CHAPTER 9

Understanding Software: A Primer for Managers

1. INTRODUCTION

LEARNING OBJECTIVES

1. Recognize the importance of software and its implications for the firm and strategic decision making.
2. Understand that software is everywhere; not just in computers, but also cell phones, cars, cameras, and many other technologies.
3. Know what software is and be able to differentiate it from hardware.
4. List the major classifications of software and give examples of each.

We know **computing hardware** is getting faster and cheaper, creating all sorts of exciting and disruptive opportunities for the savvy manager. But what's really going on inside the box? It's **software** that makes the magic of computing happen. Without software, your PC would be a heap of silicon wrapped in wires encased in plastic and metal. But it's the instructions—the software code—that enable a computer to do something wonderful, driving the limitless possibilities of information technology.

Software is everywhere. An inexpensive cell phone has about one million lines of code.[1] Ford automobiles actually have more lines of code than Twitter and Facebook combined.[2] In this chapter we'll take a peek inside the chips to understand what software is. A lot of terms are associated with software: operating systems, applications, enterprise software, distributed systems, and more. We'll define these terms up front, and put them in a managerial context. A follow-up chapter, Chapter 10, will focus on changes impacting the software business, including open source software, software as a service (SaaS), and cloud computing. These changes are creating an environment radically different from the software industry that existed in prior decades—confronting managers with a whole new set of opportunities and challenges.

Managers who understand software can better understand the possibilities and impact of technology. They can make better decisions regarding the strategic value of IT and the potential for technology-driven savings. They can appreciate the challenges, costs, security vulnerabilities, legal and compliance issues, and limitations involved in developing and deploying technology solutions. In the next two chapters we will closely examine the software industry and discuss trends, developments and economics—all of which influence decisions managers make about products to select, firms to partner with, and firms to invest in.

computer hardware

The physical components of information technology, which can include the computer itself plus peripherals such as storage devices, input devices like the mouse and keyboard, output devices like monitors and printers, networking equipment, and so on.

software

A computer program or a collection of programs. It is a precise set of instructions that tells hardware what to do.

1.1 What Is Software?

When we refer to computer hardware (sometimes just hardware), we're talking about the physical components of information technology—the equipment that you can physically touch, including computers, storage devices, networking equipment, and other peripherals.

Software refers to a computer program or collection of programs—sets of instructions that tell the hardware what to do. Software gets your computer to behave like a Web browser or word processor, makes your iPod play music and video, and enables your bank's ATM to spit out cash.

It's when we start to talk about the categories of software that most people's eyes glaze over. To most folks, software is a big, incomprehensible alphabet soup of acronyms and geeky phrases: OS, VB, SAP, SQL, to name just a few.

operating system

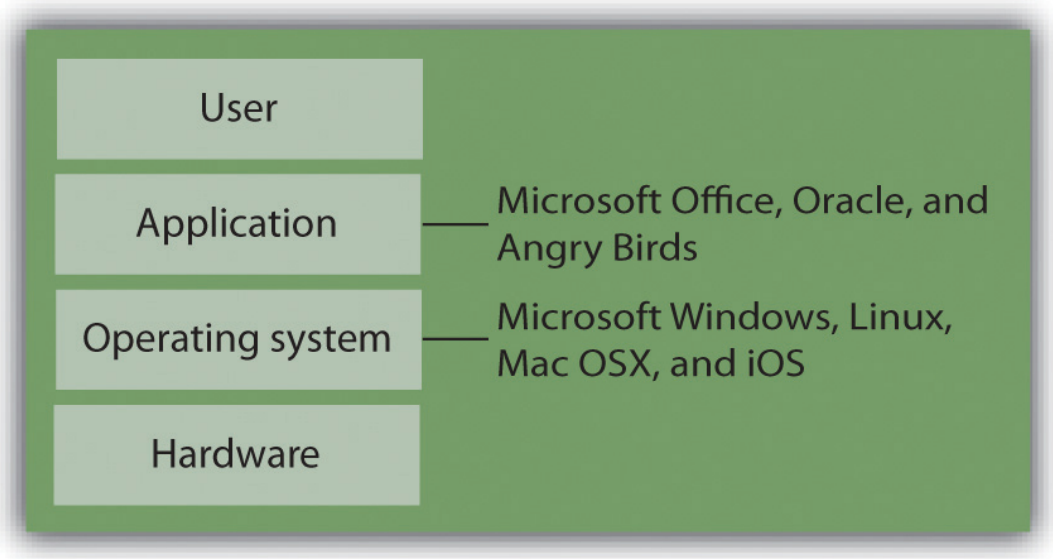
The software that controls the computer hardware and establishes standards for developing and executing applications.

applications

Includes desktop applications, enterprise software, utilities, and other programs that perform specific tasks for users and organizations.

Don't be intimidated. The basics are actually pretty easy to understand. But it's not soup; it's more of a layer cake. Think about computer hardware as being at the bottom of the layer cake. The next layer is the **operating system**, the collection of programs that control the hardware. Windows, Mac OS X, iOS, and Linux are operating systems. On top of that layer are **applications**—a range of which include end-user programs like those in Office, apps that run on smartphones, and the complex set of programs that manage a business's inventory, payroll, and accounting. At the top of the cake are users.

FIGURE 9.1 The Hardware/Software Layer Cake



The flexibility of these layers gives computers the customization options that managers and businesses demand. Understanding how the layers relate to each other helps you make better decisions on what options are important to your unique business needs, can influence what you buy, and may have implications for everything from competitiveness to cost overruns to security breaches. What follows is a manager's guide to the main software categories with an emphasis on why each is important.

KEY TAKEAWAYS

- Software refers to a computer program or collection of programs. It enables computing devices to perform tasks.
- You can think of software as being part of a layer cake, with hardware at the bottom; the operating system controlling the hardware and establishing standards, the applications executing one layer up, and the users at the top.
- How these layers relate to one another has managerial implications in many areas, including the flexibility in meeting business demand, costs, legal issues and security.
- Software is everywhere—not just in computers, but also in cell phones, cars, cameras, and many other technologies.

QUESTIONS AND EXERCISES

1. Explain the difference between hardware and software.
2. Why should a manager care about software and how software works? What critical organizational and competitive factors can software influence?
3. What role has software played in your decision to select certain products? Has this influenced why you favored one product or service over another?
4. Find the *Fortune* 500 list online. Which firm is the highest ranked software firm? While the *Fortune* 500 ranks firms according to revenue, what's this firm's profitability rank? What does this discrepancy tell you about the economics of software development? Why is the software business so attractive to entrepreneurs?
5. Refer to earlier chapters (and particularly to Chapter 2): Which resources for competitive advantage might top software firms be able to leverage to ensure their continued dominance? Give examples of firms that have leveraged these assets, and why they are so strong.

2. OPERATING SYSTEMS

LEARNING OBJECTIVES

1. **Understand what an operating system is and why computing devices require operating systems.**
2. **Appreciate how embedded systems extend Moore's Law, allowing firms to create "smarter" products and services**

Computing hardware needs to be controlled, and that's the role of the operating system. The operating system (sometimes called the "OS") provides a common set of controls for managing computer hardware, making it easier for users to interact with computers and for programmers to write application software. Just about every computing device has an operating system—desktops and laptops, enterprise-class server computers, your mobile phone. Even specialty devices like iPods, video game consoles, and television set top boxes run some form of OS.

Some firms, like Apple and Nintendo, develop their own proprietary OS for their own hardware. Microsoft sells operating systems to everyone from Dell to the ATM manufacturer Diebold (listen for the familiar Windows error beep on some cash machines). And there are a host of specialty firms, such as Wind River (purchased by Intel), that help firms develop operating systems for all sorts of devices that don't necessarily look like a PC, including cars, video editing systems, and fighter jet control panels.

Anyone who has used both a PC and a Mac and has noticed differences across these platforms can get a sense of the breadth of what an operating system does. Even for programs that are otherwise identical for these two systems (like the Firefox browser), subtitle differences are visible. Screen elements like menus, scroll bars, and window borders look different on the Mac than they do in Windows. So do the dialogue boxes that show up when you print or save.

