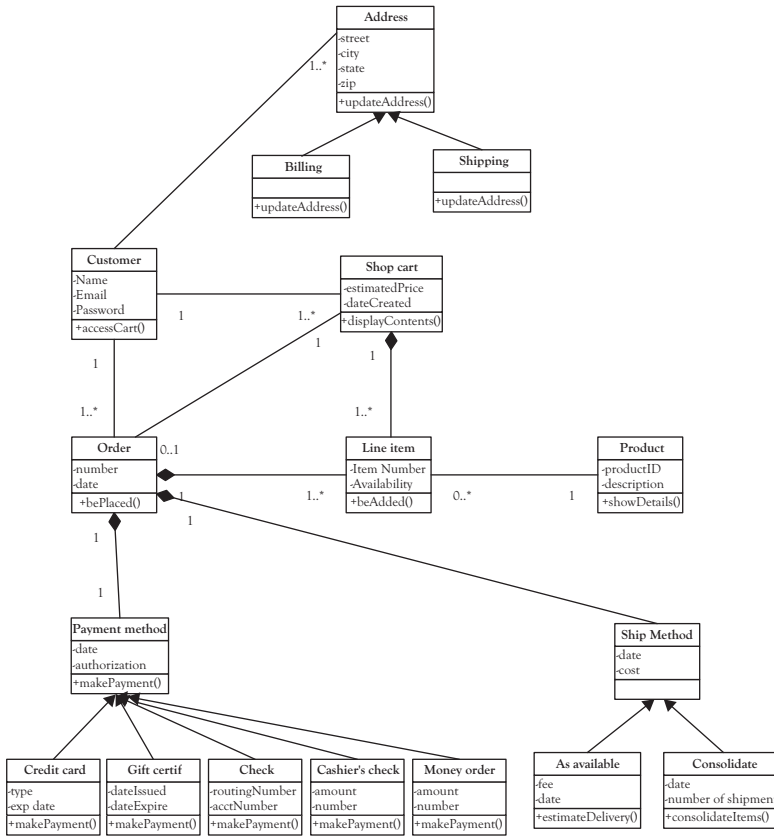


Figure 1.5 A software design diagram for a hypothetical online shopping system

class diagram, and it visually depicts the key classes or objects in the system such as the *shopping cart*, the *order*, the *item*, and the *payment method*.

Based on the analysis and design of requirements, the software code is created. Software programming involves writing the source code or computer instructions for a software system. As noted previously, in the early days of computing, programs were written by hand. Today, some programs are still written by hand, but there are also many tools available to help with coding tasks.

Sophisticated tools and development environments (such as computer-aided software engineering [CASE] tools or model-driven architecture [MDA] tools) can create software codes automatically, based on the design. These tools and others can also enable reuse of code. CASE includes a set of software programs that help automate the design, development, and implementation of a software system.[13] CASE includes tools to create visual diagrams and representations of the system; an information repository of software designs and components that can be reused; tools to design, generate, and test software codes; and other management tools. CASE can also include tools for helping the developer create the graphical user interface for the software. CASE tools are intended to help developers create software systems more easily and with fewer errors.

MDA is a recent approach to developing software that attempts to separate the design of the software from its physical implementation.[14] It involves creating design models of the functions and features to be implemented in the software and includes tools and techniques to help produce software code from those design models and diagrams. MDA approaches are intended to help developers more easily create software designs without worrying about how the software may be implemented in different environments.

Once the source code is written, developers use compilers to automatically translate the source code into machine code (as shown in Figure 1.1 earlier in this chapter) so that the program can be run on a computer. To make sure that the code works properly, developers test it. There are various types of tests that can be conducted, such as unit tests, integration tests, system tests, and acceptance tests.

Unit tests verify that the functionality of a specific part of the code works—these tests are typically conducted by the developer who created

the code. *Integration* tests check whether the interfaces between connected programs work. *System* tests verify whether the system as a whole works properly. *Acceptance* tests are usually performed by the end user or customer who verifies whether the requirements are satisfied. Acceptance tests can include *alpha* tests (where internal staff play the role of a customer to test the software) or *beta* tests (where the software is released to the external customer for testing).

