The Origins of Software

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

● Explain outsourcing.
● Describe six different sources of software.
● Discuss how to evaluate off-the-shelf software.
● Explain reuse and its role in software development.

INTRODUCTION

As you learned in Chapter 1, there was a time, not too long ago, when there were no systems analysts and no symbolic computer programming languages. Yet people still wrote and programmed applications for computers. You read in Chapter 1 about how things have changed over the last 50-plus years. Even though today’s systems analyst has dozens of programming languages and development tools to work with, you could easily argue that systems development is even more difficult now than it was 50 years ago. Then, as well as even more recently, one issue was decided for you: If you wanted to write application software, you did it in-house, and you wrote the software from scratch.

Today, there are many different sources of software, and many of you reading this book will end up working for firms that produce software rather than in the information systems department of a corporation. But for those of you who do go on to work in a corporate information systems department, the focus is no longer exclusively on in-house development. Instead, the focus will be on where to obtain the many pieces and components that you will combine into the application system you have been asked to create. You and your peers will still write code, mainly to make all the different pieces work together, but more and more of your application software will be written by someone
else. Even though you will not write the code, you will still use the basic structure and processes of the systems analysis and design life cycle to build the application systems your organization demands. The organizational process of systems development remains the focus for the rest of the book, but first you need to know more about where software originates in today’s development environment.

In this chapter, you will learn about the various sources of software for organizations. The first source considered is outsourcing, in which all or part of an organization’s information systems, their development, and their maintenance are given over to another organization. You will then read about six different sources of software: (1) information technology services firms, (2) packaged software providers, (3) vendors of enterprise-wide solution software, (4) application service providers and managed service providers, (5) open source software, and (6) the organization itself when it develops software in-house. You will learn of criteria to evaluate software from these different sources. The chapter closes with a discussion of reuse and its impact on software development.

SYSTEMS ACQUISITION

While there will always be some debate about when and where the first administrative information system was developed, there is general agreement that the first such system in the United Kingdom was developed at J. Lyons & Sons. In the United States, the first administrative information system was General Electric’s (GE) payroll system, which was developed in 1954 (Computer History Museum, 2003). At that time, and for many years afterwards, obtaining an information system meant one thing only: in-house development. The software industry itself did not even come into existence until a decade after GE’s payroll system was implemented.

Since GE’s payroll system was built, in-house development has become a progressively smaller piece of all the systems development work that takes place in and for organizations. Internal corporate information systems departments now spend a smaller and smaller proportion of their time and effort on developing systems from scratch. In 1998, corporate information systems groups reported spending 33 percent less time and money on traditional software development and maintenance than they did in 1997 (King and Cole-Gomolski, 1999). Instead, they increased work on packaged applications by a factor of three, and they increased outsourcing by 42 percent. Where in-house development occurred, it was related to Internet technology. Developers probably view Internet-related development as being more challenging and more fun than traditional development.

Organizations today have many choices when seeking an information system. We will start with a discussion of outsourcing development and operation and then move on to a presentation on the sources of software.

Outsourcing

If one organization develops or runs a computer application for another organization, that practice is called outsourcing. Outsourcing includes a spectrum of working arrangements. At one extreme is having a firm develop and run your application on its computers—all you do is supply input and take output. A common example of such an arrangement is a company that runs payroll applications for clients so that clients do not have to develop an independent in-house payroll system. Instead, they simply provide employee payroll information to the company, and, for a fee, the company returns completed paychecks, payroll accounting reports, and tax and other statements for employees. For many organizations, payroll is a very cost-effective operation when outsourced in this way. Another example of outsourcing would be if you hired a company to run your applications at your site on your computers. In some cases, an
organization employing such an arrangement will dissolve some or all of its IS unit and fire all of its information systems employees. Often the company brought in to run the organization’s computing will rehire many of the organization’s original IS unit employees.

Outsourcing is big business. Some organizations outsource the IT development of many of their IT functions, at a cost of billions of dollars. For example, the 20 largest U.S. government IT contracts awarded in fiscal 2006 were estimated to be worth a total of $250 billion (Gross, 2005). The Department of Homeland Security contract for outsourced IT services alone was worth $45 billion. Individual outsourcing vendors also sign large contracts for their services. IBM and EDS are two of the biggest, best-known outsourcing firms. Both companies have multiple outsourcing contracts in place with many different firms. IBM’s clients include American Express, with a contract worth $4 billion, and Qwest Communications, with a contract worth $2 billion. Outsourcing is not risk-free, however. JP Morgan, the giant financial firm, cancelled a $5 billion outsourcing contract with IBM (Cowley, 2004), and EDS has had well-publicized problems with its $7 billion outsourcing contract with the U.S. Navy.