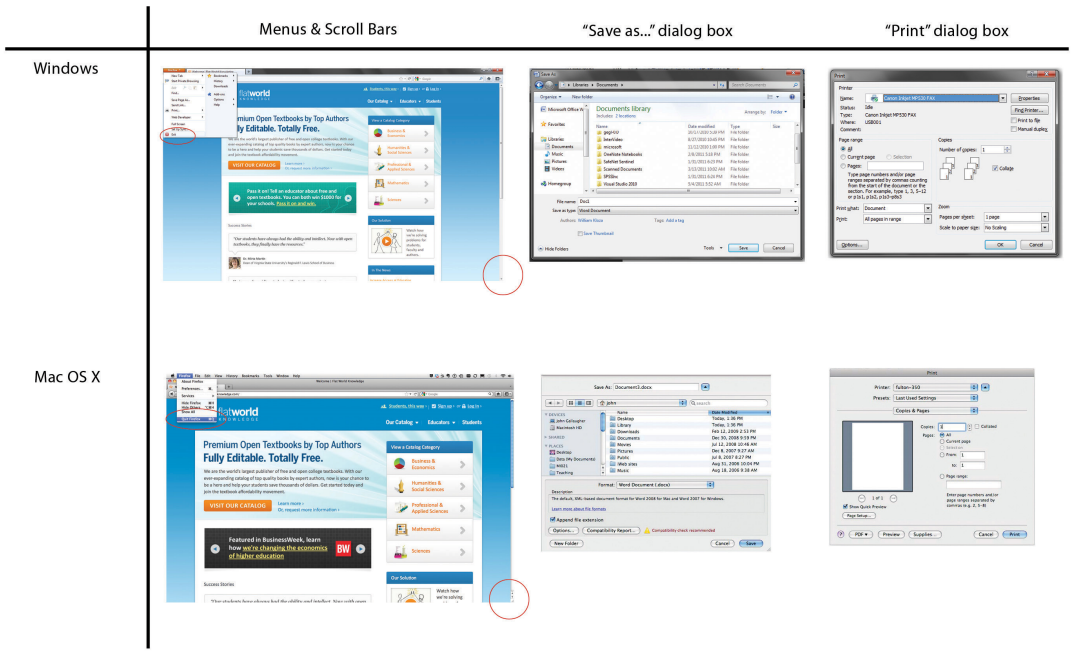
These items look and behave differently because each of these functions touches the hardware, and the team that developed Microsoft Windows created a system distinctly different from their Macintosh counterparts at Apple. Graphical **user interface (UI)** items like scroll bars and menus are displayed on the hardware of the computer display. Files are saved to the hardware of a hard drive or other storage device. Most operating systems also include control panels, desktop file management, and other support programs to work directly with hardware elements like storage devices, displays, printers, and networking equipment. The Macintosh Finder and the Windows Explorer are examples of components of these operating systems. The consistent look, feel, and functionality that operating systems enforce across various programs help make it easier for users to learn new software, which reduces training costs and operator error. See Figure 9.2 for similarities and differences.

user interface (UI)

The mechanism through which users interact with a computing device. The UI includes elements of the graphical user interface (or GUI, pronounced "*gooey*"), such as windows, scroll bars, buttons, menus, and dialogue boxes; and can also include other forms of interaction, such as touch screens, motion sensing controllers, or tactile devices used by the visually impaired.

FIGURE 9.2

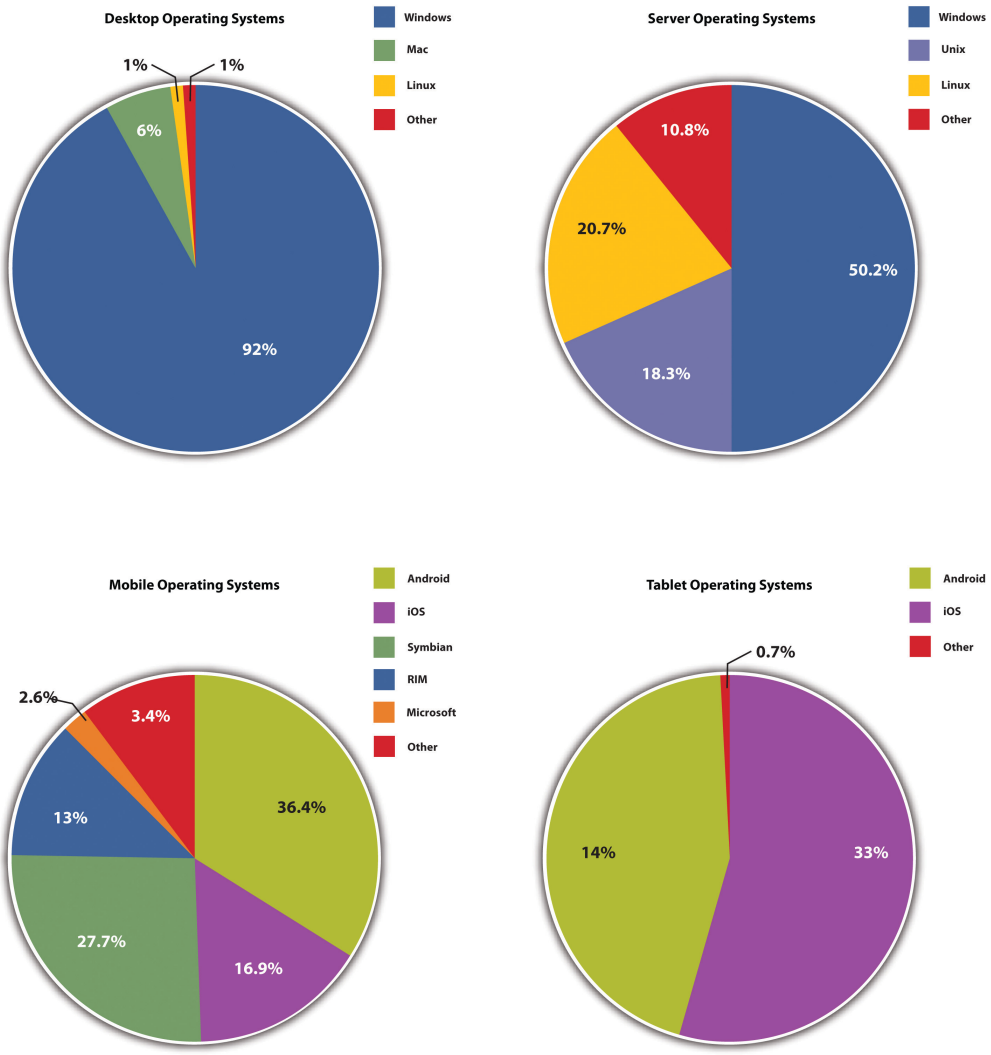
Differences between the Windows and Mac operating systems are evident throughout the user interface, particularly when a program interacts with hardware.



Operating systems are also designed to give programmers a common set of commands to consistently interact with the hardware. These commands make a programmer's job easier by reducing program complexity and making it faster to write software while minimizing the possibility of errors in code. Consider what an OS does for the Wii game developer. Nintendo's Wii OS provides Wii programmers with a set of common standards to use to access the Wiimote, play sounds, draw graphics, save files, and more. Without this, games would be a lot more difficult to write, they'd likely look differently, be less reliable, would cost more, and there would be fewer titles available.

Similarly, when Apple provided developers with a common set of robust, easy-to-use standards for the iPhone and (via the App Store) an easy way for users to install these applications on top of the iPhone/iPod touch/iPad's operating system (iOS), software development boomed, and Apple became hands-down the most versatile mobile computing device available.[3] In Apple's case some *fifty thousand apps* became available through the App Store in less than a year. A good OS and software development platform can catalyze network effects (see Chapter 6). While the OS seems geeky, its effective design has very strategic business implications!

FIGURE 9.3 Operating System Market Share for Desktop, Server, and Mobile Phones



Source: HitsLink (desktop, May 2012), IDC (server, Q1 2012), Gartner (mobile, May 2012), and IDC (March 2012). Note that Android tablet figure includes Kindle Fire.

firmware

Software stored on nonvolatile memory chips (as opposed to being stored on devices such as hard drives or removable discs). Despite the seemingly permanent nature of firmware, many products allow for firmware to be upgraded online or by connecting to another device.

embedded systems

Special-purpose software designed and included inside physical products (often on firmware). Embedded systems help make devices "smarter," sharing usage information, helping diagnose problems, indicating maintenance schedules, providing alerts, or enabling devices to take orders from other systems.

Firmware and Embedded Systems

Most personal computers have an operating system installed on their hard drives. This system allows the OS to be replaced or upgraded easily. But many smaller, special-purpose computing devices have their operating systems installed on nonvolatile memory, often on read-only memory (ROM) chips. Control programs stored on chips are sometimes referred to as **firmware**. The OS in an iPod, mobile phone, or your TV's set-top box is most likely stored as firmware. Your PC also has a tiny bit of firmware that allows it to do very basic functions like start-up (boot) and begin loading its operating system from disk.

Another term you might hear is **embedded systems**. As computing gets cheaper, special-purpose technology is increasingly becoming embedded into all sorts of devices like cars, picture frames, aircraft engines, photocopiers, and heating and air conditioning systems. The software programs that make up embedded systems are often stored as firmware too.

Moore's Law (see Chapter 5) enables embedded systems, and these systems can create real strategic value. The Otis Elevator Company, a division of United Technologies, uses embedded systems in its products to warn its service centers when the firm's elevators, escalators, and moving walkways need maintenance or repair. This warning provides Otis with several key benefits:

1. Since products automatically contact Otis when they need attention, these systems generate a lucrative service business for the firm and make it more difficult for third parties to offer a competing business servicing Otis products.
2. Products contact service technicians to perform maintenance based on exact needs (e.g., lubricant is low, or a part has been used enough to be replaced) rather than guessed schedules, which makes service more cost-effective, products less likely to break down, and customers happier.
3. Any product failures are immediately detected, with embedded systems typically dispatching technicians before a client's phone call.
4. The data is fed back to Otis's R&D group, providing information on reliability and failure so that engineers can use this info to design better products.