Software testing can also be *white box* (in which the internal workings of the software are tested) or *black box* (in which the tester examines the functionality of the software without considering how it is written).

After testing, the software is implemented—for off-the-shelf software, this is the point at which the software is made available for sale to the customer. For software created for hire, the software is installed at the customer site.

The software system then transitions to support and maintenance. In this phase, questions about software functionality are answered, bugs are fixed, the software is enhanced to add new features, and the software is upgraded to be compatible with new versions of computer hardware or systems. For many software systems, maintenance is the longest and most expensive phase of the lifecycle.

Companies use a variety of methodologies or approaches for developing software systems. Some of the primary methods include the waterfall, prototyping, spiral development, iterative and incremental development, rapid application development, object-oriented development, and agile development. These are briefly discussed next and illustrated with diagrams.

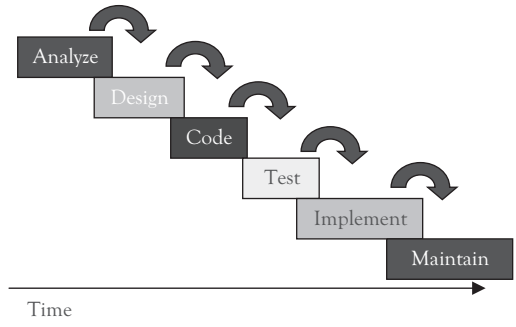


Figure 1.6 Waterfall approach to software development

One of the most common approaches to software development is called a *waterfall* method, illustrated in Figure 1.6. The waterfall approach is a step-by-step process in which developers create the software in phases by first understanding requirements, then designing the system, coding it, testing it, and finally implementing it. The process is sequential: Work in a subsequent phase cannot commence until the prior phase has been successfully completed. The waterfall process, although disciplined and straightforward to manage, suffers from problems such as long development cycles (it can take years to develop a system using this approach) and lack of flexibility (if a new requirement is found in the testing phase, it is difficult to go backward, i.e., *swim upstream* in the waterfall process).[15] Despite its disadvantages, the waterfall approach offers control and discipline of the process, and many companies use it to develop their software systems.

There are many alternative approaches to developing software systems that address some of the limitations of the waterfall process.[16]

Prototyping creates a *mock up* version of a system that is not fully functional, but can be developed quickly. It is useful for requirements determination, unfamiliar technologies, and complex interface design and can be used to manage risk as well as explore new ideas. Figure 1.7 depicts the prototyping process.

Spiral development is a method for developing software that is aimed at reducing the risks of software development projects.[17] A *spiral* starts with determining the objectives and requirements of the project, followed by a series of prototypes to reduce identified project risks. The effort then proceeds into detailed design, coding, testing, and release of the software. The effort is reviewed and the next spiral iteration is planned. Spirals continue until the project is finished. Spiral development can be useful

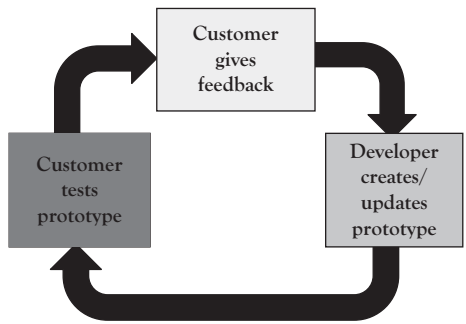


Figure 1.7 Prototyping approach to software development

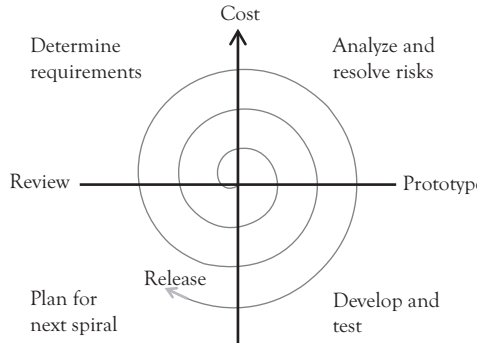


Figure 1.8 Spiral approach to software development

Source: Adapted by permission from Boehm (1988, p. 64) © 1988 by IEEE.

for very large, complex, and uncertain software projects, where there are major risks to be identified and resolved. Figure 1.8 illustrates the spiral approach to software development.