Why would an organization outsource its information systems operations? As we saw in the payroll example, outsourcing may be cost effective. If a company specializes in running payroll for other companies, it can leverage the economies of scale it achieves from running one stable computer application for many organizations into very low prices. Outsourcing also provides a way for firms to leapfrog their current position in information systems and to turn over development and operations to outside staff who possess knowledge and skills not found internally (Ketler and Willems, 1999). Other reasons for outsourcing include:

- Freeing up internal resources
- Increasing the revenue potential of the organization
- Reducing time to market
- Increasing process efficiencies
- Outsourcing noncore activities

An organization may move to outsourcing and dissolve its entire information processing unit for political reasons as well, such as overcoming operating problems the organization faces in its information systems unit. For example, the city of Grand Rapids, Michigan, hired an outside firm to run its computing center 30 years ago in order to better manage its computing center employees. Union contracts and civil service constraints then in force made it difficult to fire people, so the city brought in a facilities management organization to run its computing operations, and it was able to get rid of problem employees at the same time. As mentioned earlier, another reason for total outsourcing is that an organization’s management may feel its core mission does not involve managing an information systems unit and that it might achieve more effective computing by turning over all of its operations to a more experienced, computer-oriented company. Kodak decided in the late 1980s that it was not in the computer applications business and turned over management of its mainframes to IBM and management of its personal computers to Businessland (Applegate and Montealagre, 1991).

Outsourcing is an alternative analysts need to be aware of. When generating alternative system development strategies for a system, as an analyst you should consult organizations in your area that provide outsourcing services. It may well be that at least one such organization has already developed and is running an application very close to what your users are asking for. Perhaps outsourcing the replacement system should be one of your alternatives. Knowing what your system requirements are before you consider outsourcing means that you can carefully assess how well the suppliers of outsourcing services can respond to your needs. However, should you decide
not to consider outsourcing, you need to determine whether some software components of your replacement system should be purchased and not built in-house.

**Sources of Software**

We can group the sources of software into six major categories: information technology services firms, packaged software producers, enterprise-wide solutions, application service providers, open source software, and in-house developers (Figure 2-1). These various sources represent points along a continuum of options, with many hybrid combinations along the way.

**Information Technology Services Firms** If a company needs an information system but does not have the expertise or the personnel to develop the system in-house, and a suitable off-the-shelf system is not available, the company will likely consult an information technology (IT) services firm. IT services firms help companies develop custom information systems for internal use, or they develop, host, and run applications for customers, or they provide other services. Note in Table 2-1, a list of the Top 10 global software firms, that five out of ten specialize in services, which include custom systems development. These firms employ people with expertise in the development of information systems. Their consultants may also have expertise in a given business area. For example, consultants who work with banks understand financial institutions as well as information systems. Consultants use many of the same methodologies, techniques, and tools that companies use to develop systems in-house.

It may surprise you to see IBM listed as the top global software producer; you may think of it as primarily a hardware company. Yet IBM has been moving away from a reliance on hardware development for many years. The purchase of the IT consulting arm of PricewaterhouseCoopers by IBM in 2002 solidified its move into services and consulting. IBM is also well-known for its development of Web server and mid-
TABLE 2-1 The 2005 Top 10 Global Software Companies

<table>
<thead>
<tr>
<th>Rank</th>
<th>Company</th>
<th>2005 Software/Services Revenue (million USD)</th>
<th>Software Business Sector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IBM</td>
<td>$61,307</td>
<td>Middleware/Application Server/Web Server/ Systems Integration Services/IT Consulting</td>
</tr>
<tr>
<td>2</td>
<td>Microsoft</td>
<td>$33,969</td>
<td>Operating Systems</td>
</tr>
<tr>
<td>3</td>
<td>EDS</td>
<td>$20,669</td>
<td>IT Sourcing</td>
</tr>
<tr>
<td>4</td>
<td>Computer Sciences Corp.</td>
<td>$15,188</td>
<td>Systems Integration Services/IT Consulting</td>
</tr>
<tr>
<td>5</td>
<td>Accenture</td>
<td>$15,114</td>
<td>Systems Integration Services/IT Consulting</td>
</tr>
<tr>
<td>6</td>
<td>HP</td>
<td>$13,778</td>
<td>Systems Integration Services/IT Consulting</td>
</tr>
<tr>
<td>7</td>
<td>Oracle</td>
<td>$10,156</td>
<td>Database/Business Process Management</td>
</tr>
<tr>
<td>8</td>
<td>Hitachi</td>
<td>$9,491</td>
<td>Telecommunication Services</td>
</tr>
<tr>
<td>9</td>
<td>SAP America</td>
<td>$9,313</td>
<td>Business Process Management</td>
</tr>
<tr>
<td>10</td>
<td>Capgemini</td>
<td>$8,581</td>
<td>Systems Integration Services/IT Consulting</td>
</tr>
</tbody>
</table>

