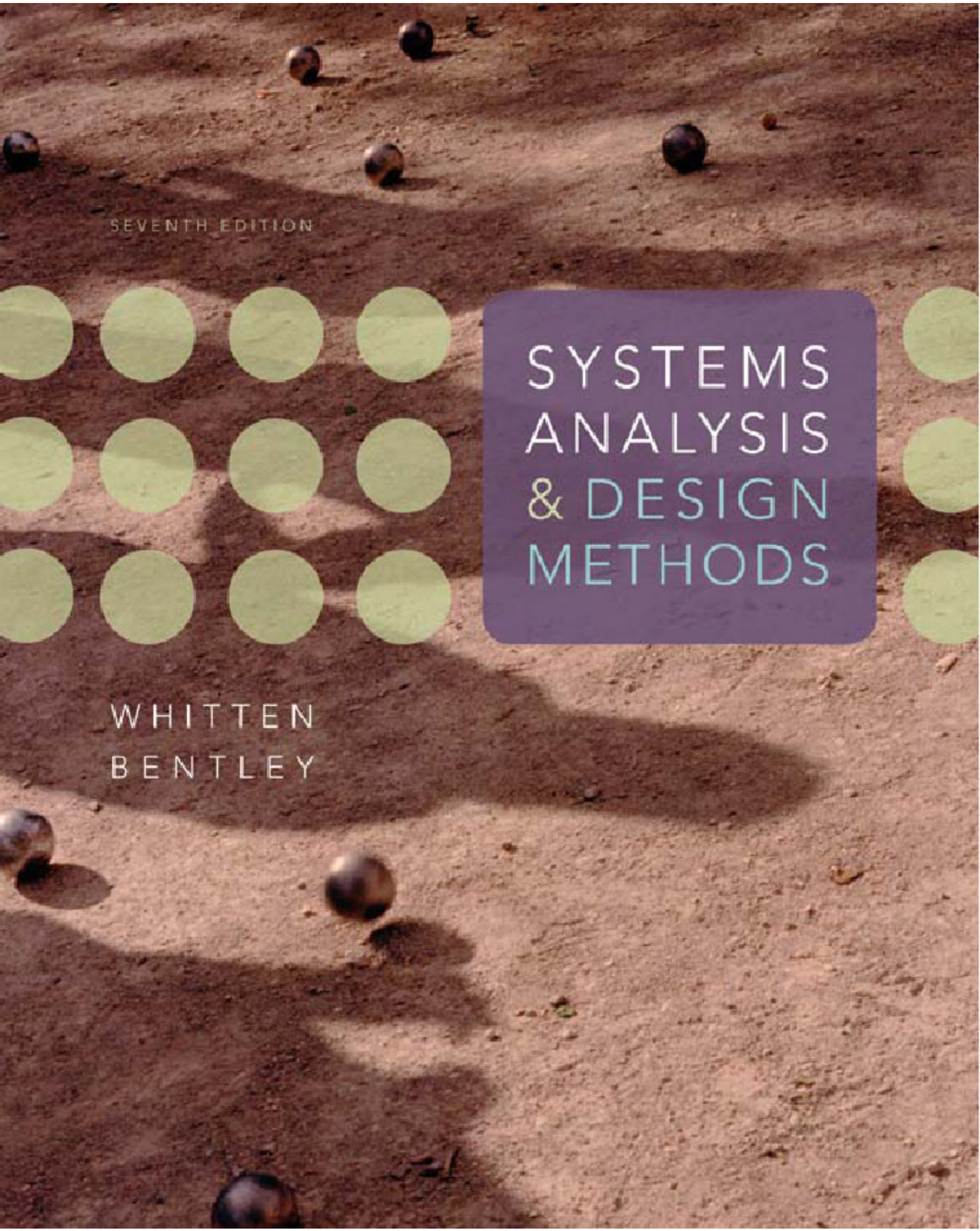

SEVENTH EDITION
SYSTEMS ANALYSIS & DESIGN METHODS
WHITTEN
BENTLEY

3 Information Systems Development

Chapter Preview and Objectives

This chapter more closely examines the systems development process that was first introduced in Chapter 1. Successful systems development is governed by some fundamental, underlying principles that we introduce in this chapter. We also introduce a basic, representative systems development methodology as a disciplined approach to developing information systems. Although such an approach will not guarantee success, it will greatly improve the chances of success. You will know that you understand information systems development when you can:

- Describe the motivation for a standard systems development process in terms of the Capability Maturity Model (CMM) for quality management.
- Differentiate between the system life cycle and a system development methodology.
- Describe 10 basic principles of systems development.
- Define problems, opportunities, and directives—the triggers for systems development projects.
- Describe the PIECES framework for categorizing problems, opportunities, and directives.
- Describe the essential phases of systems development. For each phase, describe its purpose, inputs, and outputs.
- Describe cross life-cycle activities that overlap multiple system development phases.
- Describe typical, alternative "routes" through the essential phases of systems development. Describe how routes may be combined or customized for different types of projects.
- Describe various automated tools for systems development.

Introduction

Work is getting underway at SoundStage Entertainment Club for the systems analysis and design of their member services information system. But the more Bob Martinez learns about the system, the more confused he gets. Bob can recall some of his programming assignments in college. Most of them were just a page or two of bulleted points listing required features. It was pretty easy to get your head around that. But the new SoundStage system will involve tracking member contacts and purchase requirements, promotions, sales, shipments, inventory, multiple warehouses, Web sites, and more. Bob wonders how they will even list all the requirements, let alone keep them straight. How will they know what data they need to track? How will they know what every piece of programming needs to do? He mentioned that to his boss, Sandra. She said it was all about following "the methodology." He remembered something about methodology from a systems analysis class. At the time it seemed like a lot of unnecessary work. But he is starting to see now that on a large project, following an established method may be the only path that is safe to travel.

The Process of Systems Development

systems development process a set of activities, methods, best practices, deliverables, and automated tools that stakeholders (from Chapter 1) use to develop and continuously improve information systems and software (from Chapters 1 and 2).

This chapter introduces a focus on information systems development. We will examine a **systems development process.** Notice we did not say "the" process—there are as many variations on the process as there are experts and authors. We will present one representative process and use it consistently throughout this book. The chapter home page shows an expanded number of phases compared to the home page of Chapter 1. This is because we have factored the high-level phases such as system analysis and system design from Chapter 1 into multiple phases and activities. We have also refined the size and place of the stakeholder roles to reflect "involvement" as opposed to emphasis or priority. And we have edited and enhanced the building blocks to indicate deliverables and artifacts of system development. All of these modifications will be explained in due time.

Why do organizations embrace standardized processes to develop information systems? Information systems are a complex product. Recall from Chapter 2 that an information system includes data, process, and communications building blocks and technologies that must serve the needs of a variety of stakeholders. Perhaps this explains why as many as 70 percent or more of information system development projects have failed to meet expectations, cost more than budgeted, and are delivered much later than promised. The Gartner Group suggests that "consistent adherence to moderately rigorous methodology guidelines can provide 70 percent of [systems development] organizations with a productivity improvement of at least 30 percent within two years."[1]

Increasingly, organizations have no choice but to adopt and follow a standardized systems development process. First, using a consistent process for systems development creates efficiencies that allow management to shift resources between projects. Second, a consistent methodology produces consistent documentation that reduces lifetime costs to maintain the systems. Finally, the U.S. government has mandated that any organization seeking to develop software for the government must adhere to certain quality management requirements. A consistent process promotes quality. And many other organizations have aggressively committed to total quality management goals in order to increase their competitive advantage. In order to realize quality and productivity improvements, many organizations have turned to project and process management frameworks such as the Capability Maturity Model, discussed in the next section.

[1]Richard Hunter, "AD Project Portfolio Management," *Proceedings of the Gartner Group IT98 Symposium/Expo* (CD-ROM). The Gartner Group is an industry watchdog and research group that tracks and projects industry trends for IT managers.

> The Capability Maturity Model

It has been shown that as an organization's information system development process matures, project timelines and cost decrease while productivity and quality increase. The Software Engineering Institute at Carnegie Mellon University has observed and measured this phenomenon and developed the **Capability Maturity Model (CMM)** to assist all organizations to achieve these benefits. The CMM has developed a wide following, both in industry and government. Software evaluation based on CMM is being used to qualify information technology contractors for most U.S. federal government projects.

Capability Maturity Model (CMM) a standardized framework for assessing the maturity level of an organization's information systems development and management processes and products. It consists of five levels of maturity.

The CMM framework for systems and software is intended to help organizations improve the maturity of their systems development processes. The CMM is organized into five maturity levels (see Figure 3-1):

- *Level 1—Initial:* This is sometimes called anarchy or chaos. At this level, system development projects follow no consistent process. Each development team uses its own tools and methods. Success or failure is usually a function of the skill and experience of the team. The process is unpredictable and not repeatable. A project typically encounters many crises and is frequently over budget and behind schedule. Documentation is sporadic or not consistent from one project to the next, thus creating problems for those who must maintain a system over its lifetime. Almost all organizations start at Level 1.
- *Level 2—Repeatable:* At this level, project management processes and practices are established to track project costs, schedules, and functionality. The focus is on project management. A system development process is always followed, but it may vary from project to project. Success or failure is still a function of the skill and experience of the project team; however, a concerted effort is made to repeat earlier project successes. Effective project management practices lay the foundation for standardized processes in the next level.