Collectively, software embedded on tiny chips yields very big benefits, for years helping Otis remain at the top of its industry.

KEY TAKEAWAYS

- The operating system (OS) controls a computer's hardware and provides a common set of commands for writing programs.
- Most computing devices (enterprise-class server computers, PCs, phones, set-top boxes, video games, cars, the Mars Rover) have an operating system.
- Some products use operating systems provided by commercial firms, while others develop their own operating system. Others may leverage open source alternatives (see Chapter 10).
- Embedded systems are special-purpose computer systems designed to perform one or a few dedicated functions, and are frequently built into conventional products like cars, air conditioners, and elevators.
- Embedded systems can make products and services more efficient, more reliable, more functional, and can enable entire new businesses and create or reinforce resources for competitive advantage.

QUESTIONS AND EXERCISES

1. What does an operating system do? Why do you need an operating system? How do operating systems make a programmer's job easier? How do operating systems make life easier for end users?
2. How has the market for desktop, server, and mobile operating systems changed in recent years? Do certain products seem to be gaining traction? Why do you think this is the case?
3. What kinds of operating systems are used in the devices that you own? On your personal computer? Your mobile phone? The set-top box on top of your television? Are there other operating systems that you come into contact with? If you can't tell which operating system is in each of these devices, see if you can search the Internet to find out.
4. For your list in the prior question (and to the extent that you can), diagram the hardware/software "layer cake" for these devices.
5. For this same list, do you think each device's manufacturer wrote all of the software that you use on these devices? Can you add or modify software to all of these devices? Why or why not? What would the implications be for cost, security, complexity, reliability, updates and upgrades, and the appeal of each device?
6. Some ATM machines use Windows. Why would an ATM manufacturer choose to build its systems owing Windows? Why might it want to avoid this? Are there other non-PC devices you've encountered that were running some form of Windows?
7. What are embedded systems? When might firms want to install software on chips instead of on a hard drive?
8. It's important to understand how technology impacts a firm's strategy and competitive environment. Consider the description of Otis elevator's use of embedded systems. Which parts of the value chain does this impact? How? Consider the "five forces": How does the system impact the firm's competitive environment? Are these systems a source of competitive advantage? If not, explain why not? If they are, what kinds of resources for competitive advantage can these kinds of embedded systems create?
9. Can you think of other firms that can or do leverage embedded systems? Provide examples and list the kinds of benefits these might offer firms and consumers.
10. Research the Americans with Disabilities Act of 1990 (or investigate if your nation has a similar law), and the implications of this legislation for software developers and Web site operators. Have firms been successfully sued when their software or Web sites could not be accessed by users with physical challenges? What sorts of issues should developers consider when making their products more accessible? What practices might they avoid?

3. APPLICATION SOFTWARE

LEARNING OBJECTIVES

1. **Appreciate the difference between desktop and enterprise software.**
2. **List the categories of enterprise software.**
3. **Understand what an ERP (enterprise resource planning) software package is.**
4. **Recognize the relationship of the DBMS (database system) to the other enterprise software systems.**
5. **Recognize both the risks and rewards of installing packaged enterprise systems.**

Operating systems are designed to create a **platform** so that programmers can write additional applications, allowing the computer to do even more useful things. While operating systems control the hardware, *application software* (sometimes referred to as *software applications*, *applications*, or even just *apps*) perform the work that users and firms are directly interested in accomplishing. Think of applications as the place where the users or organization's real work gets done. As we learned in Chapter 6, the more application software that is available for a platform (the more games for a video game console, the more apps for your phone), the more valuable it potentially becomes.

platform

Products and services that allow for the development and integration of software products and other complementary goods. Windows, the iPhone, the Wii, and the standards that allow users to create Facebook apps are all platforms.

desktop software

Applications installed on a personal computer, typically supporting tasks performed by a single user.

enterprise software

Applications that address the needs of multiple users throughout an organization or work group.

software package

A software product offered commercially by a third party.

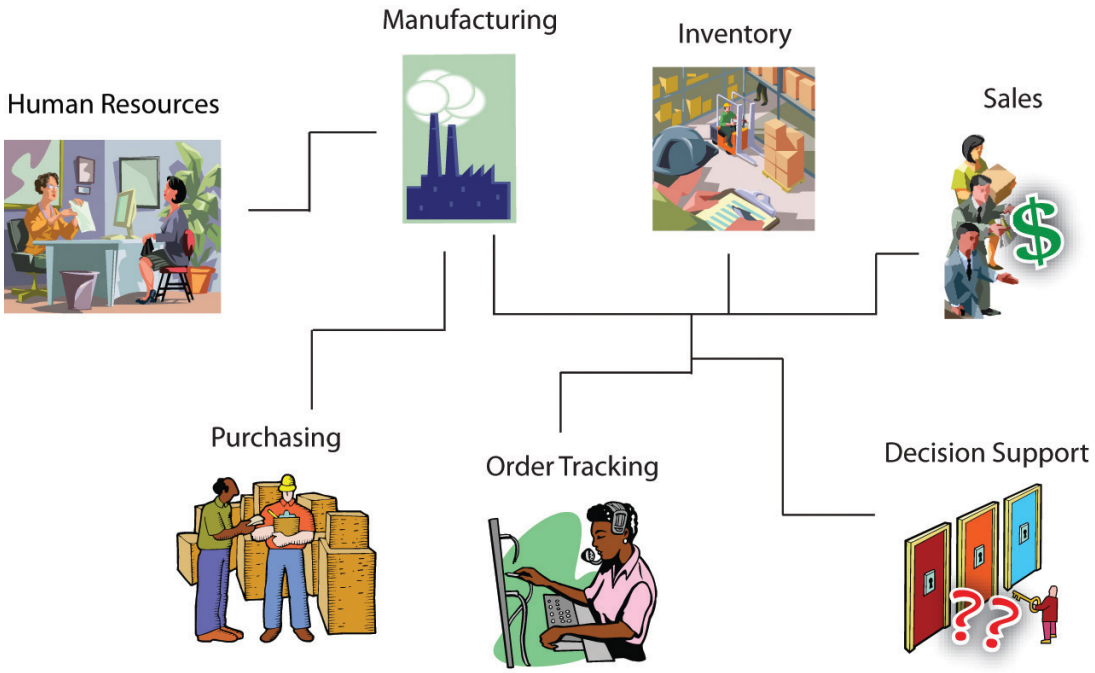
enterprise resource planning (ERP)

A software package that integrates the many functions (accounting, finance, inventory management, human resources, etc.) of a business.

Desktop software refers to applications installed on a personal computer—your browser, your Office suite (e.g., word processor, spreadsheet, presentation software), photo editors, and computer games are all desktop software. **Enterprise software** refers to applications that address the needs of multiple, simultaneous users in an organization or work group. Most companies run various forms of enterprise software programs to keep track of their inventory, record sales, manage payments to suppliers, cut employee paychecks, and handle other functions.

Some firms write their own enterprise software from scratch, but this can be time consuming and costly. Since many firms have similar procedures for accounting, finance, inventory management, and human resource functions, it often makes sense to buy a **software package** (a software product offered commercially by a third party) to support some of these functions. So-called **enterprise resource planning (ERP)** software packages serve precisely this purpose. In the way that Microsoft can sell you a suite of desktop software programs that work together, many companies sell ERP software that coordinates and integrates many of the functions of a business. The leading ERP vendors include the firm's SAP and Oracle, although there are many firms that sell ERP software. A company doesn't have to install all of the modules of an ERP suite, but it might add functions over time—for example, to plug in an accounting program that is able to read data from the firm's previously installed inventory management system. And although a bit more of a challenge to integrate, a firm can also mix and match components, linking software the firm has written with modules purchased from different enterprise software vendors.

FIGURE 9.4 ERP in Action[4]



An ERP system with multiple modules installed can touch many functions of the business:

- *Sales*—A sales rep from Vermont-based SnowboardCo. takes an order for five thousand boards from a French sporting goods chain. The system can verify credit history, apply discounts, calculate price (in euros), and print the order in French.
- *Inventory*—While the sales rep is on the phone with his French customer, the system immediately checks product availability, signaling that one thousand boards are ready to be shipped from the firm's Burlington warehouse, the other four thousand need to be manufactured and can be delivered in two weeks from the firm's manufacturing facility in Guangzhou.
- *Manufacturing*—When the customer confirms the order, the system notifies the Guangzhou factory to ramp up production for the model ordered.
- *Human Resources*—High demand across this week's orders triggers a notice to the Guangzhou hiring manager, notifying her that the firm's products are a hit and that the flood of orders coming in globally mean her factory will have to hire more workers to keep up.

- *Purchasing*—The system keeps track of raw material inventories, too. New orders trigger an automatic order with SnowboardCo.'s suppliers, so that raw materials are on hand to meet demand.
- *Order Tracking*—The French customer can log in to track her SnowboardCo. order. The system shows her other products that are available, using this as an opportunity to cross-sell additional products.
- *Decision Support*—Management sees the firm's European business is booming and plans a marketing blitz for the continent, targeting board models and styles that seem to sell better for the Alps crowd than in the U.S. market.