Rapid Application Development (RAD) approaches create *components* or small pieces of software systems that are fully functional and that can be built and installed within 60 to 90 days. A related approach is *Component-based Development* (CBD) in which the functions of the system are divided into small units or pieces. CBD includes defining, implementing, and composing the loosely coupled independent components into systems. The components are developed by separate teams, in parallel, and then integrated into a complete system. RAD-based approaches are very fast, but it can be difficult to separate a system into distinct pieces and it may be difficult to put the pieces together. Figure 1.9 provides a visual representation of the RAD process for software development.

Object-oriented approaches develop software by building self-contained modules (objects) that can be easily replaced, modified, and reused. Traditional approaches to software development structure programs into two distinct elements: code and data; the code includes the methods through which the data are processed. In contrast, an object-oriented approach integrates the code and data using the concept of an *object*. Object-oriented approaches to software development model elements as a collection of discrete objects that correspond to real-world objects. An object is an entity (person, place, or thing) that has data (attributes) and that does things (provides services or has methods). For example, a book is an object. The book object has attributes such as title, author, and

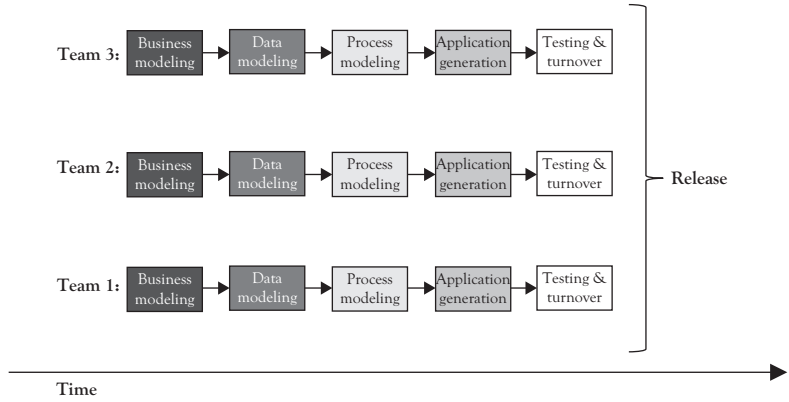


Figure 1.9 Rapid Application Development (RAD) approach to software development

chapter, and it performs services such as be Ordered and be Shipped. In the object-oriented approach, an object *encapsulates* (includes and protects) its attributes; attributes can only be accessed via methods. Objects (more generally called *classes*) are organized into class hierarchies. For example, a *book* class may be at the top of a hierarchy that distinguishes between different types of books at the next level, such as *mystery*, *romance* or *history* or *hard cover* versus *ebook*. Each of these different types of books may not only have unique attributes or methods but also *inherits* attributes and methods defined for the more general book class. Figure 1.5 (shown earlier in this chapter) visually illustrates a class diagram for an online shopping system that was created using object-oriented approaches. As can be seen in this figure, some of the classes in the system include Order, Customer, Item, and Product. Each class has attributes (such as order number and order date for Order) and methods (such as be Placed for Order). Because attributes and methods do not need to be defined for each class in a hierarchy and because objects correspond to things in the real world, object-oriented systems are thought to be easier to develop, understand, and maintain.

The unified approach (UA) is a methodology for object-oriented analysis and design. The UA combines best practices, processes, and guidelines with the Object Management Group's Unified Modeling Language (UML).[18] The UML is a set of notations and conventions used to describe

and model an object-oriented system. The OMG (object management group) agreed to make UML a standard for object-oriented analysis and design in 1995. The current version of UML is version 2. Examples of object-oriented programming languages include C++, Java, Smalltalk, Python, and Ruby. Figure 1.4 earlier in this chapter shows an example of Java code. Many companies have shifted to object-oriented approaches. Object-oriented approaches can also be used in combination with other approaches to software development.