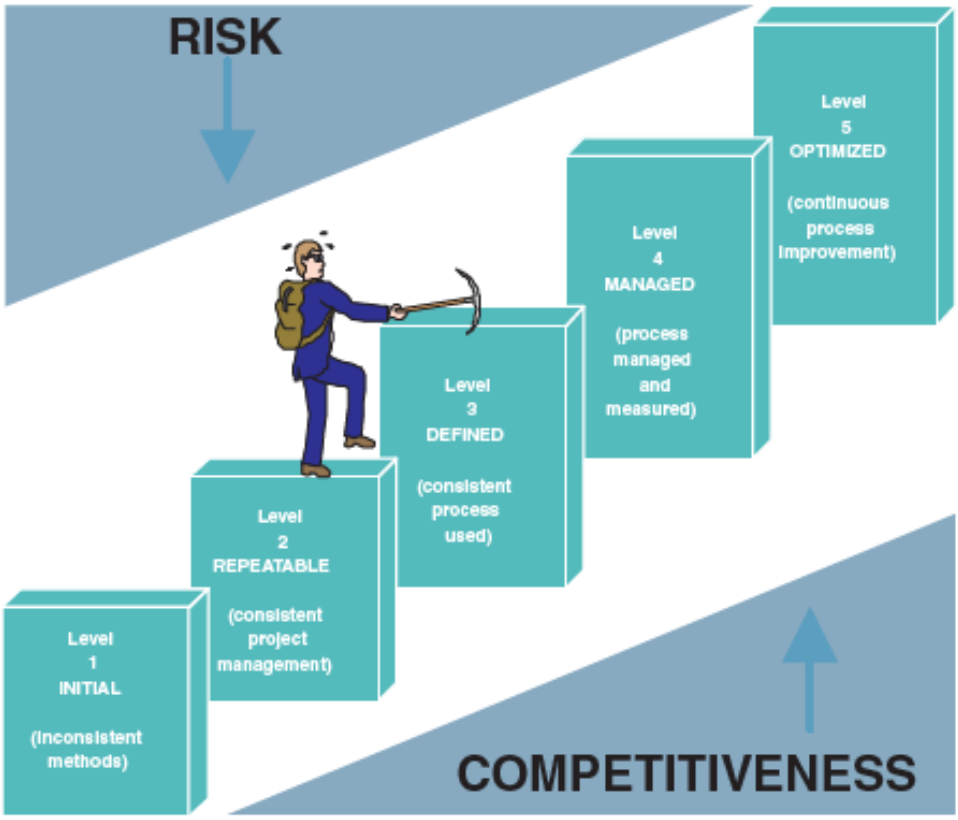


FIGURE 3-1
The Capability Maturity Model (CMM)

TABLE 3-1 Impact of System Development "Process" on Quality

CMM Project Statistics for a Project Resulting in 200,000 Lines of Code

Organization's CMM Level	Project Duration (months)	Project Person-Months	Number of Defects Shipped	Median Cost ($ millions)	Lowest Cost ($ millions)	Highest Cost ($ millions)
1	30	600	61	5.5	1.8	100+
2	18.5	143	12	1.3	.96	1.7
3	15	80	7	.728	.518	.933

Source: Master Systems, Inc.

system development methodology a formalized approach to the systems development process; a standardized process that includes the activities, methods, best practices, deliverables, and automated tools to be used for information systems development.

- *Level 3—Defined:* In this level, a standard **system development** process (sometimes called a *methodology*) is purchased or developed. All projects use a tailored version of this process to develop and maintain information systems and software. As a result of using the standardized process for all projects, each project results in consistent and high-quality documentation and deliverables. The process is stable, predictable, and repeatable.
- *Level 4—Managed:* In this level, measurable goals for quality and productivity are established. Detailed measures of the standard system development process and product quality are routinely collected and stored in a database. There is an effort to improve individual project management based on this collected data. Thus, management seeks to become more proactive than reactive to systems development problems (such as cost overruns, scope creep, schedule delays, etc.). Even when a project encounters unexpected problems or issues, the process can be adjusted based on predictable and measurable impacts.
- *Level 5—Optimizing:* In this level, the standardized system development process is continuously monitored and improved based on measures and data analysis established in Level 4. This can include changing the technology and best practices used to perform activities required in the standard system development process, as well as adjusting the process itself. Lessons learned are shared across the organization, with a special emphasis on eliminating inefficiencies in the system development process while sustaining quality. In summary, the organization has institutionalized continuous systems development process improvement.

It is very important to recognize that each level is a prerequisite for the next level.

Currently, many organizations are working hard to achieve at least CMM Level 3 (sometimes driven by a government or organizational mandate). A central theme to achieving Level 3 (Defined) is the use of a standard process or methodology to build or integrate systems. As shown in Table 3-1, an organization can realize significant improvements in schedule and cost by institutionalizing CMM Level 3 process improvements.[2]

system life cycle the factoring of the lifetime of an information system into two stages, (1) systems development and (2) systems operation and maintenance—first you build it; then you use and maintain it. Eventually, you cycle back to redevelopment of a new system.

> Life Cycle versus Methodology

The terms *system life cycle* and *system development methodology* are frequently and incorrectly interchanged. Most system development processes are derived from a natural **system life cycle.** The system life cycle just happens. Figure 3-2 illustrates two

[2]White Paper, "Rapidly Improving Process Maturity: Moving Up the Capability Maturity Model through Outsourcing" (Boston: Keane, 1997, 1998, p.11).

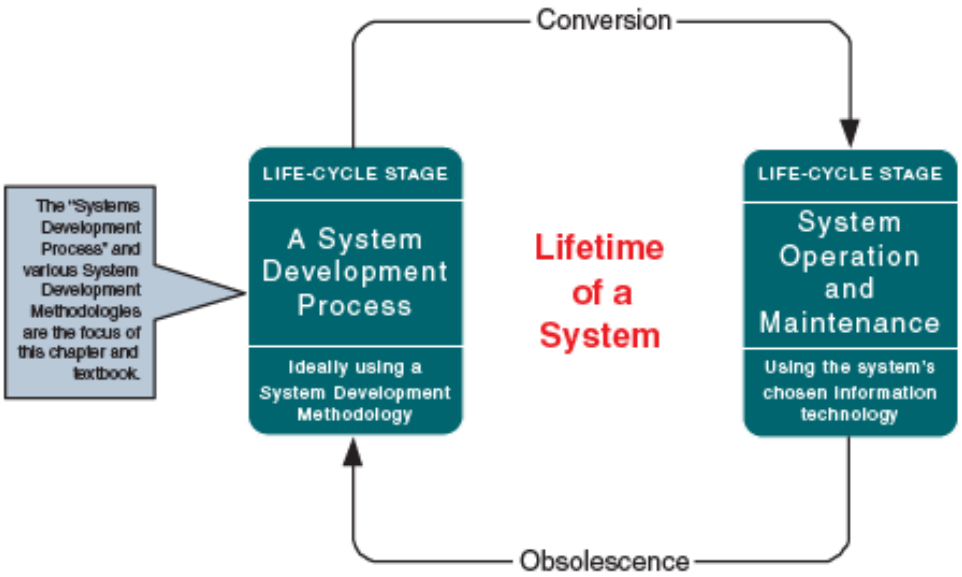


FIGURE 3-2
The System Life Cycle

life-cycle stages. Notice that there are two key events that trigger a change from one stage to the other:

- When a system cycles from development to operation and maintenance, a *conversion* must take place.
- At some point in time, *obsolescence* occurs (or is imminent) and a system cycles from operation and maintenance to redevelopment.

Actually, a system may be in more than one stage at the same time. For example, one version may be in *operation and support* while the next version is in *development.*

So how does this contrast with a systems development methodology? A systems development methodology "executes" the systems development stage of the system life cycle. Each individual information system has its own system life cycle. The methodology is the standard process to build and maintain that system and all other information systems through their life cycles. Consistent with the goals of the CMM, methodologies ensure that:

- A consistent, reproducible approach is applied to all projects.
- There is reduced risk associated with shortcuts and mistakes.
- Complete and consistent documentation is produced from one project to the next.
- Systems analysts, designers, and builders can be quickly reassigned between projects because all use the same process.
- As development teams and staff constantly change, the results of prior work can be easily retrieved and understood by those who follow.

Methodologies can be purchased or homegrown. Why purchase a methodology? Many information system organizations can't afford to dedicate staff to the development and continuous improvement of a homegrown methodology. Methodology vendors have a vested interest in keeping their methodologies current with the latest business and technology trends. Homegrown methodologies are usually based on generic methodologies and techniques that are well documented in books and on Web sites. Examples of system development methodologies are listed in the margin on the following page. You should be able to research most of them on the Web. Many of their underlying methods will be taught in this textbook.