Other categories of enterprise software that managers are likely to encounter include the following:

- **customer relationship management (CRM)** systems used to support customer-related sales and marketing activities
- **supply chain management (SCM)** systems that can help a firm manage aspects of its value chain, from the flow of raw materials into the firm through delivery of finished products and services at the point-of-consumption
- **business intelligence (BI) systems**, which use data created by other systems to provide reporting and analysis for organizational decision making

Major ERP vendors are now providing products that extend into these and other categories of enterprise application software, as well.

Most enterprise software works in conjunction with a **database management system (DBMS)**, sometimes referred to as a "database system." The database system stores and retrieves the data that an application creates and uses. Think of this as another additional layer in our cake analogy. Although the DBMS is itself considered an application, it's often useful to think of a firm's database systems as sitting above the operating system, but under the enterprise applications. Many ERP systems and enterprise software programs are configured to share the same database system so that an organization's different programs can use a common, shared set of data. This system can be hugely valuable for a company's efficiency. For example, this could allow a separate set of programs that manage an inventory and point-of-sale system to update a single set of data that tells how many products a firm has to sell and how many it has already sold—information that would also be used by the firm's accounting and finance systems to create reports showing the firm's sales and profits.

Firms that don't have common database systems with consistent formats across their enterprise often struggle to efficiently manage their value chain. Common procedures and data formats created by packaged ERP systems and other categories of enterprise software also make it easier for firms to use software to coordinate programs between organizations. This coordination can lead to even more value chain efficiencies. Sell a product? Deduct it from your inventory. When inventory levels get too low, have your computer systems send a message to your supplier's systems so that they can automatically build and ship replacement product to your firm. In many cases these messages are sent without any human interaction, reducing time and errors. And common database systems also facilitate the use of BI systems that provide critical operational and competitive knowledge and empower decision making. For more on CRM and BI systems, and the empowering role of data, see Chapter 11.

customer relationship management (CRM)

Systems used to support customer-related sales and marketing activities.

supply chain management (SCM)

Systems that can help a firm manage aspects of its value chain, from the flow of raw materials into the firm, through delivery of finished products and services at the point-of-consumption.

business intelligence (BI) systems

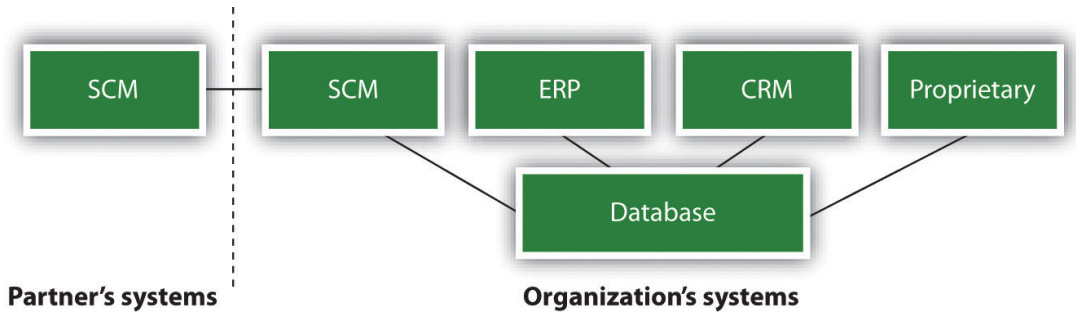
Systems that use data created by other systems to provide reporting and analysis for organizational decision making.

database management system (DBMS)

Sometimes referred to as database software; software for creating, maintaining, and manipulating data.

FIGURE 9.5

An organization's database management system can be set up to work with several applications both within and outside the firm.



The Rewards and Risks of Packaged Enterprise Systems

When set up properly, enterprise systems can save millions of dollars and turbocharge organizations. For example, the CIO of office equipment maker Steelcase credited the firm's ERP with an eighty-million-dollar reduction in operating expenses saved from eliminating redundant processes and making data more usable. The CIO of Colgate Palmolive also praised their ERP, saying, "The day we turned the switch on, we dropped two days out of our order-to-delivery cycle."[5] Packaged enterprise systems can streamline processes, make data more usable, and ease the linking of systems with software across the firm and with key business partners. Plus, the software that makes up these systems is often debugged, tested, and documented with an industrial rigor that may be difficult to match with proprietary software developed in-house.

But for all the promise of packaged solutions for standard business functions, enterprise software installations have proven difficult. Standardizing business processes in software that others can buy means that those functions are easy for competitors to match, and the vision of a single monolithic system that delivers up wondrous efficiencies has been difficult for many to achieve. The average large company spends roughly $15 million on ERP software, with some installations running into the hundreds of millions of dollars.[6] And many of these efforts have failed disastrously.

FoxMeyer was once a six-billion-dollar drug distributor, but a failed ERP installation led to a series of losses that bankrupted the firm. The collapse was so rapid and so complete that just a year after launching the system, the carcass of what remained of the firm was sold to a rival for less than $80 million. Hershey Foods blamed a $466 million revenue shortfall on glitches in the firm's ERP rollout. Among the problems, the botched implementation prevented the candy maker from getting product to stores during the critical period before Halloween. Nike's first SCM and ERP implementation was labeled a "disaster"; their systems were blamed for over $100 million in lost sales.[7] Even tech firms aren't immune to software implementation blunders. HP once blamed a $160 million loss on problems with its ERP systems.[8] Manager beware—there are no silver bullets. For insight on the causes of massive software failures, and methods to improve the likelihood of success, see Section 6.

KEY TAKEAWAYS

- Application software focuses on the work of a user or an organization.
- Desktop applications are typically designed for a single user. Enterprise software supports multiple users in an organization or work group.
- Popular categories of enterprise software include ERP (enterprise resource planning), SCM (supply chain management), CRM (customer relationship management), and BI (business intelligence) software, among many others.
- These systems are used in conjunction with database management systems, programs that help firms organize, store, retrieve, and maintain data.
- ERP and other packaged enterprise systems can be challenging and costly to implement, but can help firms create a standard set of procedures and data that can ultimately lower costs and streamline operations.
- The more application software that is available for a platform, the more valuable that platform becomes.
- The DBMS stores and retrieves the data used by the other enterprise applications. Different enterprise systems can be configured to share the same database system in order share common data.
- Firms that don't have common database systems with consistent formats across their enterprise often struggle to efficiently manage their value chain, and often lack the flexibility to introduce new ways of doing business. Firms with common database systems and standards often benefit from increased organizational insight and decision-making capabilities.
- Enterprise systems can cost millions of dollars in software, hardware, development, and consulting fees, and many firms have failed when attempting large-scale enterprise system integration. Simply buying a system does not guarantee its effective deployment and use.
- When set up properly, enterprise systems can save millions of dollars and turbocharge organizations by streamlining processes, making data more usable, and easing the linking of systems with software across the firm and with key business partners.

QUESTIONS AND EXERCISES

1. What is the difference between desktop and enterprise software?
2. Who are the two leading ERP vendors?
3. List the functions of a business that might be impacted by an ERP.
4. What do the acronyms ERP, CRM, SCM, and BI stand for? Briefly describe what each of these enterprise systems does.
5. Where in the "layer cake" analogy does the DBMS lie.
6. Name two companies that have realized multimillion-dollar benefits as result of installing enterprise systems.
7. Name two companies that have suffered multimillion-dollar disasters as result of failed enterprise system installations.
8. How much does the average large company spend annually on ERP software?

4. DISTRIBUTED COMPUTING

LEARNING OBJECTIVES

1. **Understand the concept of distributed computing and its benefits.**
2. **Understand the client-server model of distributed computing.**
3. **Know what Web services are and the benefits that Web services bring to firms.**
4. **Appreciate the importance of messaging standards and understand how sending messages between machines can speed processes, cut costs, reduce errors, and enable new ways of doing business.**

When computers in different locations can communicate with one another, this is often referred to as **distributed computing**. Distributed computing can yield enormous efficiencies in speed, error reduction, and cost savings and can create entirely new ways of doing business. Designing systems architecture for distributed systems involves many advanced technical topics. Rather than provide an exhaustive decomposition of distributed computing, the examples that follow are meant to help managers understand the bigger ideas behind some of the terms that they are likely to encounter.

distributed computing
A form of computing where systems in different locations communicate and collaborate to complete a task.

Let's start with the term **server**. This is a tricky one because it's frequently used in two ways: (1) in a hardware context a server is a computer that has been configured to support requests from other computers (e.g., Dell sells servers) and (2) in a software context a server is a program that fulfills requests (e.g., the Apache open source Web server). Most of the time, server *software* resides on server-class *hardware*, but you can also set up a PC, laptop, or other small computer to run server software, albeit less powerfully. And you can use mainframe or super-computer-class machines as servers, too. Also note that many firms chose not to own some of their applications or any of their own server hardware at all. Instead, they pay third-party firms to host their software "in the cloud." This option is particularly attractive for smaller firms that can't or don't want to invest in the expense and expertise associated with owning and operating hardware, for firms looking for extra computing capacity, and for firms that want public servers (e.g., Web sites) to be in fast, reliable locations outside of a company's own private network.

server
A program that fulfills the requests of a client.