Iterative and Incremental Development (used by many large software companies) combines traditional and modern approaches, whereby a system is repeatedly developed and delivered in increments, versions, or releases. The initial release is the core product. Updates and extensions to the core product are made in subsequent increments, versions, or releases. The system is developed via repeated iterations in small increments. This allows software developers to learn and improve the system as they are developing it. Figure 1.10 illustrates the incremental development approach to software development.

Finally, *agile methodologies* are becoming an increasingly popular way to develop systems. Agile methods break down software development into small pieces or increments that can be completed within a short time frame (time boxes) that typically last from one to four weeks. A small, cross-functional team works on an increment, and at the end of the iteration, a working product results.

Popular agile methodologies include eXtreme Programming (XP) and Scrum. XP promotes small, frequent releases of software after short development cycles.[19] Among other elements, it involves programmers

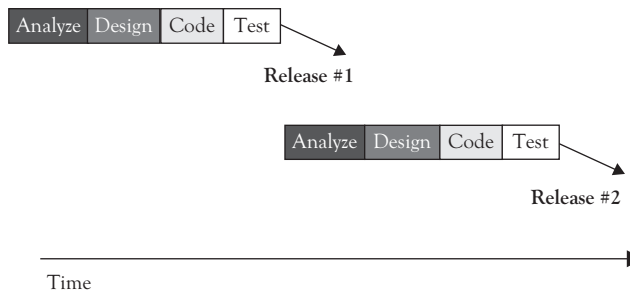


Figure 1.10 Incremental approach to software development

working in pairs, extensive communication with customers, limited management, frequent code reviews, and comprehensive unit tests. The approach gets its name from the idea that the best practices for software development (such as communication and code reviews) are taken to the extreme.

Scrum is a development approach in which the developers work as a team to achieve a common goal.[20] It encourages team colocation and communication. Like XP, Scrum methods are set up to accommodate changes in requirements. Scrum involves three primary roles: the product owner (or customer), the development team, and the scrum master (who enforces Scrum rules and makes sure the development team can be effective). Figure 1.11 shows the Scrum process. As shown in Figure 1.11, each development effort is referred to as a sprint, which is scoped to last between one week and one month. The input to the sprint is the backlog of requirements for the product (which are prioritized by the customer). From the backlog, the goal for the sprint is identified and the requirements relevant to the sprint are collected. The team then works on coding and testing the requirements for the sprint, meeting daily with short scrum meetings to communicate progress and identify impediments to progress. When the sprint is finished and a working piece of software has been created, a final review meeting is held, and the software is released.

Agile methodologies can increase the flexibility and speed of software development, while building in quality. However, the methodologies can be intense for the participants and require close collaboration, which

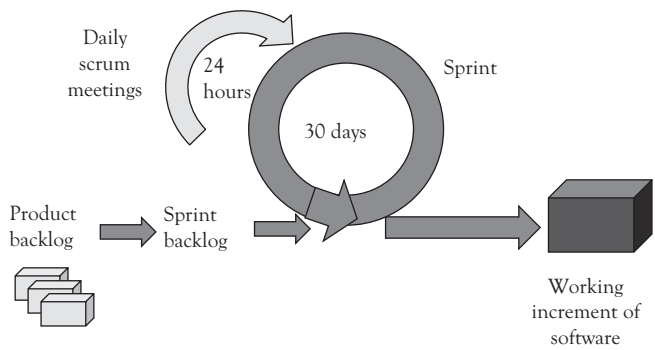


Figure 1.11 Scrum approach to software development

Source: Adapted by permission from Schwaber (2004, Figs. 1–3, p. 9).

may be challenging in some settings (such as where team members are geographically dispersed). In addition, the flexibility of the approach can make it difficult to control the scope of the product.