Note: All figures in U.S. dollars.
(Source: www.softwaremag.com. Used with permission.)

dleware software. Other leading IT services firms include traditional consulting firms, such as Computer Sciences Corp., Accenture, and Capgemini. The list also includes HP (Hewlett-Packard), another company formerly focused on hardware that has made the transition to an IT services firm.

**Packaged Software Producers** The growth of the software industry has been phenomenal since its beginnings in the mid-1960s. Some of the largest computer companies in the world are companies that produce software exclusively. A good example is Microsoft, which is number two in the Top 10 list shown in Table 2-1. Almost 98 percent of Microsoft’s revenue comes from its software sales, mostly for its Windows operating systems and its personal productivity software, the Microsoft Office Suite. Number seven on the list, Oracle, is exclusively a software company known primarily for its database software, but Oracle also makes enterprise systems. The ninth company on the list, SAP America, is also a software-focused company which develops enterprise-wide system solutions. You will read more about Oracle and SAP shortly, in the section on enterprise systems.

Software companies develop what are sometimes called prepackaged or off-the-shelf systems. Microsoft’s Project (Figure 2-2) and Intuit’s Quicken, QuickPay, and QuickBooks are popular examples of such software. The packaged software development industry serves many market segments. Their software offerings range from general, broad-based packages, such as general ledgers, to very narrow, niche packages, such as software to help manage a day care center. Software companies develop software to run on many different computer platforms, from microcomputers to large mainframes. The companies range in size from just a few people to thousands of employees.

Software companies consult with system users after the initial software design has been completed and an early version of the system has been built. The systems are then tested in actual organizations to determine whether there are any problems or if any improvements can be made. Until testing is completed, the system is not offered for sale to the public.

Some off-the-shelf software systems cannot be modified to meet the specific, individual needs of a particular organization. Such application systems are sometimes
Figure 2-2
Microsoft Project

called turnkey systems. The producer of a turnkey system will only make changes to
the software when a substantial number of users ask for a specific change. However,
other off-the-shelf application software can be modified or extended, by the pro-
ducer or by the user, to more closely fit the needs of the organization. Even though
many organizations perform similar functions, no two organizations do the same
thing in quite the same way. A turnkey system may be good enough for a certain level
of performance, but it will never perfectly match the way a given organization does
business. A reasonable estimate is that off-the-shelf software can at best meet 70 per-
cent of an organization’s needs. Thus, even in the best case, 30 percent of the soft-
ware system does not match the organization’s specifications.

Enterprise Solutions Software
As mentioned in Chapter 1, more and more orga-
nizations are choosing complete software solutions, called enterprise solutions or
enterprise resource planning (ERP) systems, to support their operations and business
processes. These ERP software solutions consist of a series of integrated modules.
Each module supports an individual, traditional business function, such as account-
ing, distribution, manufacturing, or human resources. The difference between the
modules and traditional approaches is that the modules are integrated to focus on
business processes rather than on business functional areas. For example, a series of
modules will support the entire order entry process, from receiving an order to
adjusting inventory to shipping to billing to after-the-sale service. The traditional
approach would use different systems in different functional areas of the business,
such as a billing system in accounting and an inventory system in the warehouse.
Using enterprise software solutions, a firm can integrate all parts of a business
process in a unified information system. All aspects of a single transaction occur
seamlessly within a single information system, rather than as a series of disjointed,
separate systems focused on business functional areas.

The benefits of the enterprise solutions approach include a single repository of
data for all aspects of a business process and the flexibility of the modules. A single
repository ensures more consistent and accurate data, as well as less maintenance.
The modules are flexible because additional modules can be added as needed once
the basic system is in place. Added modules are immediately integrated into the existing system. However, there are disadvantages to enterprise solutions software. The systems are very complex, so implementation can take a long time to complete. Organizations typically do not have the necessary expertise in-house to implement the systems, so they must rely on consultants or employees of the software vendor, which can be very expensive. In some cases, organizations must change how they do business in order to benefit from a migration to enterprise solutions.