The World Wide Web, like many other distributed computing services, is what geeks call a *client-server* system. Client-server refers to two pieces of software, a **client** that makes a request, and a server that receives and attempts to fulfill the request. In our WWW scenario, the client is the browser (e.g., Internet Explorer, Chrome, Firefox, Safari). When you type a Web site's address into the location field of your browser, you're telling the client to "go find the Web server software at the address provided, and tell the server to return the Web site requested."

client
A software program that makes requests of a server program.

application server

Software that houses and serves business logic for use (and reuse) by multiple applications.

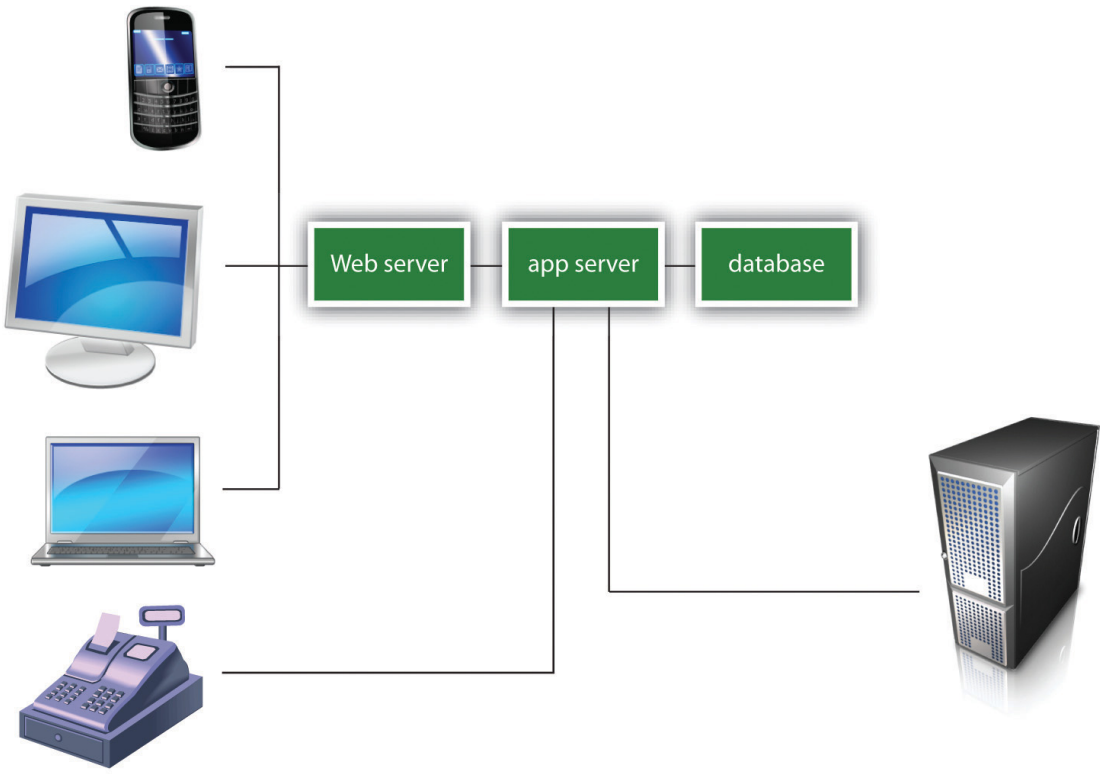
Web services

Small pieces of code that are accessed via the application server which permit interoperable machine-to-machine interaction over a network.

It is possible to link simple scripting languages to a Web server for performing calculations, accessing databases, or customizing Web sites. But more advanced distributed environments may use a category of software called an **application server**. The application server (or app server) houses business logic for a distributed system. Individual **Web services** served up by the app server are programmed to perform different tasks: returning a calculation ("sales tax for your order will be $11.58"), accessing a database program ("here are the results you searched for"), or even making a request to another server in another organization ("Visa, please verify this customer's credit card number for me").

FIGURE 9.6

In this multitiered distributed system, client browsers on various machines (desktop, laptop, mobile) access the system through the Web server. The cash register doesn't use a Web browser, so instead the cash register logic is programmed to directly access the services it needs from the app server. Web services accessed from the app server may be asked to do a variety of functions, including perform calculations, access corporate databases, or even make requests from servers at other firms (for example, to verify a customer's credit card).



application programming interfaces (APIs)

Programming hooks, or guidelines, published by firms that tell other programs how to get a service to perform a task such as send or receive data. For example, Amazon.com provides APIs to let developers write their own applications and Websites that can send the firm orders.

Those little chunks of code that are accessed via the application server are sometimes referred to as Web services. The World Wide Web consortium defines *Web services* as software systems designed to support interoperable machine-to-machine interaction over a network.[9] And when computers can talk together (instead of people), this often results in fewer errors, time savings, cost reductions, and can even create whole new ways of doing business! Each Web service defines the standard method for other programs to request it to perform a task and defines the kind of response the calling client can expect back. These standards are referred to as **application programming interfaces (APIs)**.

Look at the advantages that Web services bring a firm like Amazon. Using Web services, the firm can allow the same order entry logic to be used by Web browsers, mobile phone applications, or even by third parties who want to access Amazon product information and place orders with the firm (there's an incentive to funnel sales to Amazon—the firm will give you a cut of any sales that you send Amazon's way). Google offers many APIs, including hooks that other developers use to leverage Google Maps. And Facebook and Twitter have APIs that, among other things, allow programmers to write apps that can post status updates and tweets. Organizations that have created a robust set of Web services around their processes and procedures are said to have a **service-oriented architecture (SOA)**. Organizing systems like this, with separate applications in charge of client presentation, business logic, and database, makes systems more flexible. Code can be reused, and each layer can be separately maintained, upgraded, or migrated to new hardware—all with little impact on the others.

service-oriented architecture (SOA)

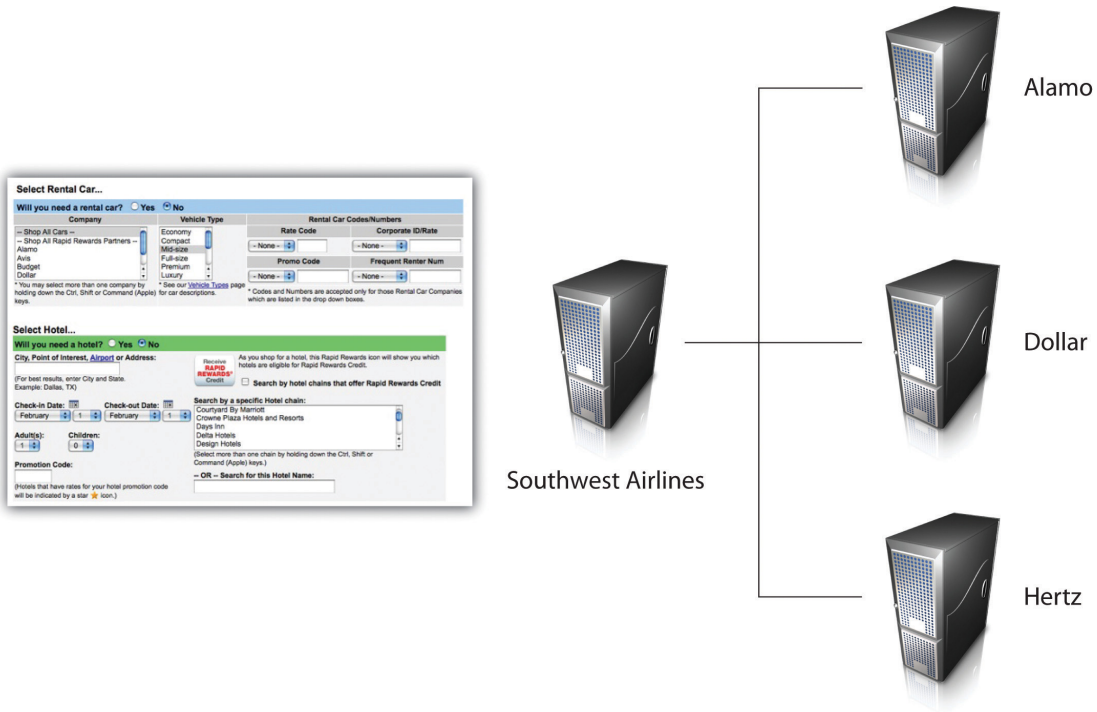
A robust set of Web services built around an organizations processes and procedures.