Throughout this book, we will use a methodology called ***FAST,*** which stands for *F*ramework for the *A*pplication of *S*ystems *T*hinking. *FAST* is not a real-world commercial methodology. We developed it as a composite of the best practices we've

FAST a hypothetical methodology used throughout this book to demonstrate a representative systems development process. The acronym's letters stand for *F*ramework for the *A*pplication of *S*ystems *T*hinking.

REPRESENTATIVE SYSTEM DEVELOPMENT METHODOLOGIES

Architected Rapid Application Development (Architected RAD)
Dynamic Systems Development Methodology (DSDM)
Joint Application Development (JAD)
Information Engineering (IE)
Rapid Application Development (RAD)
Rational Unified Process (RUP)
Structured Analysis and Design (old, but still occasionally encountered)
eXtreme Programming (XP)
Note: There are many commercial methodologies and software tools (sometimes called *methodware*) based on the above general methodologies.

encountered in many commercial and reference methodologies. Unlike many commercial methodologies, *FAST* is not prescriptive. That is to say, *FAST* is an agile framework that is flexible enough to provide for different types of projects and strategies. Most important, *FAST* shares much in common with both the book-based and the commercial methodologies that you will encounter in practice.

> Underlying Principles for Systems Development

Before we examine the *FAST* methodology, let's introduce some general principles that should underlie all systems development methodologies.

Principle 1: Get the System Users Involved Analysts, programmers, and other information technology specialists frequently refer to "my system." This attitude has, in part, created an "us versus them" conflict between technical staff and their users and management. Although analysts and programmers work hard to create technologically impressive solutions, those solutions often backfire because they don't address the real organization problems. Sometimes they even introduce new organization problems. For this reason, system user involvement is an absolute necessity for successful systems development. Think of systems development as a partnership between system users, analysts, designers, and builders. The analysts, designers, and builders are responsible for systems development, but they must engage their owners and users, insist on their participation, and seek agreement from all stakeholders concerning decisions that may affect them.

Miscommunication and misunderstandings continue to be a significant problem in many systems development projects. However, owner and user involvement and education minimize such problems and help to win acceptance of new ideas and technological change. Because people tend to resist change, information technology is often viewed as a threat. The best way to counter that threat is through constant and thorough communication with owners and users.

Principle 2: Use a Problem-Solving Approach A system development methodology is, first and foremost, a problem-solving approach to building systems. The term *problem* is broadly used throughout this book to include (1) real problems, (2) opportunities for improvement, and (3) directives from management. The classical problem-solving approach is as follows:

1. Study and understand the problem, its context, and its impact.
2. Define the requirements that must be met by *any* solution.
3. Identify candidate solutions that fulfill the requirements, and select the best solution.
4. Design and/or implement the chosen solution.
5. Observe and evaluate the solution's impact, and refine the solution accordingly.

Systems analysts should approach all projects using some variation of this problem-solving approach.

Inexperienced or unsuccessful problem solvers tend to eliminate or abbreviate one or more of the above steps. For example, they fail to completely understand the problem, or they prematurely commit to the first solution they think of. The result can range from (1) solving the wrong problem, to (2) incorrectly solving the problem, (3) picking the wrong solution, or (4) picking a less-than-optimal solution. A methodology's problem-solving process, when correctly applied, can reduce or eliminate these risks.

Principle 3: Establish Phases and Activities All methodologies prescribe phases and activities. The number and scope of phases and activities vary from author to author, expert to expert, methodology to methodology, and business to business. The chapter home page at the beginning of this chapter illustrates the eight phases of our *FAST* methodology in the context of your information systems framework. The phases

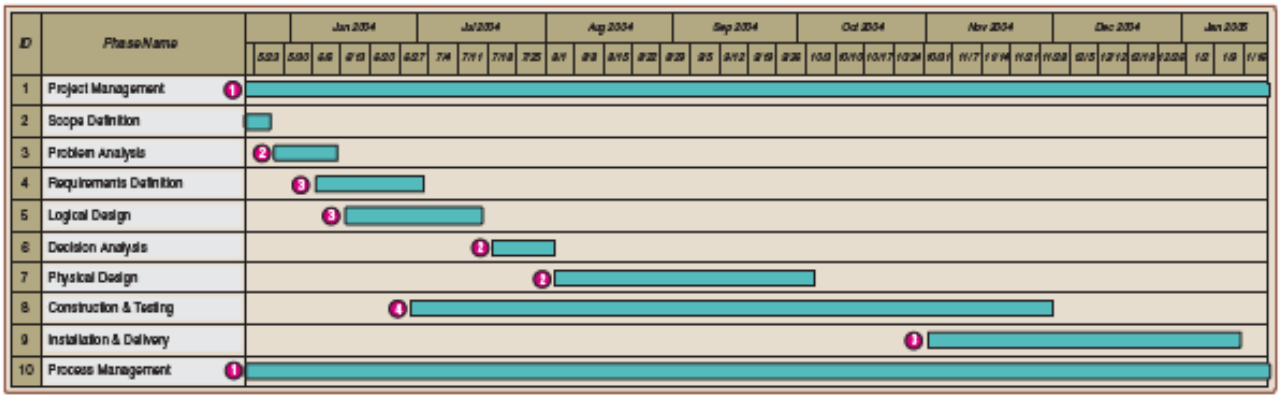


FIGURE 3-3 Overlap of System Development Phases and Activities

are listed on the far right-hand side of the illustration. In each phase, the focus is on those building blocks and on stakeholders that are aligned to the left of that phase.

The phases are: scope definition, problem analysis, requirements analysis, logical design, decision analysis, physical design and integration, construction and testing, and installation and delivery. Each of these phases will be discussed later in this chapter. These phases are not absolutely sequential. The phases tend to overlap one another, as illustrated in Figure 3-3. Also, the phases may be customized to the special needs of a given project (e.g., deadlines, complexity, strategy, resources). In this chapter, we will describe each customization as alternative routes through the methodology and problem-solving process.

Principle 4: Document throughout Development When do you document the programs you write? Be honest. We must confess that, like most students, we did our documentation after we wrote the programs. We knew better, but we postdocumented anyway. That just does not work in the business world. In medium to large organizations, system owners, users, analysts, designers, and builders come and go. Some will be promoted; some will have extended medical leaves; some will quit the organization; and still others will be reassigned. To promote good communication between constantly changing stakeholders, documentation should be a working by-product of the entire systems development effort.

Documentation enhances communications and acceptance. Documentation reveals strengths and weaknesses of the system to multiple stakeholders. It stimulates user involvement and reassures management about progress. At the same time, some methodologies have been criticized for expecting too much documentation that adds little value to the process or resulting system. Our *FAST* methodology advocates a balance between the value of documentation and the effort to produce it. Experts call this *agile modeling*.

Principle 5: Establish Standards In a perfect world, all information systems would be integrated such that they behave as a single system. Unfortunately, this never happens because information systems are developed and replaced over a very long period of time. Even organizations that purchase and install an enterprise resource planning (ERP) product usually discover that there are applications and needs that fall outside the ERP system. Systems integration has become critical to the success of any organization's information systems.

To achieve or improve systems integration, organizations turn to standards. In many organizations, these standards take the form of enterprise information technology architecture. An IT architecture sets standards that serve to direct technology solutions and information systems to a common technology vision or configuration.

An information technology architecture typically standardizes on the following (note: it is not important that you know what all these sample technologies are):

- *Database technology*—What database engine(s) will be used (e.g., *Oracle,* IBM *DB2,* Microsoft *SQL Server*)? On what platforms will they be operated (e.g., *UNIX, Linux, Windows XP, MVS*)? What technologies will be used to load data into online transaction processing (OLTP) databases, operational data stores, and data warehouses (i.e., Extract Transform and Load [ETL])?
- *Software technology*—What application development environment(s)/language(s) will be used to write software (e.g., IBM's *Websphere* with *Java,* Microsoft's *Visual Studio .NET* with *Visual Basic .NET, Visual C++*, and/or *Visual C#;* Syebase's *Powerbuilder,* Oracle's *Oracle Forms*)?
- *Interface technology*—How will user interfaces be developed—with *MS Windows* components or Web languages and components (e.g., an *xHTML* editor such as Macromedia's *Dreamweaver,* a portal engine such as IBM's *Websphere*)? How will data be exchanged between different information systems (e.g., a data broker such as IBM's *MQ Messaging,* an XML-based data exchange, or a custom programmed interface)?

Notice how these architectural questions closely correspond to the technology drivers in your information system model.

In the absence of an IT architecture, each information system and application may be built using radically different technologies. Not only does this make it difficult to integrate applications, but it creates resource management problems—IT organizations cannot as easily move developers between projects as priorities change or emergencies occur because different teams are staffed with technical competencies based on the various technologies used and being used to develop information systems. Creating an enterprise IT architecture and driving projects and teams to that architecture make more sense.