If a company does not want to use its own staff to develop the software for its product, the company can *outsource* the software development by hiring another firm to create the product using the development approaches and tools previously described. Software companies can even *purchase* software capabilities by acquiring a company that created an application and merging the application with their own product offerings. For example, in 2000, Microsoft acquired Visio Corporation—a Seattle company that developed the Visio diagramming application, for $1.5 billion in stock; Microsoft then integrated Visio within its suite of business productivity products.[21]

Types of Software Products and Services

Companies in the software industry produce software products and software-related services. A software product is a set of one or more computer software programs that a company offers for sale. Examples of software products include a spreadsheet, a word processing program, or an antivirus software tool.

Software products can be organized into two broad categories: systems software and applications software. As explained in Chapter 3, the software industry is segmented by these broad categories of software.

Systems software runs the computer system. It controls, integrates, and manages the individual hardware components so that other software and the users of the system do not have to worry about the lower level details such as reading or transferring data from memory to disk or onto a display. Generally, system software consists of an operating system and utilities such as file managers, display managers, text editors, user authentication (login), security and systems management tools, and networking and device control software.

Applications software is used to accomplish specific tasks other than just running the computer system. Application software may consist of a single program, such as an app to verify a credit card number or to check a password. A group of programs or a *software package* works closely

together to accomplish a task, such as an architectural rending package like AutoCAD. A group of software programs can also be called an *application software system*, such as an accounts receivable system, which is a collection of fundamental programs that may provide some service to a variety of other independent applications. A larger group of programs or software suite includes related, but independent, programs and packages that have a common user interface or shared data format, such as Microsoft Office. The Microsoft Office suite consists of an integrated word processor, spreadsheet, database, presentation software, and other programs. The largest grouping of application software can also be referred to as a systems of systems such as enterprise resource planning systems that are a collection of interdependent systems. Figure 1.12 visually illustrates the different levels of software applications, and Table 1.2 shows examples of each type of software product.

<table>
<tr><th>Instruction</th><th>Program (App)</th><th>Package (System)</th><th>Suite (System of systems)</th></tr>
<tr><td>Instruction₁₁</td><td rowspan="3">Program₁</td><td rowspan="9">Package₁</td><td rowspan="12">Suite₁</td></tr>
<tr><td>Instruction₁₂</td></tr>
<tr><td>…</td></tr>
<tr><td>Instruction₂₁</td><td rowspan="5">Program₂</td></tr>
<tr><td>Instruction₂₂</td></tr>
<tr><td>Instruction₂₃</td></tr>
<tr><td>Instruction₂₄</td></tr>
<tr><td>…</td></tr>
<tr><td>Instruction₃₁</td><td rowspan="3">Program₃</td><td rowspan="3">Package₂</td></tr>
<tr><td>Instruction₃₂</td></tr>
<tr><td>…</td></tr>
<tr><th colspan="4">EXAMPLE</th></tr>
<tr><th>Instructions</th><th>Programs (Apps)</th><th>Packages (Systems)</th><th>Suite (System of systems)</th></tr>
<tr><td>Draw a line</td><td rowspan="2">Draw a rectangle</td><td rowspan="3">Create diagrams with shapes</td><td rowspan="5">Create documents and diagrams with shapes</td></tr>
<tr><td>Connect a line</td></tr>
<tr><td>Draw a curved line</td><td>Draw a circle</td></tr>
<tr><td>Draw a line</td><td rowspan="2">Draw an arrow</td><td rowspan="2">Create documents with shapes</td></tr>
<tr><td>Draw an endpoint</td></tr>
</table>

Figure 1.12 Levels of software

Table 1.2 Examples of types of systems and applications software products

Systems software	Applications software
Operating systems (e.g., MS Windows, Linux, Unix, DOS)	Office productivity suite (e.g., MS Office)
Device drivers (e.g., graphics drivers and firmware)	Game software (e.g., World of Warcraft)
Compiler or Debugger software	Enterprise resource planning software (e.g., SAP)
Utility software (Network utilities, data compression or encryption software, antivirus software, disk defragmenters, archive utilities, etc.)	Graphics software (e.g., Adobe)
	Database management systems (e.g., SQL Server, Oracle database management system)

In addition, software products can be organized into software product lines. A software product line is a set of software programs that share common features that satisfy the needs of a particular market segment. Programs in a software product line are developed using a common or shared set of software functions and tools. A software product line might distinguish between types of users and include the products that fit the needs of the users in a particular segment. For example, one software product line might include software security products for corporations while another software product line could include software security products for noncorporate, individual users.