Web services sound geeky, but here's a concrete example illustrating their power. Southwest Airlines had a Web site where customers could book flights, but many customers also wanted to rent a car or book a hotel, too. To keep customers on Southwest.com, the firm and its hotel and rental car partners created a set of Web services and shared the APIs. Now customers visiting Southwest.com can book a hotel stay and rental car on the same page where they make their flight reservation. This process transforms Southwest.com into a full service travel destination and allows the site to compete head-to-head with the likes of Expedia, Travelocity, and Orbitz.[10]

Think about why Web services are important from a strategic perspective. By adding hotel and rental car services, Southwest is now able to eliminate the travel agent, along with any fees they might share with the agent. This shortcut allows the firm to capture more profits or pass on savings to customers, securing its position as the first place customers go for low-cost travel. And perhaps most importantly, Southwest can capture key data from visitor travel searches and bookings (something it likely couldn't do if customers went to a site like Expedia or Travelocity). Data is a hugely valuable asset, and this kind of customer data can be used by Southwest to send out custom e-mail messages and other marketing campaigns to bring customers back to the airline. As geeky as they might at first seem, Web services can be very strategic!

FIGURE 9.7

Southwest.com uses Web services to allow car rental and hotel firms to book services through Southwest. This process transforms Southwest.com into a full-service online travel agent.



4.1 Formats to Facilitate Sharing Data

EDI (electronic data interchange)

A set of standards for exchanging messages containing formatted data between computer applications.

Two additional terms you might hear within the context of distributed computing are EDI and XML. **EDI (electronic data interchange)** is a set of standards for exchanging information between computer applications. EDI is most often used as a way to send the electronic equivalent of structured documents between different organizations. Using EDI, each element in the electronic document, such as a firm name, address, or customer number, is coded so that it can be recognized by the receiving computer program. Eliminating paper documents makes businesses faster and lowers data entry and error costs. One study showed that firms that used EDI decreased their error rates by 82 percent, and their cost of producing each document fell by up to 96 percent.[11]

extensible markup language (XML)

A tagging language that can be used to identify data fields made available for use by other applications. Most APIs and Web services send messages where the data exchanged is wrapped in identifying XML tags.

EDI is a very old standard, with roots stretching back to the 1948 Berlin Air Lift. While still in use, a new generation of more-flexible technologies for specifying data standards are taking its place. Chief among the technologies replacing EDI is **extensible markup language (XML)**. XML has lots of uses, but in the context of distributed systems, it allows software developers to create a set of standards for common data elements that, like EDI messages, can be sent between different kinds of computers, different applications, and different organizations. XML is often thought of as easier to code than EDI, and it's more robust because it can be extended—organizations can create formats to represent any kind of data (e.g., a common part number, photos, the complaint field collected by customer support personnel). In fact, most messages sent between Web services are coded in XML (the technology is a key enabler in mash-ups, discussed in Chapter 7). Many computer programs also use XML as a way to export and import data in a common format that can be used regardless of the kind of computer hardware, operating system, or application program used. And if you design Web sites, you might encounter XML as part of the coding behind the cascading style sheets (CSS) that help maintain a consistent look and feel to the various Web pages in a given Web site.

Rearden Commerce: A Business Built on Web Services

Web services, APIs, and open standards not only transform businesses, they can create entire new firms that change how we get things done. For a look at the mashed-up, integrated, hyperautomated possibilities that Web services make possible, check out Rearden Commerce, a Foster City, California, firm that is using this technology to become what AMR's Chief Research Office referred to as "Travelocity on Steroids."

Using Rearden, firms can offer their busy employees a sort of Web-based concierge/personal assistant. Rearden offers firms a one-stop shop where employees can not only make the flight, car, and hotel bookings they might do from a travel agent, they can also book dinner reservations, sports and theatre tickets, and arrange for business services like conference calls and package shipping. Rearden doesn't supply the goods and services it sells. Instead it acts as the middleman between transactions. A set of open APIs to its Web services allows Rearden's one hundred and sixty thousand suppliers to send product and service data to Rearden, and to receive booking and sales data from the site.

In this ultimate business mash-up, a mobile Rearden user could use her phone to book a flight into a client city, see restaurants within a certain distance of her client's office, have these locations pop up on a Google map, have listings accompanied by Zagat ratings and cuisine type, book restaurant reservations through Open Table, arrange for a car and driver to meet her at her client's office at a specific time, and sync up these reservations with her firm's corporate calendaring systems. If something unexpected comes up, like a flight delay, Rearden will be sure she gets the message. The system will keep track of any cancelled reservation credits, and also records travel reward programs, so Rearden can be used to spend those points in the future.

In order to pull off this effort, the Rearden maestros are not only skilled at technical orchestration, but also in coordinating customer and supplier requirements. As *TechCrunch*'s Erick Schonfeld put it, "The hard part is not only the technology—which is all about integrating an unruly mess of APIs and Web services—[it also involves] signing commercially binding service level agreements with [now over 160,000] merchants across the world." For its efforts, Rearden gets to keep between 6 percent and 25 percent of every nontravel dollar spent, depending on the service. The firm also makes money from subscriptions, and distribution deals.

The firm's first customers were large businesses and included ConAgra, GlaxoSmithKline, and Motorola. Rearden's customers can configure the system around special parameters unique to each firm: to favor a specific airline, benefit from a corporate discount, or to restrict some offerings for approved employees only. Rearden investors include JPMorgan Chase and American Express—both of whom offer Rearden to their employees and customers. Even before the consumer version was available, Rearden had over four thousand corporate customers and two million total users, a user base larger than better-known firms like Salesforce.com.[12] For all the pizzazz we recognize that, as a start-up, the future of Rearden Commerce remains uncertain; however, the firm's effective use of Web services illustrates the business possibilities as technologies allow firms to connect with greater ease and efficiency.

Connectivity has made our systems more productive and enables entire new strategies and business models. But these wonderful benefits come at the price of increased risk. When systems are more interconnected, opportunities for infiltration and abuse also increase. Think of it this way—each "connection" opportunity is like adding another door to a building. The more doors that have to be defended, the more difficult security becomes. It should be no surprise that the rise of the Internet and distributed computing has led to an explosion in security losses by organizations worldwide.

KEY TAKEAWAYS

- Client-server computing is a method of distributed computing where one program (a client) makes a request to be fulfilled by another program (a server).
- Server is a tricky term and is sometimes used to refer to hardware. While server-class hardware refers to more powerful computers designed to support multiple users, just about any PC or notebook can be configured to run server software. Many firms chose to have their server software hosted "in the cloud" on the computers of third-party firms.
- Web servers serve up Web sites and can perform some scripting.
- Most firms serve complex business logic from an application server.
- Isolating a system's logic in three or more layers (presentation or user interface, business logic, and database) can allow a firm flexibility in maintenance, reusability, and in handling upgrades.
- Web services allow different applications to communicate with one another. APIs define the method to call a Web service (e.g., to get it to do something), and the kind of response the calling program can expect back.
- Web services make it easier to link applications as distributed systems, and can make it easier for firms to link their systems across organizations.
- Popular messaging standards include EDI (older) and XML. Sending messages between machines instead of physical documents can speed processes, drastically cut the cost of transactions, and reduce errors.
- Distributed computing can yield enormous efficiencies in speed, error reduction, and cost savings and can create entirely new ways of doing business.
- When computers can communicate with each other (instead of people), this often results in fewer errors, time savings, cost reductions, and can even create whole new ways of doing business.
- Web services, APIs, and open standards not only transform businesses, they can create entire new firms that change how we get things done.

QUESTIONS AND EXERCISES

1. Differentiate the term "server" used in a hardware context, from "server" used in a software context.
2. Describe the "client-server" model of distributed computing. What products that you use would classify as leveraging client-server computing?
3. List the advantages that Web services have brought to Amazon.
4. How has Southwest Airlines utilized Web services to its competitive advantage?
5. What is Rearden Commerce and which technologies does it employ? Describe Rearden Technology's revenue model. Who were Rearden Technology's first customers? Who were among their first investors?
6. What are the security risks associated with connectivity, the Internet, and distributed processing?

5. WRITING SOFTWARE

LEARNING OBJECTIVES

1. Understand, at a managerial level, what programming languages are and how software is developed.
2. Recognize that an operating system and microprocessor constrain the platform upon which most compiled application software will run.
3. Understand what Java is and why it is significant.
4. Know what scripting languages are.

programming language

Provides the standards, syntax, statements, and instructions for writing computer software.

So you've got a great idea that you want to express in software—how do you go about creating a program? Programmers write software in a **programming language**. While each language has its strengths and weaknesses, most commercial software is written in a variant of the C programming language such as C++ (pronounced "see plus plus"), C# (pronounced "see sharp"), or Objective C (popular for iOS app development). Visual Basic (from Microsoft) and Java (from Sun) are also among the more popular of the dozens of programming languages available. Web developers may favor specialty languages like Ruby and Python, while languages like SQL are used in databases.

integrated development environment (IDE)

An application that includes an editor (a sort of programmer's word processor), debugger, and compiler, among other tools.