As new technologies emerge, an IT architecture must change. But that change can be managed. The chief technology officer (CTO) in an organization is frequently charged with technology exploration and IT architecture management. Given that architecture, all information systems projects are constrained to implement new systems that conform to the architecture (unless otherwise approved by the CTO).

Principle 6: Manage the Process and Projects Most organizations have a system development process or methodology, but they do not always use it consistently on projects. Both the process and the projects that use it must be managed. **Process management** ensures that an organization's chosen process or management is used consistently on and across all projects. Process management also defines and improves the chosen process or methodology over time. **Project management** ensures that an information system is developed at minimum cost, within a specified time frame, and with acceptable quality (using the standard system development process or methodology). Effective project management is essential to achieving CMM Level 2 success. Use of a repeatable process gets us to CMM Level 3. CMM Levels 4 and 5 require effective process management. Project management can occur without a standard process, but in mature organizations all projects are based on a standardized and managed process.

process management an ongoing activity that documents, teaches, oversees the use of, and improves an organization's chosen methodology (the "process") for systems development. Process management is concerned with phases, activities, deliverables, and quality standards that should be consistently applied to all projects.

project management the process of scoping, planning, staffing, organizing, directing, and controlling a project to develop an information system at minimum cost, within a specified time frame, and with acceptable quality.

Process management and project management are influenced by the need for quality management. Quality standards are built into a process to ensure that the activities and deliverables of each phase will contribute to the development of a high-quality information system. They reduce the likelihood of missed problems and requirements, as well as flawed designs and program errors (bugs). Standards also make the IT organization more agile. As personnel changes occur, staff can be relocated between projects with the assurance that every project is following an understood and accepted process.

Principle 7: Justify Information Systems as Capital Investments Information systems are capital investments, just like a fleet of trucks or a new building. System owners commit to this investment. Notice that the initial commitment occurs early in a project, when system owners agree to sponsor and fund the project. Later (during the phase called *decision analysis*), system owners recommit to the more costly technical decisions. In considering a capital investment, two issues must be addressed:

1. For any problem, there are likely to be several possible solutions. The systems analyst and other stakeholders should not blindly accept the first solution suggested. The analyst who fails to look at alternatives may be doing the business a disservice.
2. After identifying alternative solutions, the systems analyst should evaluate each possible solution for feasibility, especially for **cost-effectiveness.** Cost-effectiveness is measured using a technique called *cost-benefit analysis.*

Like project and process management, cost-benefit analysis is performed throughout the system development process.

A significant advantage of the phased approach to systems development is that it provides several opportunities to reevaluate cost-effectiveness, risk, and feasibility. There is often a temptation to continue with a project only because of the investment already made. In the long run, canceled projects are usually much less costly than implemented disasters. This is extremely important for young analysts to remember.

Most system owners want more from their systems than they can afford or are willing to pay for. Furthermore, the scope of most information system projects increases as the analyst learns more about the business problems and requirements as the project progresses. Unfortunately, most analysts fail to adjust estimated costs and schedules as the scope increases. As a result, the analyst frequently and needlessly accepts responsibility for cost and schedule overruns.

Because information systems are recognized as capital investments, system development projects are often driven by enterprise planning. Many contemporary information technology business units create and maintain a **strategic information systems plan.** Such a plan identifies and prioritizes information system development projects. Ideally, a strategic information systems plan is driven by a **strategic enterprise plan** that charts a course for the entire business.

cost-effectiveness the result obtained by striking a balance between the lifetime costs of developing, maintaining, and operating an information system and the benefits derived from that system. Cost-effectiveness is measured by cost-benefit analysis.

strategic information systems plan a formal strategic plan (3 to 5 years) for building and improving an information technology infrastructure and the information system applications that use that infrastructure.

strategic enterprise plan a formal strategic plan (3 to 5 years) for an entire business that defines its mission, vision, goals, strategies, benchmarks, and measures of progress and achievement. Usually, the strategic enterprise plan is complemented by strategic business unit plans that define how each business unit will contribute to the enterprise plan. The information systems plan (above) is one of those unit-level plans.

Principle 8: Don't Be Afraid to Cancel or Revise Scope There is an old saying: "Don't throw good money after bad." In other words, don't be afraid to cancel a project or revise scope, regardless of how much money has been spent so far—cut your losses. To this end, we advocate a **creeping commitment** approach to systems development.[3] With the creeping commitment approach, multiple feasibility checkpoints are built into any systems development methodology. At each checkpoint feasibility is reassessed. All previously expended costs are considered sunk (meaning not recoverable). They are, therefore, irrelevant to the decision. Thus, the project should be reevaluated at each checkpoint to determine if it remains feasible to continue investing time, effort, and resources into the project. At each checkpoint, the analyst should consider the following options:

- Cancel the project if it is no longer feasible.
- Reevaluate and adjust the costs and schedule if project scope is to be increased.
- Reduce the scope if the project budget and schedule are frozen and not sufficient to cover all project objectives.

The concept of sunk costs is more or less familiar to most financial analysts, but it is frequently forgotten or not used by the majority of systems analysts, most system users, and even many system owners.

creeping commitment a strategy in which feasibility and risks are continuously reevaluated throughout a project. Project budgets and deadlines are adjusted accordingly.

[3]Thomas Gildersleeve, *Successful Data Processing Systems Analysis,* 2nd ed. (Englewood Cliffs, NJ: Prentice Hall, 1985), pp. 5–7.

risk management the process of identifying, evaluating, and controlling what might go wrong in a project before it becomes a threat to the successful completion of the project or implementation of the information system. Risk management is driven by risk analysis or assessment.

In addition to managing feasibility throughout the project, we must manage risk. **Risk management** seeks to balance risk and reward. Different organizations are more or less averse to risk, meaning that some are willing to take greater risks than others in order to achieve greater rewards.

Principle 9: Divide and Conquer Whether you realize it or not, you learned the divide-and-conquer approach throughout your education. Since high school, you've been taught to outline a paper before you write it. Outlining is a divide-and-conquer approach to writing. A similar approach is used in systems development. We divide a system into subsystems and components in order to more easily conquer the problem and build the larger system. In systems analysis, we often call this *factoring*. By repeatedly dividing a larger problem (system) into more easily managed pieces (subsystems), the analyst can simplify the problem-solving process. This divide-and-conquer approach also complements communication and project management by allowing different pieces of the system to be communicated to different and the most appropriate stakeholders.

The building blocks of your information system framework provide one basis for dividing and conquering an information system's development. We will use this framework throughout the book.

Principle 10: Design Systems for Growth and Change Businesses change over time. Their needs change. Their priorities change. Accordingly, information systems that support a business must change over time. For this reason, good methodologies should embrace the reality of change. Systems should be designed to accommodate both growth and changing requirements. In other words, well-designed information systems can both scale up and adapt to the business. But regardless of how well we design systems for growth and change, there will always come a time when they simply cannot support the business.

System scientists describe the natural and inevitable decay of all systems over time as *entropy*. As described earlier in this section, after a system is implemented it enters the *operations and maintenance* stage of the life cycle. During this stage the analyst encounters the need for changes that range from correcting simple mistakes, to redesigning the system to accommodate changing technology, to making modifications to support changing user requirements. Such changes direct the analyst and programmers to rework formerly completed phases of the life cycle. Eventually, the cost of maintaining the current system exceeds the costs of developing a replacement system—the current system has reached entropy and becomes obsolete.

But system entropy can be managed. Today's tools and techniques make it possible to design systems that can grow and change as requirements grow and change. This book will teach you many of those tools and techniques. For now, it's more important to simply recognize that flexibility and adaptability do not happen by accident—they must be built into a system.

We have presented 10 principles that should underlie any methodology. These principles are summarized in the margin and can be used to evaluate any methodology, including our *FAST* methodology.

PRINCIPLES OF SYSTEMS DEVELOPMENT

Get the System Users Involved.
Use a Problem-Solving Approach.
Establish Phases and Activities.
Document throughout Development.
Establish Standards.
Manage the Process and Projects.
Justify Information Systems as Capital Investments.
Don't Be Afraid to Cancel or Revise Scope.
Divide and Conquer.
Design Systems for Growth and Change.

A Systems Development Process

In this section we'll examine a logical process for systems development. (Reminder: *FAST* is a hypothetical methodology created for learning purposes.) We'll begin by studying types of system projects and how they get started. Then we'll introduce the eight *FAST* phases. Finally, we'll examine alternative variations, or "routes" through the phases, for different types of projects and development strategies.

> Where Do Systems Development Projects Come From?

System owners and system users initiate most projects. The impetus for most projects is some combination of **problems, opportunities,** and **directives.** To simplify this discussion, we will frequently use the term *problem* to collectively refer to problems, opportunities, and directives. Accordingly, *problem solving* refers to solving problems, exploiting opportunities, and fulfilling directives.

problem an undesirable situation that prevents the organization from fully achieving its mission, vision, goals, and/or objectives.