As described earlier, producing software products involves a variety of related activities. A software service includes the different tasks associated with producing a software product, such as design, programming, testing, documentation, deployment, training, and support. Companies can sell one or more of these services to customers.

Companies in the Software Industry in This Book

The software industry accounts for a significant market share of the information technology (IT) industry overall. Figure 1.13 shows the global spending on IT products and services projected by Forrester for 2013.[22] Forrester estimates a total of $2.069 trillion in spending on products produced by the IT industry in 2013, of which $542 billion was spent on software products and services, and $389 billion was spent on IT consulting and systems integration services.

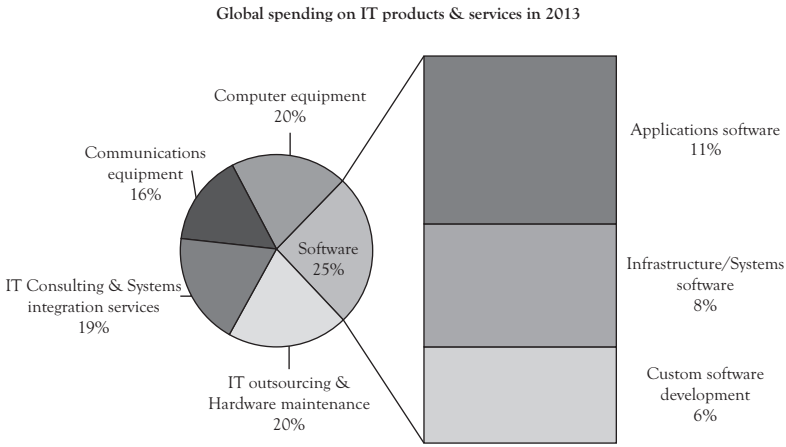


Figure 1.13 Projected sales of IT industry products and services, by major segment

Source: Lunden (2013).

As can be seen in Figure 1.13, the IT industry produces and sells computer and communications equipment (which together comprise 36 percent of IT industry sales); IT outsourcing, systems integration, and consulting services (which together make up 39 percent of the IT industry); and software products and services which account for 25 percent of IT industry sales. Within the software product segment, there are three major types of products: applications software products, infrastructure/systems software products, and custom software development services, of which applications and systems software products account for 75 percent of the projected 2013 sales in the software segment of the IT industry.

From the U.S. government's perspective, software is classified as a business service (Standard Industrial Classification Code[SIC],73) and is further described in industry group SIC 737: Computer Programming, Data Processing, and Other Computer Related Services. This industry group includes companies who provide a variety of computing products and services, ranging from computer hardware repair to custom programming. Table 1.3 shows the SIC 737 industry group and subgroups, relating them also to their respective North American Industry Classification

Table 1.3 SIC 737 industry subgroups for computer programming, data processing, and other computer-related services

SIC Code	NAICS Code	Title	Description
7371*	541511	Computer programming services	Companies primarily engaged in providing computer programming services on a contract or fee basis. Companies in this industry perform a variety of additional services, such as computer software design and analysis; modifications of custom software; and training in the use of custom software.
7372**	511210	Prepackaged software	Companies primarily engaged in the design, development, and production of prepackaged computer software. Important products of this industry include operating, utility, and applications programs. Companies in this industry may also provide services such as preparation of software documentation for the user installation of software for the user; and training the user in the use of the software.
7373***	541512	Computer integrated systems design	Companies primarily engaged in developing or modifying computer software and packaging or bundling the software with purchased computer hardware (computers and computer peripheral equipment) to create and market an integrated system for specific application. Companies in this industry must provide each of the following services: (1) the development or modification of the computer software; (2) the marketing of purchased computer hardware; and (3) involvement in all phases of systems development from design through installation.
7374	514210	Computer processing and data preparation and processing services	Companies primarily engaged in providing computer processing and data preparation services. The service may consist of complete processing and preparation of reports from data supplied by the customer or a specialized service, such as data entry or making data processing equipment available on an hourly or time-sharing basis.