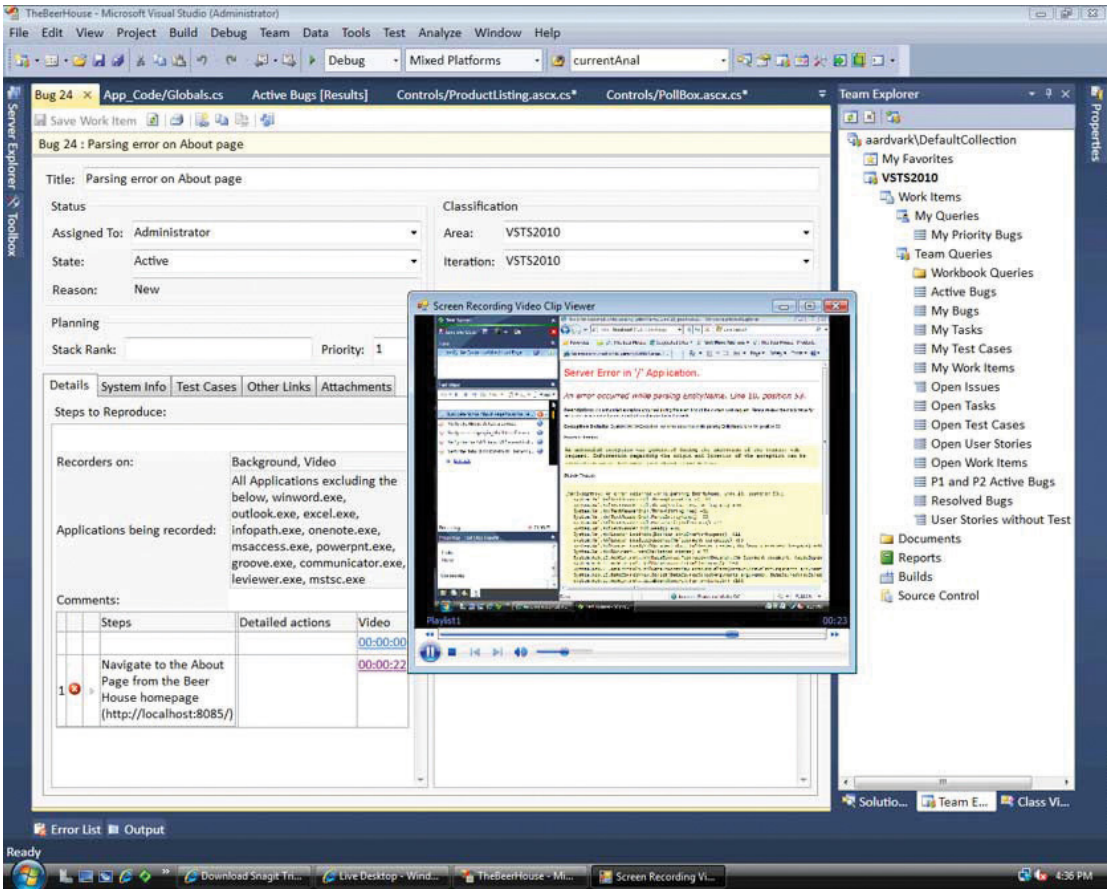
compile

Step in which program code written in a language that humans can more easily understand, is then converted into a form (expressed in patterns of ones and zeros) that can be understood and executed by a microprocessor. Programmers using conventional programming languages must compile their software before making it available for execution.

Most professional programmers use an **integrated development environment (IDE)** to write their code. The IDE includes a text editor, a debugger for sleuthing out errors, and other useful programming tools. The most popular IDE for Windows is Visual Studio, while Apple offers the Xcode IDE. Most IDEs can support several different programming languages. The IDE will also **compile** a programmer's code, turning the higher-level lines of instructions that are readable by humans into lower-level instructions expressed as the patterns of ones and zeros that are readable by a computer's microprocessor.

FIGURE 9.8

Microsoft's Visual Studio IDE supports desktop, server, mobile, and cloud computing software development.



Look at the side of a box of commercial software and you're likely to see system requirements that specify the operating system and processor that the software is designed for (e.g., "this software works on computers with Windows 7 and Intel-compatible processors"). Wouldn't it be great if software could be written once and run everywhere? That's the idea behind **Java**—a programming language developed by Sun Microsystems.

Java programmers don't write code with specific operating system commands (say for Windows, Mac OS X, or Linux), instead they use special Java commands to control their user interface or interact with the display and other hardware. Java programs can run on any computer that has a Java Virtual Machine (JVM), a software layer that interprets Java code so that it can be understood by the operating system and processor of a given computer. Java's platform independence—the ability for developers to "write once, run everywhere"—is its biggest selling point. Many Web sites execute Java applets to run the animation you might see in advertisements or games. Java has also been deployed on over six billion mobile phones worldwide, and is popular among enterprise programmers who want to be sure their programs can scale from smaller hardware up to high-end supercomputers. As long as the machine receiving the Java code has a JVM, then the Java application should run. However, Java has not been popular for desktop applications. Since Java isn't optimized to take advantage of interface elements specific to the Mac or Windows, most Java desktop applications look clunky and unnatural. Java code that runs through the JVM interpreter is also slower than code compiled for the native OS and processor that make up a platform.[13]

Scripting languages are the final category of programming tool that we'll cover. **Scripting languages** typically execute within an application. Microsoft offers a scripting language called VB Script (a derivative of Visual Basic) to automate functions in Office. And most browsers and Web servers support JavaScript, a language that helps make the Web more interactive (despite its name, JavaScript is unrelated to Java). Scripting languages are **interpreted** within their applications, rather than compiled to run directly by a microprocessor. This distinction makes them slower than the kinds of development efforts found in most commercial software. But most scripting languages are usually easy to use, and are often used both by professional programmers and power users.

Java

A programming language, initially developed by Sun Microsystems, designed to provide true platform independence ("write once, run anywhere") for application developers. In most cases, Java apps are developed to be executed by a Java Virtual Machine—an interpreting layer that translates code as it executes, into the format required by the operating system and microprocessor. Without Java, application developers have to write and compile software to execute natively by a specific operating system / microprocessor combination (e.g., Windows/Intel, Linux PowerPC, Mac/Intel, Linux/Intel).

scripting languages

Programming tool that executes within an application. Scripting languages are interpreted within their applications, rather than compiled to run directly by a microprocessor.

interpreted

Languages where each line of written code is converted (by a software program, called an "interpreter") for execution at run-time. Most scripting languages are interpreted languages. Many programmers also write Java applications to be interpreted by the Java Virtual Machine.

KEY TAKEAWAYS

- Programs are often written in a tool called an IDE, an application that includes an editor (a sort of programmer's word processor), debugger, and compiler, among other tools.
- Compiling takes code from the high-level language that humans can understand and converts them into the sets of ones and zeros in patterns representing instructions that microprocessors understand.
- Popular programming languages include C++, C#, Visual Basic, and Java.
- Most software is written for a platform—a combination of an operating system and microprocessor.
- Java is designed to be platform independent. Computers running Java have a separate layer called a Java Virtual Machine that translates (interprets) Java code so that it can be executed on an operating system/processor combination. In theory, Java is "write once, run everywhere," as opposed to conventional applications that are written for an operating system and compiled for an OS/processor combination.
- Java is popular on mobile phones, enterprise computing, and to make Web sites more interactive. Java has never been a successful replacement for desktop applications, largely because user interface differences among the various operating systems are too great to be easily standardized.
- Scripting languages are interpreted languages, such as VB Script or Java Script. Many scripting languages execute within an application (like the Office programs, a Web browser, or to support the functions of a Web server). They are usually easier to program, but are less powerful and execute more slowly than compiled languages.

QUESTIONS AND EXERCISES

1. List popular programming languages.
2. What's an IDE? Why do programmers use IDEs? Name IDEs popular for Windows and Mac users.
3. What is the difference between a compiled programming language and an interpreted programming language?
4. Name one advantage and one disadvantage of scripting languages.
5. In addition to computers, on what other technology has Java been deployed? Why do you suppose Java is particularly attractive for these kinds of applications?
6. What's a JVM? Why do you need it?
7. What if a programmer wrote perfect Java code, but there was a bug on the JVM installed on a given computer? What might happen?
8. Why would developers choose to write applications in Java? Why might they skip Java and choose another programming language?
9. Why isn't Java popular for desktop applications?
10. Go to http://www.java.com. Click on "Do I have Java?" Is Java running on your computer? Which version?

6. UNDERSTANDING TECHNOLOGY BEYOND THE PRICE TAG: TOTAL COST OF OWNERSHIP (TCO) AND THE COST OF TECH FAILURE

LEARNING OBJECTIVES

1. **List the different cost categories that comprise total cost of ownership.**
2. **Understand that once a system is implemented, the costs of maintaining and supporting the system continue.**
3. **List the reasons that technology development projects fail and the measures that can be taken to increase the probability of success.**

Managers should recognize that there are a whole host of costs that are associated with creating and supporting an organization's information systems. Of course, there are programming costs for custom software as well as purchase, configuration, and licensing costs for packaged software, but there's much, much more.