There are far too many potential system problems to list them all in this book. However, James Wetherbe developed a useful framework for classifying problems.[4] He calls it *PIECES* because the letters of each of the six categories, when put together, spell the word "pieces." The categories are:

opportunity a chance to improve the organization even in the absence of an identified problem.

- **P** the need to correct or improve *performance.*
- **I** the need to correct or improve *information* (and data).
- **E** the need to correct or improve *economics,* control costs, or increase profits.
- **C** the need to correct or improve *control* or security.
- **E** the need to correct or improve *efficiency* of people and processes.
- **S** the need to correct or improve *service* to customers, suppliers, partners, employees, and so on.

directive a new requirement that's imposed by management, government, or some external influence.

Figure 3-4 expands on each of the PIECES categories.

The categories of the PIECES framework are neither exhaustive nor mutually exclusive—they overlap. Any given project is usually characterized by one or more categories, and any given problem or opportunity may have implications with respect to more than one category. But PIECES is a practical framework (used in *FAST*), not just an academic exercise. We'll revisit PIECES several times in this book.

Projects can be either planned or unplanned. A *planned project* is the result of one of the following:

- An *information systems strategy plan* has examined the business as a whole to identify those system development projects that will return the greatest strategic (long-term) value to the business.
- A *business process redesign* has thoroughly analyzed a series of business processes to eliminate redundancy and bureaucracy and to improve efficiency and value added. Now it is time to redesign the supporting information system for those redesigned business processes.

The opposite of planned projects are *unplanned projects*—those that are triggered by a specific problem, opportunity, or directive that occurs in the course of doing business. Most organizations have no shortage of unplanned projects. Anyone can submit a proposed project based on something that is happening in the business. The number of unplanned-project proposals can easily overwhelm the largest information systems organization; therefore, they are frequently screened and prioritized by a **steering committee** of system owners and IT managers to determine which requests get approved. Those requests that are not approved are **backlogged** until resources become available (which sometimes never happens).

steering committee an administrative body of system owners and information technology executives that prioritizes and approves candidate system development projects.

Both planned and unplanned projects go through the same essential system development process. Let's now examine the project phases in somewhat greater detail.

> The *FAST* Phases

FAST, like most methodologies, consists of phases. The number of phases will vary from one methodology to another. In Chapter 1 you were introduced to the four classic phases of the system development life cycle. The *FAST* methodology employs

backlog a repository of project proposals that cannot be funded or staffed because they are a lower priority than those that have been approved for system development. Note that priorities change over time; therefore, a backlogged project might be approved at some future date.

[4] James Wetherbe and Nicholas P. Vitalari, *Systems Analysis and Design: Traditional, Best Practices,* 4th ed. (St. Paul, MN: West Publishing, 1994), pp. 196–199. James Wetherbe is a respected information systems educator, researcher, and consultant.

The PIECES Problem-Solving Framework and Checklist

The following checklist for problem, opportunity, and directive identification uses Wetherbe's PIECES framework. Note that the categories of PIECES are not mutually exclusive; some possible problems show up in multiple lists. Also, the list of possible problems is not exhaustive. The PIECES framework is equally suited to analyzing both manual and computerized systems and applications.

PERFORMANCE

A. Throughput – the amount of work performed over some period of time.
B. Response times – the average delay between a transaction or request, and a response to that transaction or request.

INFORMATION (and Data)

A. Outputs
   1. Lack of any information
   2. Lack of necessary information
   3. Lack of relevant information
   4. Too much information – "information overload"
   5. Information that is not in a useful format
   6. Information that is not accurate
   7. Information that is difficult to produce
   8. Information is not timely to its subsequent use
B. Inputs
   1. Data is not captured
   2. Data is not captured in time to be useful
   3. Data is not accurately captured – contains errors
   4. Data is difficult to capture
   5. Data is captured redundantly – same data captured more than once
   6. Too much data is captured
   7. Illegal data is captured
C. Stored data
   1. Data is stored redundantly in multiple files and/or databases
   2. Same data items have different values in different files (poor data integration)
   3. Stored data is not accurate
   4. Data is not secure to accident or vandalism
   5. Data is not well organized
   6. Data is not flexible – not easy to meet new information needs from stored data
   7. Data is not accessible

ECONOMICS

A. Costs
   1. Costs are unknown
   2. Costs are untraceable to source
   3. Costs are too high
B. Profits
   1. New markets can be explored
   2. Current marketing can be improved
   3. Orders can be increased

CONTROL (and Security)

A. Too little security or control
   1. Input data is not adequately edited
   2. Crimes (e.g., fraud, embezzlement) are (or can be) committed against data
   3. Ethics are breached on data or information – refers to data or information getting to unauthorized people
   4. Redundantly stored data is inconsistent in different files or databases
   5. Data privacy regulations or guidelines are being (or can be) violated
   6. Processing errors are occurring (either by people, machines, or software)
   7. Decision-making errors are occurring
B. Too much control or security
   1. Bureaucratic red tape slows the system
   2. Controls inconvenience customers or employees
   3. Excessive controls cause processing delays

EFFICIENCY

A. People, machines, or computers waste time
   1. Data is redundantly input or copied
   2. Data is redundantly processed
   3. Information is redundantly generated
B. People, machines, or computers waste materials and supplies
C. Effort required for tasks is excessive
D. Material required for tasks is excessive

SERVICE

A. The system produces inaccurate results
B. The system produces inconsistent results
C. The system produces unreliable results
D. The system is not easy to learn
E. The system is not easy to use
F. The system is awkward to use
G. The system is inflexible to new or exceptional situations
H. The system is inflexible to change
I. The system is incompatible with other systems

FIGURE 3-4 The PIECES Framework for Problem Identification

eight phases to better define periodic milestones and the deliverables. The grid below compares the *FAST* phases to the classic phases. As you can see, both sets of phases cover the same ground, but *FAST* is more detailed.

FAST Phases	Classic Phases			
	Project Initiation	System Analysis	System Design	System Implementation
Scope definition	X			
Problem analysis	X	X		
Requirements analysis		X		
Logical design		X		
Decision analysis	(a system analysis transition phase)			
Physical design and integration			X	
Construction and testing			X	X
Installation and delivery				X

Figure 3-5 illustrates the phases of the *FAST* methodology. Each phase produces deliverables that are passed to the next phase. And documentation accumulates as you complete each phase. Notice that we have included an iconic representation of the building blocks to symbolize this accumulation of knowledge and work-in-process artifacts during system development. Notice also that a project starts with some combination of PROBLEMS, OPPORTUNITIES, and DIRECTIVES from the user community (the green arrow) and finishes with a WORKING BUSINESS SOLUTION (the red arrow) for the user community.

Figure 3-6 shows the *FAST* methodology from the perspective of your information system building blocks that you learned in Chapters 1 and 2. The phases are on the right-hand side. The deliverables are built around the building blocks for knowledge, processes, and communications. The stakeholders are on the left-hand side. Notice how we have expanded and duplicated some stakeholders to reflect their involvement opposite the phases in which they primarily participate.

> NOTE: The remainder of this section briefly describes each of the eight basic phases. Throughout this discussion, we will be referring to the process flowchart in Figure 3-5, as well as the building blocks view of the process in Figure 3-6. Also throughout the discussion, all terms printed in SMALL CAPS refer to phases, prerequisites (inputs), and deliverables (outputs) shown in Figures 3-5 and 3-6.

Scope Definition The first phase of a typical project is SCOPE DEFINITION. The purpose of the scope definition phase is twofold. First, it answers the question, "Is this problem worth looking at?" Second, and assuming the problem *is* worth looking at, it establishes the size and boundaries of the project, the project vision, any constraints or limitations, the required project participants, and, finally, the budget and schedule.

In Figure 3-6, we see that the participants in the scope definition phase primarily include SYSTEM OWNERS, PROJECT MANAGERS, and SYSTEM ANALYSTS. System users are generally excluded because it is too early to get into the level of detail they will eventually bring to the project.