Several major vendors provide enterprise solution software. The best known is probably SAP AG, the German firm mentioned earlier, known for its flagship product R/3. SAP stands for Systems, Applications, and Products in Data Processing. SAP AG was founded in 1972, but most of its growth has occurred since 1992. In 2005, SAP America was the ninth-largest supplier of software in the world (see Table 2-1). The other major vendor of enterprise solutions is Oracle Corp., also a U.S.-based firm, perhaps better known for its database software. Oracle is seventh on the list of the Top 10 software companies for 2005 (Table 2-1). Oracle captured a large share of the ERP market through its own financial systems and through the acquisition of other ERP vendors. At the end of 2004, Oracle acquired PeopleSoft, Inc., a U.S. firm founded in 1987. PeopleSoft began with enterprise solutions that focused on human resources management and expanded to cover financials, materials management, distribution, and manufacturing before Oracle acquired them. Just before being purchased by Oracle, PeopleSoft had boosted its corporate strength in 2003 through acquiring another ERP vendor, J.D. Edwards. Together, SAP and Oracle control about 60 percent of the ERP market, which was estimated at around $24 billion in revenues for 2005. The market for ERP is predicted to grow in the range of 6 to 7 percent per year through 2009 (Woodie, 2005). Because the higher end of the market has become saturated with ERP systems, most ERP vendors are looking to medium and small businesses for growth. For example, Oracle’s offering for medium and small businesses is called Oracle’s Business Suite Special Edition (Figure 2-3).

Application Service Providers and Managed Service Providers Another method for organizations to obtain applications is to rent them or license them from third-party providers who run the applications at remote sites. Users have access to the

---

**Figure 2-3**

Functional areas supported by Oracle’s Business Suite for small to medium businesses

applications through the Internet or through virtual private networks. The application provider buys, installs, maintains, and upgrades the applications. Users pay on a per-use basis or they license the software, typically month to month. The companies that host the applications are called application service providers, or ASPs.

Although ASPs sometimes develop and rent or license their own applications to customers, for the most part ASPs purchase or license applications from other software vendors. For example, Oracle and Microsoft make their applications available through ASPs (Holohan and Hall, 2000; Wilcox and Farmer, 2000). Microsoft offers its Windows operating system and Office software to ASPs, whereas Oracle offers its ERP applications.

As a business grows or changes, however, using an application through an ASP becomes limiting. Demands of suppliers, partners, and customers necessitate changes in the application and how it is run. Responding to the need for growth and flexibility is a new generation of providers called managed service providers (MSPs). An MSP can offer customized applications and also include business processes, engineering, security, and maintenance (Computerworld, 2005). As is true of ASPs, MSP customers can pay based on a monthly charge on a per-use basis, and like ASPs, MSPs can provide services from outside a company’s network. The key difference is in the extras the MSP offers. Instead of providing access to a commodity application, MSPs offer network-based services, customized applications, and even equipment, for a fee.

As of late 2005, about one-third of companies used an ASP or MSP, and another 22 percent planned to use a provider by the end of 2007 (Computerworld, 2005). Some analysts have predicted that by 2010, approximately 80 percent of all corporate applications will be hosted (Harney, 2000). As these growth forecasts indicate, taking the provider route has its advantages. The top three reasons for choosing to go with a provider, all of which result in benefits for the company, are: (1) freeing internal IT staff, (2) gaining access to applications faster than via internal development, and (3) achieving lower-cost access to corporate-quality applications (Computerworld, 2005). Especially appealing is the ability to gain access to large and complex systems without having to go through the expensive and time-consuming process of implementing the systems themselves in-house. Using a provider also makes it easier to walk away from an unsatisfactory systems solution.

Open Source Software Open source software is unlike the other types of software you have read about so far. Open source software is different because it is freely available, not just the final product but the source code itself. It is also different because it is developed by a community of interested people instead of by employees of a particular company. Open source software performs the same functions as commercial software, such as operating systems, e-mail, database systems, Web browsers, and so on. Some of the most well-known and popular open source software names are Linux, the operating system; mySQL, a database system; and Firefox, a Web browser. Open source also applies to software components and objects. Open source is developed and maintained by communities of people, and sometimes these communities can be very large. Developers often use common Web resources, such as SourceForge.net, to organize their activities. In May 2006, SourceForge.net hosted almost 120,000 projects and had over 1.3 million registered users. There is no question that the open source movement would not be having the success it enjoys without the availability of the Internet for providing access and organizing development activities.