There are costs associated with design and documentation (both for programmers and for users). There are also testing costs. New programs should be tested thoroughly across the various types of hardware the firm uses, and in conjunction with existing software and systems, *before* being deployed throughout the organization. Any errors that aren't caught can slow down a business or lead to costly mistakes that could ripple throughout an organization and its partners. Studies have shown that errors not caught before deployment could be one hundred times more costly to correct than if they were detected and corrected beforehand.[14]

Once a system is "turned on," the work doesn't end there. Firms need to constantly engage in a host of activities to support the system that may also include the following:

compliance
Ensuring that an organization's systems operate within required legal constraints, and industry and organizational obligations

- providing training and end user support
- collecting and relaying comments for system improvements
- auditing systems to ensure **compliance** (i.e., that the system operates within the firm's legal constraints and industry obligations)
- providing regular backup of critical data
- planning for redundancy and disaster recovery in case of an outage
- vigilantly managing the moving target of computer security issues

With so much to do, it's no wonder that firms spend 70 to 80 percent of their information systems (IS) budgets just to keep their systems running.[15] The price tag and complexity of these tasks can push some managers to think of technology as being a cost sink rather than a strategic resource. These tasks are often collectively referred to as the **total cost of ownership (TCO)** of an information system. Understanding TCO is critical when making technology investment decisions. TCO is also a major driving force behind the massive tech industry changes discussed in Chapter 10.

total cost of ownership (TCO)

All of the costs associated with the design, development, testing, implementation, documentation, training and maintenance of a software system.

6.1 Why Do Technology Projects Fail?

Even though information systems represent the largest portion of capital spending at most firms, an astonishing one in three technology development projects fail to be successfully deployed.[16] Imagine if a firm lost its investment in one out of every three land purchases, or when building one in three factories. These statistics are dismal! Writing in *IEEE Spectrum*, risk consultant Robert Charette provides a sobering assessment of the cost of software failures, stating, "The yearly tab for failed and troubled software conservatively runs somewhere from $60 to $70 billion in the United States alone. For that money, you could launch the space shuttle one hundred times, build and deploy the entire 24-satellite Global Positioning System, and develop the Boeing 777 from scratch—and still have a few billion left over."[17]

Why such a bad track record? Sometimes technology itself is to blame, other times it's a failure to test systems adequately, and sometimes it's a breakdown of process and procedures used to set specifications and manage projects. In one example, a multimillion-dollar loss on the NASA Mars Observer was traced back to a laughably simple oversight—Lockheed Martin contractors using English measurements, while the folks at NASA used the metric system.[18] Yes, a $125 million taxpayer investment was lost because a bunch of rocket scientists failed to pay attention to third grade math. When it comes to the success or failure of technical projects, the devil really is in the details.

Projects rarely fail for just one reason. Project post-mortems often point to a combination of technical, project management, and business decision blunders. The most common factors include the following:[19]

- Unrealistic or unclear project goals
- Poor project leadership and weak executive commitment
- Inaccurate estimates of needed resources
- Badly defined system requirements and allowing "feature creep" during development
- Poor reporting of the project's status
- Poor communication among customers, developers, and users
- Use of immature technology
- Unmanaged risks
- Inability to handle the project's complexity
- Sloppy development and testing practices
- Poor project management
- Stakeholder politics
- Commercial pressures (e.g., leaving inadequate time or encouraging corner-cutting)

Managers need to understand the complexity involved in their technology investments, and that achieving success rarely lies with the strength of the technology alone.

But there is hope. Information systems organizations can work to implement procedures to improve the overall quality of their development practices. Mechanisms for quality improvement include **capability maturity model integration (CMMI)**, which gauge an organization's process maturity and capability in areas critical to developing and deploying technology projects, and provides a carefully chosen set of best practices and guidelines to assist quality and process improvement.[20]

capability maturity model integration (CMMI)

A process-improvement approach (useful for but not limited to software engineering projects) that can assist in assessing the maturity, quality, and development of certain organizational business processes, and suggest steps for their improvement.

Firms are also well served to leverage established project planning and software development methodologies that outline critical businesses processes and stages when executing large-scale software development projects. The idea behind these methodologies is straightforward—why reinvent the wheel when there is an opportunity to learn from and follow blueprints used by those who have executed successful efforts. When methodologies are applied to projects that are framed with clear business goals and business metrics, and that engage committed executive leadership, success rates can improve dramatically.[21]

While software development methodologies are the topic of more advanced technology courses, the savvy manager knows enough to inquire about the development methodologies and quality programs used to support large scale development projects, and can use these investigations as further

input when evaluating whether those overseeing large scale efforts have what it takes to get the job done.

KEY TAKEAWAYS

- The care and feeding of information systems can be complex and expensive. The total cost of ownership of systems can include software development and documentation, or the purchase price and ongoing license and support fees, plus configuration, testing, deployment, maintenance, support, training, compliance auditing, security, backup, and provisions for disaster recovery. These costs are collectively referred to as TCO, or a system's total cost of ownership.
- Information systems development projects fail at a startlingly high rate. Failure reasons can stem from any combination of technical, process, and managerial decisions.
- IS organizations can leverage software development methodologies to improve their systems development procedures, and firms can strive to improve the overall level of procedures used in the organization through models like CMMI. However, it's also critical to engage committed executive leadership in projects, and to frame projects using business metrics and outcomes to improve the chance of success.
- System errors that aren't caught before deployment can slow down a business or lead to costly mistakes that could ripple throughout an organization. Studies have shown that errors not caught before deployment could be 100 times more costly to correct than if they were detected and corrected beforehand.
- Firms spend 70 to 80 percent of their IS budgets just to keep their systems running.
- IS organizations can employ project planning and software development methodologies to implement procedures to improve the overall quality of their development practices.

QUESTIONS AND EXERCISES

1. List the types of total ownership costs associated with creating and supporting an organization's information systems.
2. On average, what percent of firms' IS budgets is spent to keep their systems running?
3. What are the possible effects of not detecting and fixing major system errors before deployment?
4. List some of the reasons for the failure of technology development projects.
5. What is the estimated yearly cost of failed technology development projects?
6. What was the reason attributed to the failure of the NASA Mars Observer project?
7. What is capability maturity model integration (CMMI) and how is it used to improve the overall quality of a firm's development practices?
8. Perform an Internet search for "IBM Rational Portfolio Manager." How might IBM's Rational Portfolio Manager software help companies realize more benefit from their IT systems development project expenditures? What competing versions of this product offered by other organizations?

ENDNOTES

1. R. Charette, "Why Software Fails," *IEEE Spectrum*, September 2005.
2. S. Lacy, "Is Atlassian the Next Big Enterprise Software IPO?" *Pando Daily*, February 22, 2012.
3. The iPhone and iPod touch OS is derived from Apple's Mac OS X operating system.
4. Adapted from G. Edmondson, "Silicon Valley on the Rhine," *BusinessWeek International*, November 3, 1997.
5. A. Robinson and D. Dilts, "OR and ERP," *ORMS Today*, June 1999.
6. C. Rettig, "The Trouble with Enterprise Software," *MIT Sloan Management Review* 49, no. 1 (2007): 21–27.
7. C. Koch, "Nike Rebounds: How (and Why) Nike Recovered from Its Supply Chain Disaster," *CIO*, June 15, 2004.
8. R. Charette, "Why Software Fails," *IEEE Spectrum*, September 2005.
9. W3C, "Web Services Architecture," *W3C Working Group Note*, February 11, 2004.
10. J. McCarthy, "The Standards Body Politic," *InfoWorld*, May 17, 2002.
11. "Petroleum Industry Continues to Explore EDI," *National Petroleum News* 90, no. 12 (November 1998).
12. M. Arrington, "Rearden Commerce: Time for the Adults to Come In and Clean House," *TechCrunch*, April 5, 2007; E. Schonfeld, "At Rearden Commerce, Addiction Is Job One," *TechCrunch*, May 6, 2008; and M. Arrington, "2008: Rearden Commerce Has a Heck of a Year," *TechCrunch*, January 13, 2009.
13. Some offerings have attempted to overcome the speed issues associated with interpreting Java code. Just-in-time compilation stores code in native processor-executable form after each segment is initially interpreted, further helping to speed execution. Other environments allow for Java to be compiled ahead of time so that it can be directly executed by a microprocessor. However, this process eliminates code portability—Java's key selling point. And developers preparing their code for the JVM actually precompile code into something called Java bytecode, a format that's less human friendly but more quickly interpreted by JVM software.
14. R. Charette, "Why Software Fails," *IEEE Spectrum*, September 2005.
15. C. Rettig, "The Trouble with Enterprise Software," *MIT Sloan Management Review* 49, no. 1 (2007): 21—27.
16. L. Dignan, "Survey: One in 3 IT Projects Fail; Management OK with It," *ZDNet*, December 11, 2007.
17. R. Charette, "Why Software Fails," *IEEE Spectrum*, September 2005.
18. R. Lloyd, "Metric Mishap Caused Loss of NASA Orbiter," *CNN*, September 20, 1999.
19. List largely based on R. Charette, "Why Software Fails," *IEEE Spectrum*, September 2005.
20. R. Kay, "QuickStudy: Capability Maturity Model Integration (CMMI)," *Computerworld*, January 24, 2005; and Carnegie Mellon Software Engineering Institute, *Welcome to CMMI*, 2009, http://www.sei.cmu.edu/cmmi.
21. A. Shenhar and D. Dvir, *Reinventing Project Management: The Diamond Approach to Successful Growth and Innovation* (Boston: Harvard Business School Press, 2007).