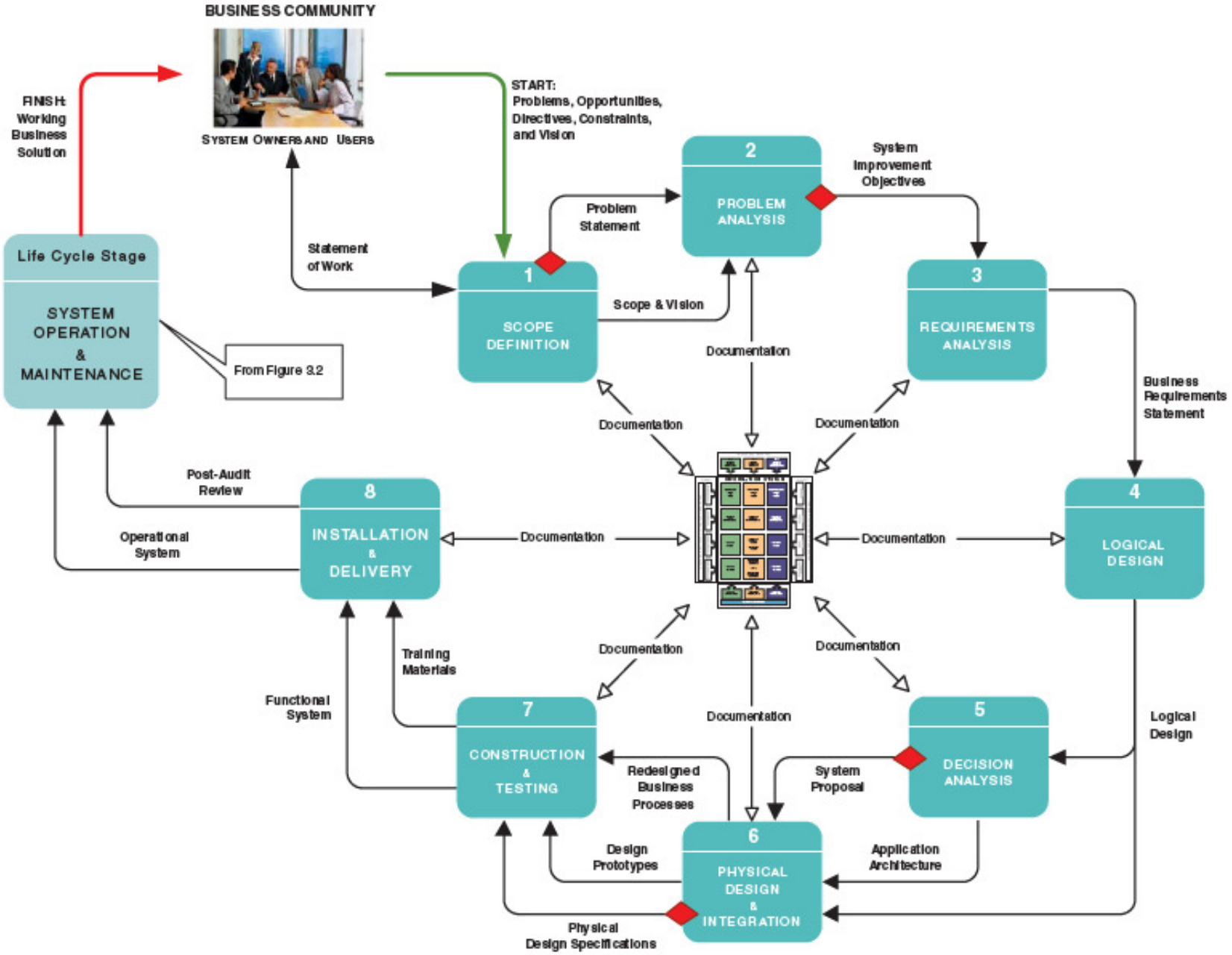


FIGURE 3-5 Process View of System Development

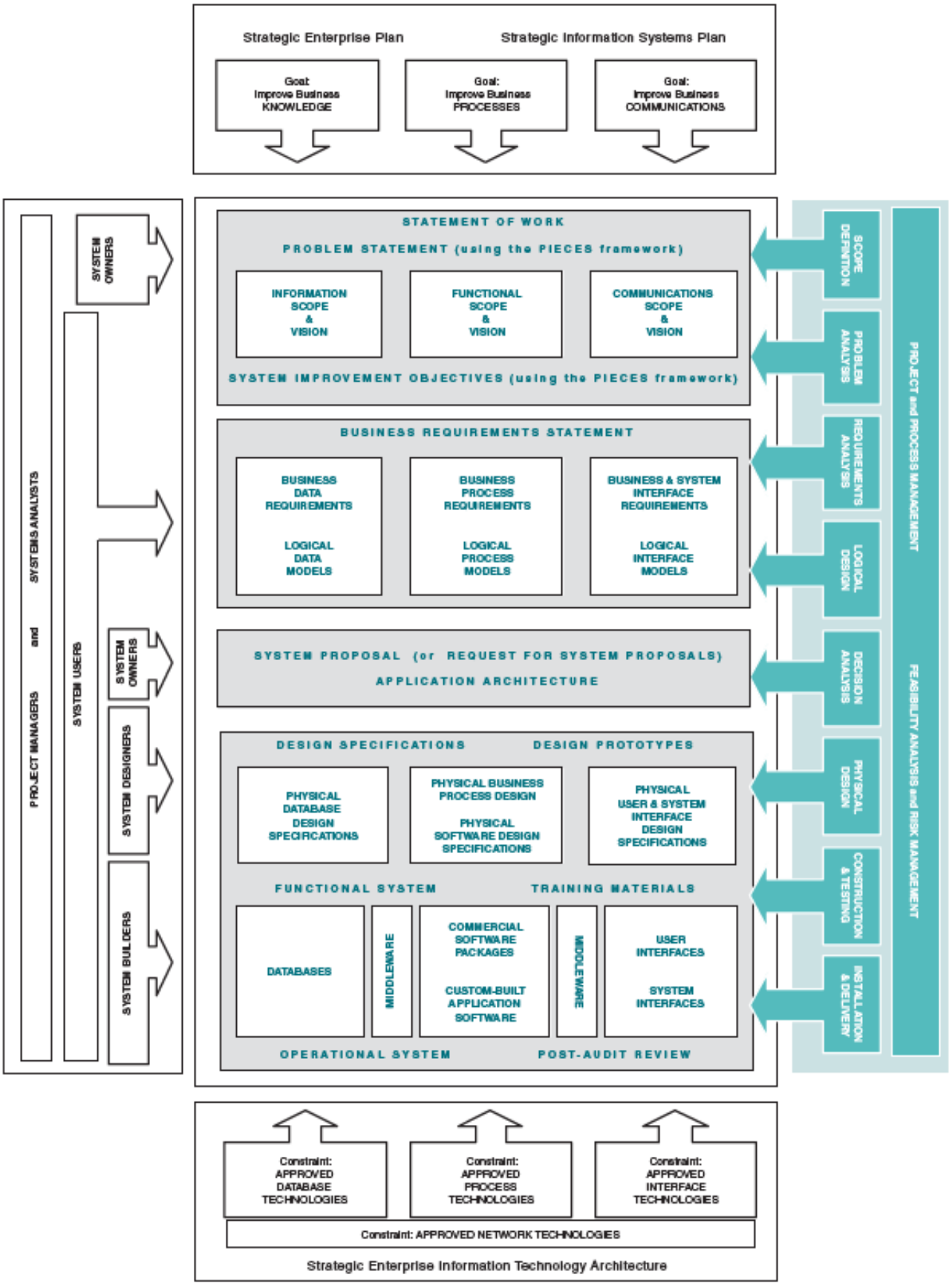


FIGURE 3-6 Building Blocks View of System Development

problem statement a statement and categorization of problems, opportunities, and directives; may also include constraints and an initial vision for the solution. Synonyms include *preliminary study and feasibility assessment*.

In Figure 3-5, we see that the scope definition phase is triggered by some combination of PROBLEMS, OPPORTUNITIES, and DIRECTIVES (to which we will add CONSTRAINTS and VISION). There are several deliverables or outcomes of a scope definition. One important outcome is a PROBLEM STATEMENT, a succinct overview of the problems, opportunities, and/or directives that triggered the project. The PIECES framework provides an excellent outline for a **problem statement.** The goal here is not to solve the problems, opportunities, and directives but only to catalog and categorize them. We should also identify any **constraints** that may impact the proposed project. Examples of constraints include budget limits, deadlines, human resources available or not available, business policies or government regulations, and technology standards. Finally, the system owners should be asked for at least a high-level vision for the system improvements they are seeking.

constraint any factor, limitation, or restraint that may limit a solution or the problem-solving process.

Given a basic understanding of problems, opportunities, directives, constraints, and vision, we need to establish initial scope. Thus, an initial SCOPE STATEMENT is another important outcome of this phase. Scope defines how big we think the project is. Your information system building blocks provide a useful framework for defining scope. Figure 3-6 illustrates that scope and vision can be defined in terms of INFORMATION, FUNCTIONS, and INTERFACES. Scope can, and frequently does, change during a project. But by documenting initial scope, you establish a baseline for controlling **scope creep** on both the budget and the schedule.

scope creep a common phenomenon wherein the requirements and expectations of a project increase, often without regard to the impact on budget and schedule.

Given the initial problem and scope statements for the project, the analyst can staff the project team, estimate the budget for system development, and prepare a schedule for the remaining phases. Ultimately, this phase concludes with a "go or no-go" decision from system owners. Either the system owners agree with the proposed scope, budget, and schedule for the project, or they must reduce scope (to reduce costs and time) or cancel the project. This feasibility checkpoint is illustrated in Figure 3-5 as a diamond.

statement of work a contract with management and the user community to develop or enhance an information system; defines vision, scope, constraints, high-level user requirements, schedule, and budget. Synonyms include *project charter, project plan,* and *service-level agreement*.