If the software is free, you might wonder how anybody makes any money by developing open source software. There are two primary ways companies and individuals can make money with open source: (1) by providing maintenance and other services or (2) by providing one version of the software free and selling a more fully
Some open source solutions have more of an impact on the software industry than others. Linux, for example, has been very successful in the server software market, where it is estimated to have 24 percent of the market share now, a number that is projected to grow to 33 percent by 2007. In the desktop operating systems, Linux has 3 percent market share now, projected to double to 6 percent by 2007. These market shares together translate into a market value of $11 billion for Linux currently, with the potential for the market value to grow to $35.7 billion by 2008 (Coy, 2004). Other open source software products, such as mySQL, have also been successful, and open source’s share of the software industry seems destined to continue to grow.

**In-House Development** We have talked about several different types of external organizations that serve as sources of software, but in-house development remains an option. In-house development has become a progressively smaller piece of all systems development work that takes place in and for organizations. As you read earlier in this chapter, internal corporate information systems departments now spend a smaller and smaller proportion of their time and effort on developing systems from scratch. According to a recent study (Banker et al., 1998), in-house development can lead to a larger maintenance burden than other development methods, such as packaged applications. The study found that using a code generator as the basis for in-house development was related to an increase in maintenance hours, whereas using packaged applications was associated with a decrease in maintenance effort.

Of course, in-house development need not entail development of all of the software that will comprise the total system. Hybrid solutions involving some purchased and some in-house software components are common. If you choose to acquire software from outside sources, this choice is made at the end of the analysis phase. The choice between a package and an external supplier will be determined by your needs, not by what the supplier has to sell. As we will discuss, the results of your analysis study will define the type of product you want to buy and will make working with an external supplier much easier, more productive, and worthwhile. Table 2-2 compares the six different software sources discussed in this section.

**TABLE 2-2 Comparison of Six Different Sources of Software Components**

<table>
<thead>
<tr>
<th>Producers</th>
<th>When to Go to This Type of Organization for Software</th>
<th>Internal Staffing Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT services firms</td>
<td>When task requires custom support and system can’t be built internally or system needs to be sourced</td>
<td>Internal staff may be needed, depending on application</td>
</tr>
<tr>
<td>Packaged software producers</td>
<td>When supported task is generic</td>
<td>Some IS and user staff to define requirements and evaluate packages</td>
</tr>
<tr>
<td>Enterprise-wide solutions vendors</td>
<td>For complete systems that cross functional boundaries</td>
<td>Some internal staff necessary but mostly need consultants</td>
</tr>
<tr>
<td>Application service providers and/or managed service providers</td>
<td>For instant access to an application; when supported task is generic (ASP only)</td>
<td>Few; both ASP and MSP use free up staff for other IT work</td>
</tr>
<tr>
<td>Open source software</td>
<td>When supported task is generic but cost is an issue</td>
<td>Some IS and user staff to define requirements and evaluate packages</td>
</tr>
<tr>
<td>In-house developers</td>
<td>When resources and staff are available and system must be built from scratch</td>
<td>Internal staff necessary though staff size may vary</td>
</tr>
</tbody>
</table>
Choosing Off-the-Shelf Software

Once you have decided to purchase off-the-shelf software rather than write some or all of the software for your new system, how do you decide what to buy? There are several criteria to consider, and special criteria may arise with each potential software purchase. For each criterion, an explicit comparison should be made between the software package and the process of developing the same application in-house. The most common criteria are:

- Cost
- Functionality
- Vendor support
- Viability of vendor
- Flexibility
- Documentation
- Response time
- Ease of installation

These criteria are presented in no particular order. The relative importance of the criteria will vary from project to project and from organization to organization. If you had to choose two criteria that would always be among the most important, those two would probably be vendor viability and vendor support. You do not want to get involved with a vendor that might not be in business tomorrow. Similarly, you do not want to license software from a vendor with a reputation for poor support. How you rank the importance of the remaining criteria will very much depend on the specific situation in which you find yourself.

Cost involves comparing the cost of developing the same system in-house with the cost of purchasing or licensing the software package. You should include a comparison of the cost of purchasing vendor upgrades or annual license fees with the costs you would incur to maintain your own software. Costs for purchasing and developing in-house can be compared based on economic feasibility measures (e.g., a present value can be calculated for the cash flow associated with each alternative).

Functionality refers to the tasks the software can perform and the mandatory, essential, and desired system features. Can the software package perform all or just some of the tasks your users need? If only some, can it perform the necessary core tasks? Note that meeting user requirements occurs at the end of the analysis phase because you cannot evaluate packaged software until user requirements have been gathered and structured. Purchasing application software is not a substitute for conducting the systems analysis phase; rather, purchasing software is part of one design strategy for acquiring the system identified during analysis.