(*continued*)

Table 1.3 SIC 737 industry subgroups for computer programming, data processing, and other computer-related services (continued)

SIC Code	NAICS Code	Title	Description
7375	514191	Information retrieval services	Companies primarily engaged in providing on-line information retrieval services on a contract or fee basis. The information generally involves a range of subjects and is taken from other primary sources.
7376***	541513	Computer facilities management services	Companies primarily engaged in providing on-site management and operation of computer systems and data processing facilities on a contract or fee basis.
7377	532420	Computer rental and leasing	Companies primarily engaged in renting or leasing computers and related data processing equipment on the customers' site, whether or not also providing maintenance or support services.
7378	811212	Computer maintenance and repair	Companies primarily engaged in the maintenance and repair of computers and computer peripheral equipment.
7379***	334611, 541512, 541519	Computer-related services, not elsewhere classified	Companies primarily engaged in supplying computer-related services, not elsewhere classified. Computer consultants operating on a contract or fee basis are classified in this industry.

Source: U.S. Department of Labor (2013).

**Note*: Subgroup SIC 7371 performs custom software development for hire. The book covers companies in this subgroup as the primary service involves software development.

** *Note*: Subgroup SIC 7372 is a focal industry segment that is covered in this book.

*** *Note*: Firms in subgroups SIC 7373, 7376, and 7379 that provide software-related services are covered in this book.

System codes (NAICS), a more recent classification system of industries introduced in 1997.[23]

Given the complexities of the types of products and services offered by the companies in the SIC 737 industry group, it is important to clarify the focus of the software industry profiled in this book.

As noted at the beginning of this chapter, this book examines companies that make and sell applications software products, systems software products, or their related software services. This includes the subsegments within the software segment of the IT industry shown earlier in Figure 1.13 (i.e., applications software, systems software, and custom software development). Companies producing IT consulting and systems integration services are also relevant to the extent that the services are software related. This also corresponds to companies that list their primary SIC code as 7371, 7372, 7373, 7376, or 7379.

Companies producing software for the mass market are included in SIC 7372 or the NAICS number 511210. This segment is frequently referred to as the *packaged* or *prepackaged* software industry. As noted in Table 1.3, companies in the prepackaged software industry SIC 7372 are primarily engaged in the design, development, and production of prepackaged computer software. These companies create and market software products that perform functions such as desktop productivity suites, enterprise resource planning systems, customer relationship management systems, business intelligence tools, video games, statistical software, operating systems, and security software. Such products generate about $400 billion in annual sales worldwide.[24] Companies in this industry also carry out operations necessary for producing and distributing their software products, such as designing, providing documentation, assisting in installation, and providing support services to software purchasers. Prominent companies in this industry include Microsoft, Oracle, Salesforce.com, Adobe, and Intuit.

Companies listed in SIC 7371, SIC 7373, SIC 7376, and SIC 7379 (NAICS codes 541511, 541512, or 541513) also perform software development but produce custom software for hire on a contract or fee basis or provide software-related services such as systems integration. Although a primary focus of this book is on companies that produce software for sale in the mass market, the custom software development services

sector has become increasingly important, especially in the global software industry. For example, Infosys, a large consulting and software services company that is headquartered in Bangalore, India, is listed in SIC 7371. Custom software development generates over $130 billion in sales worldwide, while IT consulting and systems integration services generate $389 billion in global sales.[25] In addition to Infosys, prominent companies in this industry segment include IBM, Hewlett-Packard, Computer Sciences Corporation, and Accenture.

This book does NOT cover companies that operate in other sectors within the SIC 737 industry that are hardware or data related—including companies providing data preparation and processing services (SIC 7374), information retrieval services (SIC 7375), computer rental and leasing services (SIC 7377), and computer maintenance or repair services (SIC 7378). The services provided by companies within these subsectors do not involve producing software products or software services for sale.