The final and most important deliverable is a STATEMENT OF WORK. **A statement of work** is a contract or agreement to develop the information system. It consolidates the problem statement, scope statement, and schedule and budget for all parties who will be involved in the project.

Problem Analysis There is always an existing system, regardless of whether it currently uses information technology. The PROBLEM ANALYSIS phase studies the existing system and analyzes the findings to provide the project team with a more thorough understanding of the problems that triggered the project. The analyst frequently uncovers new problems and answers the most important question, "Will the benefits of solving these problems exceed the costs of building the system to solve these problems?"

Once again, Figure 3-6 provides a graphical overview of the problem analysis phase in terms of your information system building blocks. Notice that the participants still include the SYSTEM OWNERS but that this phase begins to actively involve the SYSTEM USERS as well. The system users are the business subject matter experts in any project. (Notice the intentional expansion of the system users' perspective to overlap many phases—remember principle 1: "Get the system users involved.") Of course, PROJECT MANAGERS and SYSTEM ANALYSTS are always involved in all phases of a project.

As shown in Figure 3-5, the prerequisites for the problem analysis phase are the SCOPE and PROBLEM STATEMENTS as defined and approved in the scope definition phase. The deliverable of the problem analysis phase is a set of SYSTEM IMPROVEMENT OBJECTIVES derived from a thorough understanding of the business problems. These objectives do not define inputs, outputs, or processes. Instead, they define the business criteria on which any new system will be evaluated. For instance, we might define a system improvement objective as any of the following:

Reduce the time between order processing and shipping by three days.
Reduce bad credit losses by 45 percent.
Comply with new financial aid federal qualification requirements by January 1.

Think of system improvement objectives as the *grading criteria* for evaluating any new system that you might eventually design and implement. System improvement objectives may be presented to system owners and users as a written recommendation or an oral presentation.

Depending on the complexity of the problem and the project schedule, the team may or may not choose to formally document the existing system. Such documentation frequently occurs when the business processes are considered dated or overly bureaucratic. Documentation of the existing system is sometimes called an "AS IS" BUSINESS MODEL. The as-is model may be accompanied by analysis demonstrating inefficiencies, bottlenecks, or other problems related to the business processes.

Every existing system has its own terminology, history, culture, and nuances. Learning those aspects of the system is an important by-product of this phase. From all of the information gathered, the project team gains a better understanding of the existing system's problems and opportunities. After reviewing the findings, the system owners will either agree or disagree with the recommended system improvement objectives. And consistent with the creeping commitment principle, we include another go or no-go feasibility checkpoint (the red diamond) at the end of the phase. The project can be either:

- Canceled if the problems are deemed no longer worth solving.
- Approved to continue to the next phase.
- Reduced or expanded in scope (with budget and schedule modifications) and then approved to continue to the next phase.

Requirements Analysis Given system owner approval to continue from the problem analysis phase, now you can design a new system, right? No, not yet! What capabilities should the new system provide for its users? What data must be captured and stored? What performance level is expected? Careful! This requires decisions about *what* the system must do, *not how* it should do those things. The REQUIREMENTS ANALYSIS phase defines and prioritizes the *business* requirements. Simply stated, the analyst approaches the users to find out what they need or want out of the new system, carefully avoiding any discussion of technology or technical implementation. This is perhaps the most important phase of systems development. Errors and omissions in requirements analysis result in user dissatisfaction with the final system and costly modifications.

Returning again to Figure 3-6, notice that the participants primarily include both SYSTEM USERS (which may include owners who will actually *use* the system) and SYSTEMS ANALYSTS. PROJECT MANAGERS are also involved. SYSTEM DESIGNERS are omitted from this phase in order to prevent premature attention to technology solutions. The building blocks can themselves provide the framework for defining many business requirements, including BUSINESS DATA REQUIREMENTS, BUSINESS PROCESS REQUIREMENTS, and BUSINESS AND SYSTEM INTERFACE REQUIREMENTS. Because the business requirements are intended to solve problems, the PIECES framework can also provide a useful outline, this time for a requirements statement.

In Figure 3-5, we see that the SYSTEM IMPROVEMENT OBJECTIVES from the problem analysis phase are the prerequisite to the requirements analysis phase. The deliverable is a BUSINESS REQUIREMENTS STATEMENT. Again, this requirements statement does not specify any technical possibilities or solutions. The requirements statement may be a document as small as a few pages, or it may be extensive with a page or more of documentation per requirement.

To produce a business requirements statement, the systems analyst works closely with system users to identify needs and priorities. This information is collected by way of interviews, questionnaires, and facilitated meetings. The challenge to the team is to validate those requirements. The system improvement objectives provide the "grading key" for business requirements: *Does each requirement contribute to meeting one or more system improvement objectives?* Chapters 6

and 7 will introduce systems analysis tools and techniques for identifying and documenting user requirements.

Typically, requirements must also be prioritized. Priorities serve two purposes. First, if project timelines become stressed, requirements priorities can be used to rescope the project. Second, priorities can frequently be used to define iterations of design and construction to create staged releases or versions of the final product.

The requirements analysis phase should never be skipped or shortchanged. One of the most common complaints about new systems and applications is that they don't really satisfy the users' needs. This usually happens when system designers and builders become preoccupied with a technical solution before fully understanding the business needs. System designers and builders are dependent on competent systems analysts to work with users to define and document complete and accurate business requirements before applying any technology.

system model a picture of a system that represents reality or a desired reality. System models facilitate improved communication between system users, system analysts, system designers, and system builders.

Logical Design Business requirements (above) are usually expressed in words. Systems analysts have found it useful to translate those words into pictures called **system models** to validate the requirements for completeness and consistency. (Figure 3-5 is an example of a common system model called a *data flow diagram.*) System modeling implements a timeless concept: "A picture is worth a thousand words."

logical design the translation of business user requirements into a system model that depicts only the business requirements and not any possible technical design or implementation of those requirements. Common synonyms include *conceptual design* and *essential design,* the latter of which refers to modeling the "essence" of a system, or the "essential requirements" independent of any technology. The antonym of logical design is *physical design* (defined later in this chapter).

The LOGICAL DESIGN PHASE translates business requirements into system models. The term **logical design** should be interpreted as "technology independent," meaning the pictures illustrate the system independent of any possible technical solution—hence, they model business requirements that must be fulfilled by any technical solution we might want to consider.

Different methodologies require or recommend different amounts and degrees of system modeling or logical design. Prescriptive methodologies like *structured analysis and design, information engineering,* and the *Rational Unified Process (RUP)* usually require that many types and/or instances of system models be drawn in various levels of detail. Fortunately, computer-automated tools are available to assist the systems analyst in these drawing tasks. Alternatively, agile methodologies like *architected rapid application development* and *extreme programming* recommend "just enough modeling." This so-called *agile modeling* seeks to prevent the project from degenerating into a condition called **analysis paralysis.** This textbook leans toward agile methods but recognizes that complex problems may best be solved using more prescriptive approaches.

analysis paralysis a satirical term coined to describe a common project condition in which excessive system modeling dramatically slows progress toward implementation of the intended system solution.

In Figure 3-6, we see that the participants include SYSTEM ANALYSTS (who draw the models) and SYSTEM USERS (who validate the models). PROJECT MANAGERS are always included to ensure that modeling meets standards and does not deter overall project progress. We can draw (1) LOGICAL DATA MODELS that depict data and information requirements, (2) LOGICAL PROCESS MODELS that depict business processes requirements, and (3) LOGICAL INTERFACE MODELS that depict business and system interface requirements.[5]

In Figure 3-5, we see that the prerequisite to logical design is the BUSINESS REQUIREMENTS STATEMENT from the previous phase. In practice, the requirements analysis and logical design phases almost always have considerable overlap. In other words, as business requirements are identified and documented, they can be modeled. The deliverables of logical design are the LOGICAL SYSTEM MODELS AND SPECIFICATIONS themselves. Depending on the methodology used, the level of detail in the specifications will vary. For example, we may define a business rule that specifies the legitimate values for a data attribute such as *Credit Rating* or a rule that specifies the business policy for a *Credit Check.*

[5]Those of you already familiar with *object-oriented* modeling should note that object models tend to blur the boundaries of our framework somewhat, but the framework can still be applied since the problem to be solved is still driven by the three fundamental business goals illustrated in our framework. This will be demonstrated in the object-oriented analysis and design chapters of this book.

Before we move on to the next phase, we should note that the SCOPE DEFINITION, PROBLEM ANALYSIS, REQUIREMENTS ANALYSIS, and LOGICAL DESIGN PHASES are collectively recognized by most experts as *system analysis*. Some experts would also include our next phase, DECISION ANALYSIS. But we consider it to be a system analysis to system design transition phase because it makes the transition from the business concerns of system owners and users to the technology concerns of system designers and builders. And of course, systems analysts are the common thread that ensures continuity as we make this transition. Let's examine the transition.