As we said earlier, vendor support refers to whether and how much support the vendor can provide. Support occurs in the form of assistance to install the software, to train user and systems staff on the software, and to provide help as problems arise after installation. Recently, many software companies have significantly reduced the amount of free support they will provide customers, so the cost to use telephone, onsite, fax, or computer bulletin board support facilities should be considered. Related to support is the vendor’s viability. You do not want to get stuck with software developed by a vendor that might go out of business soon. This latter point should not be minimized. The software industry is quite dynamic, and innovative application software is created by entrepreneurs working from home offices—the classic cottage industry. Such organizations, even with outstanding software, often do not have the resources or business management ability to stay in business very long. Further, competitive moves by major software firms can render the products of smaller firms outdated or incompatible with operating systems. One software firm we talked to while...
developing this book was struggling to survive just trying to make its software work on any supposedly IBM-compatible PC (given the infinite combination of video boards, monitors, BIOS chips, and other components). Keeping up with hardware and system software changes may be more than a small firm can handle, and good off-the-shelf application software can be lost.

Flexibility refers to how easy it is for you, or the vendor, to customize the software. If the software is not very flexible, your users may have to adapt the way they work to fit the software. Are they likely to adapt in this manner? Purchased software can be modified in several ways. Sometimes the vendor will be willing to make custom changes for you, if you are willing to pay for the redesign and programming. Some vendors design the software for customization. For example, the software may include several different ways of processing data and, at installation time, the customer chooses which to initiate. Also, displays and reports may be easily redesigned if these modules are written in a fourth-generation language. Reports, forms, and displays may be easily customized using a process whereby your company name and chosen titles for reports, displays, forms, column headings, and so forth are selected from a table of parameters you provide. You may want to employ some of these same customization techniques for systems developed in-house so that the software can be easily adapted for different business units, product lines, or departments.

Documentation includes the user’s manual as well as technical documentation. How understandable and up to date is the documentation? What is the cost for multiple copies, if required? Response time refers to how long it takes the software package to respond to the user’s requests in an interactive session. Another measure of time would be how long it takes the software to complete running a job. Finally, ease of installation is a measure of the difficulty of loading the software and making it operational.

Validating Purchased Software Information

One way to get all of the information you want about a software package is to collect it from the vendor. Some of this information may be contained in the software documentation and technical marketing literature. Other information can be provided upon request. For example, you can send prospective vendors a questionnaire asking specific questions about their packages. This may be part of a request for proposal (RFP) or a request for quote (RFQ) your organization requires when major purchases are made. Space does not permit us to discuss the topic of RFPs and RFQs here; you may wish to refer to purchasing and marketing texts if you are unfamiliar with such processes (additional references about RFPs and RFQs are found at the end of this chapter).

There is, of course, no replacement for actually using the software yourself and running it through a series of tests based on your software selection criteria. Remember to test not only the software, but also the documentation, training materials, and even the technical support facilities. One requirement you can place on prospective software vendors as part of the bidding process is that they install (free or at an agreed-upon cost) their software for a limited amount of time on your computers. This way you can determine how their software works in your environment, not in some optimized environment they have for demonstration purposes.

One of the most reliable and insightful sources is other users of the software. Vendors will usually provide a list of customers (remember, they will naturally tell you about satisfied customers, so you may have to probe for a cross section of customers) and people who are willing to be contacted by prospective customers. And here is where your personal network of contacts, developed through professional groups, college friends, trade associations, or local business clubs, can be a resource; do not hesitate to find some contacts on your own. Such current or former customers can provide a depth of insight on the use of a package at their organizations.

**Request for proposal (RFP):** A document provided to vendors that asks them to propose hardware and system software that will meet the requirements of a new system.
To gain a range of opinions about possible packages, you can use independent software testing and abstracting services that periodically evaluate software and collect user opinions. Such surveys are available for a fee either as subscription services or on demand (two popular services are Auerbach Publishers and DataPro); occasionally, unbiased surveys appear in trade publications. Often, however, articles in trade publications, even software reviews, are actually seeded by the software manufacturer and are not unbiased.

If you are comparing several software packages, you can assign scores for each package on each criterion and compare the scores using the quantitative method we demonstrate in Chapter 4 for comparing alternative system design strategies.

REUSE

Reuse is the use of previously written software resources in new applications. Because so many bits and pieces of applications are relatively generic across applications, it seems intuitive that great savings can be achieved in many areas if those generic bits and pieces do not have to be written anew each time they are needed. Reuse should increase programmer productivity, because being able to use existing software for some functions means they can perform more work in the same amount of time. Reuse should also decrease development time, minimizing schedule overruns. Because existing pieces of software have already been tested, reusing them should also result in higher-quality software with lower defect rates, decreasing maintenance costs.