It should also be noted that this book does not cover Internet retailing companies (such as amazon.com) whose primary product for sale is not software, nor does it consider companies that are primarily computer equipment manufacturers (such as Advanced Micro Devices or AMD whose primary SIC code is 3674, semiconductors and related equipment). These companies do not list themselves as belonging in the software industry and may not sell the software if they create it (or such sales may be a minor part of their business).

Finally, it is important to note that companies may produce their own software, internally for their own use, but do not sell it, and such internal software activities are not considered part of the software industry (and are not covered in this book).

Key Takeaways

The software industry profiled in this book focuses on companies that produce and sell software and software-related services for hire or for the mass market. The software industry is dynamic and constantly evolving. It is high risk but also high reward for the participants. Thus, it is very important to understand some fundamental concepts that define the industry.

The main *product* is software and related services. As described in this chapter, software does not have a physical component. It is just a set of instructions that tells a computer what to do. Software is created by programmers using different languages, development approaches, and tools. These languages have evolved over the last 50 years and have generally become easier and less expensive to use. There are many approaches to software development, with more recent approaches focused on developing software more quickly and easily, in smaller increments. Software can be as small as a tiny app or program with a few thousand lines of code running on a mobile phone or as large as an enterprise-wide system of systems with hundreds of millions of lines of code.

Companies in the software industry make either systems software products or applications software products (or both) or provide software-related services, and the industry is segmented by these products. This book examines companies that primarily make packaged software products for sale or custom software development for hire and that provide related services. Computer hardware manufacturers that do not produce software or software services for sale, Internet retailers, and companies that develop software for their internal use are not included.

The next chapter—Chapter 2—explores the history of the software industry. Chapter 3 includes a more detailed discussion of the companies in the software industry and how they compete. While Chapter 3 has a U.S. focus, Chapter 4 takes a global perspective of the software industry.

THE BUSINESS EXPERT PRESS DIGITAL LIBRARIES

EBOOKS FOR BUSINESS STUDENTS

Curriculum-oriented, born-digital books for advanced business students, written by academic thought leaders who translate real-world business experience into course readings and reference materials for students expecting to tackle management and leadership challenges during their professional careers.

POLICIES BUILT BY LIBRARIANS

- *Unlimited simultaneous usage*
- *Unrestricted downloading and printing*
- *Perpetual access for a one-time fee*
- *No platform or maintenance fees*
- *Free MARC records*
- *No license to execute*

The Digital Libraries are a comprehensive, cost-effective way to deliver practical treatments of important business issues to every student and faculty member.

For further information, a free trial, or to order, contact:

sales@businessexpertpress.com

www.businessexpertpress.com/librarians

A Profile of the Software Industry

Emergence, Ascendance, Risks, and Rewards

Sandra A. Slaughter

Software plays a critical role in today's global information economy. It runs the computers, networks, and devices that enable countless products and services. Software varies in size from vast enterprise and communications systems like the enormous enterprise resource planning system from SAP to the tiny app Angry Birds.

This book offers a profile of the software industry and the companies in the industry. It describes the primary products and services produced; reviews its history; explains how the industry is structured; discusses its economics and competitive environment; and examines important trends and issues including globalization, workforce, regulation, and the emergence of new software business models.

Software runs the computers and networks that support the flow of information in the global economy, and this book provides a real look at the intricacies of this industry.

Sandra A. Slaughter has a PhD from the University of Minnesota and is the Alton M. Costley Chair and Professor of Information Technology Management in the Ernest Scheller Jr. College of Business at the Georgia Institute of Technology. Before joining the college's faculty in 2007, she was a member of the information technology faculty in the David A. Tepper School of Business at Carnegie Mellon University. Prior to her academic career, Slaughter worked in the software industry with Hewlett-Packard, the Allen-Bradley division of Rockwell International, and Square D Corporation. She has published more than 100 articles in research journals, conference proceedings, and edited books; her work has won awards; and her research has been supported by more than $2 million in grants.

INDUSTRY PROFILES COLLECTION

Donald N. Stengel, *Editor*

businessexpert Press

www.businessexpertpress.com

ISBN: 978-1-60649-654-1

9 781606 496541 90000