Although reuse can conceivably apply to many different aspects of software, typically it is most commonly applied to two different development technologies: object-oriented and component-based development. You were briefly introduced to object-oriented development in Chapter 1. For example, consider an object class created to model an employee. The object class Employee would contain both the data about employees and the instructions necessary for calculating payroll for a variety of job types. The object class could be used in any application that dealt with employees, but if changes had to be made in calculating payroll for different types of employees, the changes would only have to be made to the object class and not to the various applications that used it. By definition, using the Employee object class in more than one application constitutes reuse.

Component-based development is similar to object-oriented development in that the focus is on creating general-purpose pieces of software that can be used interchangeably in many different programs. Components can be as small as objects or as large as pieces of software that handle single business functions, such as currency conversion. The idea behind component-based development is the assembly of an application from many different components at many different levels of complexity and size. Many vendors are working on developing libraries of components that can be retrieved and assembled as needed into desired applications.

Some evidence suggests that reuse can be effective, especially for object classes. For example, one laboratory study found that reuse of object class libraries resulted in increased productivity, reduced defect density, and reduced rework (Basili et al., 1996). For HP, a reuse program resulted in cutting time to market for certain products by a factor of three or more, from 18 months to less than 5 months (Griss, 2003). However, for reuse to work in an organizational setting, many different issues must be addressed. Technical issues include the current lack of a methodology for creating and clearly defining and labeling reusable components for placement in a library and the small number of reusable and reliable software resources currently available. Key organizational issues include the lack of commitment to reuse, as well as the lack of proper training and rewards needed to promote it, the lack of organizational support for institutionalizing reuse, and the difficulty in measuring the economic gains from reuse. Royce (1998) argues that, due to the considerable costs of developing a
Reusable component, most organizations cannot compete economically with established commercial organizations that focus on selling components as their main line of business. Success depends on being able to leverage the cost of components across a large user and project base (Figure 2-4). There are also key legal and contractual issues concerning the reuse of object classes and components originally used in other applications (Kim and Stohr, 1998).

When an organization’s management decides to pursue reuse as a strategy, it is important for the organization to match its approach to reuse with its strategic business goals (Griss, 2003). The benefits of reuse grow as more corporate experience is gained from it, but so do the costs and the amount of resources necessary for reuse to work well. Software reuse has three basic steps: abstraction, storage, and recontextualization (Grinter, 2001). Abstraction involves the design of a reusable piece of software, starting from existing software assets or from scratch. Storage involves making software assets available for others to use. Although it sounds like a simple problem, storage can actually be very challenging. The problem is not simply putting software assets on a shelf; the problem is correctly labeling and cataloging assets so that others can find the ones they want to use. Once an asset has been found, recontextualization becomes important. This involves making the reusable asset understandable to developers who want to use it in their systems. Software is complex, and a software asset developed for a particular system under system-specific circumstances may not at all be the asset it appears to be. What appears to be a generic asset called “Customer” may actually be something quite different, depending on the context in which it was developed. It may often appear to be easier to simply build your own assets rather than invest the time and energy it takes to establish a good understanding of software someone else has developed. A key part of a reuse strategy, as mentioned previously, is establishing rewards, incentives, and organizational support for reuse to help make it more worthwhile than developing your own assets.

According to Griss (2003), an organization can take one of four approaches to reuse (Table 2-3). The ad hoc approach to reuse is not really an approach at all, at least from an official organizational perspective. With this approach, individuals are free to find or develop reusable assets on their own, and there are few, if any, organizational rewards for reusing assets. Storage is not an issue, because individuals keep track of and distribute their own software assets. For such an ad hoc, individually driven approach, it is difficult to measure any potential benefits to the company.
Another approach to reuse is facilitated reuse. With this approach, developers are not required to practice reuse, but they are encouraged to do so. The organization makes available some tools and techniques that enable the development and sharing of reusable assets, and one or more employees may be assigned the role of evangelist to publicize and promote the program. Very little is done to track the quality and use of reusable assets, however, and the overall corporate investment is small.

Managed reuse is a more structured, and more expensive, mode of managing software reuse. With managed reuse, the development, sharing, and adoption of reusable assets is mandated. The organization establishes processes and policies for ensuring that reuse is practiced and that the results are measured. The organization also establishes policies and procedures for ensuring the quality of its reusable assets. The focus is on identifying existing assets that can be potentially reused from various sources, including from utility asset libraries that come with operating systems, from companies that sell assets, from the open source community, from internal repositories, from scouring existing legacy code, and so on.

The most expensive and extensive approach to reuse is designed reuse. In addition to mandating reuse and measuring its effectiveness, the designed reuse approach takes the extra step of mandating that assets be designed for reuse as they are being designed for specific applications. The focus is more on developing reusable assets than on finding existing assets that might be candidates for reuse. A corporate reuse office may be established to monitor and manage the overall methodology. Under such an approach, as much as 90 percent of software assets may be reused across different applications.

Each approach to reuse has its advantages and disadvantages. No single approach is a silver bullet that will solve the reuse puzzle for all organizations and for...
all situations. Successful reuse requires an understanding of how reuse fits within larger organizational goals and strategies as well as an understanding of the social and technical world into which the reusable assets must fit.

**Summary**

As a systems analyst, you must be aware of where you can obtain software that meets some or all of an organization’s needs. You can obtain application (and system) software from information technology services firms, packaged software providers, vendors of enterprise-wide solution software, application (and managed) service providers, and open source software providers, as well as from internal systems development resources, including the reuse of existing software components. You can even hire an organization to handle all of your systems development needs, which is called outsourcing. You must also know the criteria to use to when choosing among off-the-shelf software products. These criteria include cost, functionality, vendor support, vendor viability, flexibility, documentation, response time, and ease of installation. Requests for proposals are one way you can collect more information about system software, its performance, and its costs.

**Key Terms**

1. Application service provider (ASP)  
2. Enterprise resource planning (ERP) systems  
3. Managed service provider (MSP)  
4. Outsourcing  
5. Request for proposal (RFP)  
6. Reuse

Match each of the key terms above with the definition that best fits it.

- **Application service provider (ASP)** Organizations that host and run computer applications for other companies, typically on a per-use or license basis.
- **Enterprise resource planning (ERP) systems** The practice of turning over responsibility of some to all of an organization’s information systems applications and operations to an outside firm.
- **Managed service provider (MSP)** The use of previously written software resources, especially objects and components, in new applications.
- **Outsourcing** An organization that remotely provides customized computer applications and network-based services for other companies for a monthly or per-use fee.
- **Request for proposal (RFP)** A document that is provided to vendors to ask them to propose hardware and system software that will meet the requirements of your new system.
- **Reuse** A system that integrates individual traditional business functions into a series of modules so that a single transaction occurs seamlessly within a single information system rather than several separate systems.

**Review Questions**

1. Describe and compare the various sources of software.  
2. What are the differences between ASPs and MSPs?  
3. How can you decide among various off-the-shelf software options? What criteria should you use?  
4. What is an RFP and how do analysts use one to gather information on hardware and system software?  
5. What methods can a systems analyst employ to verify vendor claims about a software package?  
6. What are ERP systems? What are the benefits and disadvantages of such systems as a design strategy?  
7. Explain reuse, its advantages and disadvantages.  
8. Compare and contrast the four approaches to reuse.

**Problems and Exercises**

1. Research how to prepare an RFP.  
2. Review the criteria for selecting off-the-shelf software presented in this chapter. Use your experience and imagination and describe other criteria that are or might be used to select off-the-shelf software in the “real world.” For each new criterion, explain how use of this criterion might be functional (i.e., it is useful to use this criterion), dysfunctional, or both.
3. In the section on choosing off-the-shelf software, eight criteria are proposed for evaluating alternative packages. Suppose the choice was between alternative custom software developers rather than prewritten packages. What criteria would be appropriate to select and compare among competing bidders for custom development of an application? Define each of these criteria.

4. How might the project team recommending an ERP design strategy justify its recommendation as compared with other types of design strategies?

**Field Exercises**

1. Interview businesspeople who participate in the purchase of off-the-shelf software in their organizations. Review with them the criteria for selecting off-the-shelf software presented in this chapter. Have them prioritize the list of criteria as they are used in their organization and provide an explanation of the rationale for the ranking of each criterion. Ask them to list and describe any other criteria that are used in their organization.

2. Obtain copies of actual RFPs used for information systems developments and/or purchases. If possible, obtain RFPs from public and private organizations. Find out how they are used. What are the major components of these proposals? Do these proposals seem to be useful? Why or why not? How and why do RFPs from public and private organizations differ?

3. Contact an organization that has or is implementing an integrated ERP application. Why did it choose this design strategy? How has it managed this development project differently from prior large projects? What organizational changes have occurred due to this design strategy? How long did the implementation last and why?

**References**