Decision Analysis Given business requirements and the logical system models, there are usually numerous alternative ways to design a new information system to fulfill those requirements. Some of the pertinent questions include the following:

- How much of the system should be automated with information technology?
- Should we purchase software or build it ourselves (called the *make-versus-buy decision*)?
- Should we design the system for an internal network, or should we design a Web-based solution?
- What information technologies (possibly emerging) might be useful for this application?

These questions are answered in the DECISION ANALYSIS phase of the methodology. The purpose of this phase is to (1) identify candidate technical solutions, (2) analyze those candidate solutions for feasibility, and (3) recommend a candidate system as the target solution to be designed.

In Figure 3-6, we see that the decision analysis phase is positioned halfway through the development process. Half the building blocks are positioned higher, and half are positioned lower. This is consistent with the decision analysis phase's role as a transition from analysis to design—and from business concerns of SYSTEM USERS to those of SYSTEM DESIGNERS (and, ultimately, system builders). Designers (the technical experts in specific technologies) begin to play a role here along with system users and SYSTEM ANALYSTS. Analysts help to define and analyze the alternatives. Decisions are made regarding the technologies to be used as part of the application's architecture. Ultimately, SYSTEM OWNERS will have to approve or disapprove the approved decisions since they are paying for the project.

Figure 3-5 shows that a decision analysis is triggered by validated business requirements plus any logical system models and specifications that expand on those requirements. The project team solicits ideas and opinions for technical design and implementation from a diverse audience, possibly including IT software vendors. Candidate solutions are identified and characterized according to various criteria. It should be noted that many modern organizations have information technology and architecture standards that constrain the number of candidate solutions that might be considered and analyzed. (The existence of such standards is illustrated at the bottom of your information system building blocks model in Figure 3-6.) After the candidate solutions have been identified, each one is evaluated by the following criteria:

- *Technical feasibility*—Is the solution technically practical? Does our staff have the technical expertise to design and build this solution?
- *Operational feasibility*—Will the solution fulfill the user's requirements? To what degree? How will the solution change the user's work environment? How do users feel about such a solution?
- *Economic feasibility*—Is the solution cost-effective (as defined earlier in the chapter)?
- *Schedule feasibility*—Can the solution be designed and implemented within an acceptable time period?
- *Risk feasibility*—What's the probability of a successful implementation using the technology and approach?

The project team is usually looking for the *most* feasible solution—the solution that offers the best combination of technical, operational, economic, schedule, and risk feasibility. Different candidate solutions may be most feasible on a single criterion; however, one solution will usually prove *most* feasible based on all of the criteria.

The key deliverable of the decision analysis phase is a SYSTEM PROPOSAL. This proposal may be written and/or presented verbally. Several outcomes are possible. The creeping commitment feasibility checkpoint (again, the red diamond) may result in any one of the following options:

- Approve and fund the system proposal for design and construction (possibly including an increased budget and timetable if scope has significantly expanded).
- Approve or fund one of the alternative candidate solutions.
- Reject all the candidate solutions and either cancel the project or send it back for new recommendations.
- Approve a reduced-scope version of the proposed solution.

Optionally, the decision analysis phase may also produce an APPLICATION ARCHITECTURE for the approved solution. Such a model serves as a high-level blueprint (like a simple house floor plan) for the recommended or approved proposal.

Before we move on, you may have noticed in Figure 3-6 a variation on the SYSTEM PROPOSAL deliverable called a REQUEST FOR SYSTEM PROPOSALS (OR RFP). This variation is for a recommendation to purchase the hardware and/or software solution as opposed to building it in-house. We'll defer any further discussion of this option until later in the chapter when we discuss the commercial package integration variation of our basic process.

Physical Design and Integration Given approval of the SYSTEM PROPOSAL from the decision analysis phase, you can finally design the new system. The purpose of the PHYSICAL DESIGN AND INTEGRATION phase is to transform the business requirements (represented in part by the LOGICAL SYSTEM MODELS) into PHYSICAL DESIGN SPECIFICATIONS that will guide system construction. In other words, physical design addresses greater detail about *how* technology will be used in the new system. The design will be constrained by the approved ARCHITECTURAL MODEL from the previous phase. Also, design requires adherence to any internal technical design standards that ensure completeness, usability, reliability, performance, and quality.

physical design the translation of business user requirements into a system model that depicts a technical implementation of the users' business requirements. Common synonyms include *technical design* or, in describing the output, *implementation model*. The antonym of physical design is *logical design* (defined earlier in this chapter).

Physical design is the opposite of logical design. Whereas logical design dealt exclusively with business requirements independent of any technical solution, physical design represents a specific technical solution. Figure 3-6 demonstrates the physical design phase from the perspective of your building blocks. Notice that the design phase is concerned with technology-based views of the system: (1) PHYSICAL DATABASE DESIGN SPECIFICATIONS, (2) PHYSICAL BUSINESS PROCESS and SOFTWARE DESIGN SPECIFICATIONS, and (3) PHYSICAL USER AND SYSTEM INTERFACE SPECIFICATIONS. The SYSTEM DESIGNER and SYSTEM ANALYST (possibly overlapping roles for some of the same individuals) are the key participants; however, certain aspects of the design usually have to be shared with the SYSTEM USERS (e.g., screen designs and work flow). You may have already had some exposure to physical design specifications in either programming or database courses.

There are two extreme philosophies of physical design.

- *Design by specification*—Physical system models and detailed specifications are produced as a series of written (or computer-generated) blueprints for construction.
- *Design by prototyping*—Incomplete but functioning applications or subsystems (called *prototypes*) are constructed and refined based on feedback from users and other designers.

In practice, some combination of these extremes is usually performed.

No new information system exists in isolation from other existing information systems in an organization. Consequently, a design must also reflect system integration concerns. The new system must be integrated both with other information systems

and with the business's processes themselves. Integration is usually reflected in physical system models and design specifications.

In summary, Figure 3-5 shows that the deliverables of the physical design and integration phase include some combination of PHYSICAL DESIGN MODELS AND SPECIFICATIONS, DESIGN PROTOTYPES, and REDESIGNED BUSINESS PROCESSES. Notice that we have included one final go or no-go feasibility checkpoint for the project (the red diamond). A project is rarely canceled after the design phase unless it is hopelessly over budget or behind schedule. On the other hand, scope could be decreased to produce a minimum acceptable product in a specified time frame. Or the schedule could be extended to build a more complete solution in multiple versions. The project plan (schedule and budget) would need to be adjusted to reflect these decisions.

It should be noted that in modern methodologies, there is a trend toward merging the design phase with our next phase, construction. In other words, the design and construction phases usually overlap.

Construction and Testing Given some level of PHYSICAL DESIGN MODELS AND SPECIFICATIONS (and/or DESIGN PROTOTYPES), we can begin to construct and test system components for that design. Figure 3-5 shows that the primary deliverable of the CONSTRUCTION AND TESTING phase is a FUNCTIONAL SYSTEM that is ready for implementation. The purpose of the construction and testing phase is twofold: (1) to build and test a system that fulfills business requirements and physical design specifications, and (2) to implement the interfaces between the new system and existing systems. Additionally, FINAL DOCUMENTATION (e.g., help systems, training manuals, help desk support, production control instructions) will be developed in preparation for training and system operation. The construction phase may also involve installation of purchased software.

Your information system framework (Figure 3-6) identifies the relevant building blocks and activities for the construction phase. The focus is on the last row of building blocks. The project team must construct or install:

- DATABASES—Databases may include *online transaction processing (OLTP)* databases to support day-to-day business transactions, *operational data stores (ODS)* to support day-to-day reporting and queries, and *data warehouses* to support data analysis and decision support needs.
- COMMERCIAL SOFTWARE PACKAGES and/or CUSTOM-BUILT SOFTWARE—Packages are installed and customized as necessary. Application programs are constructed according to the physical design and/or prototypes from the previous phase. Both packages and custom software must be thoroughly tested.
- USER AND SYSTEM INTERFACES—User interfaces (e.g., Windows and Web interfaces) must be constructed and tested for usability and stability. System-to-system interfaces must be either constructed or implemented using application integration technologies. Notice that MIDDLEWARE (a type of system software) is often used to integrate disparate database, software, and interface technologies. We'll talk more about middleware in the design unit of this book.

Figure 3-6 also identifies the participants in this phase as SYSTEM BUILDERS, SYSTEM ANALYSTS, SYSTEM USERS, and PROJECT MANAGERS. SYSTEM DESIGNERS may also be involved to clarify design specifications.

You probably already have some experience with part of this activity—application programming. Programs can be written in many different languages, but the current trend is toward the use of visual and object-oriented programming languages such as *Java, C++*, or *Visual Basic*. As components are constructed, they are typically demonstrated to users in order to solicit feedback.

One of the most important aspects of construction is conducting tests of both individual system components and the overall system. Once tested, a system (or version of a system) is ready for INSTALLATION AND DELIVERY.

Installation and Delivery What's left to do? New systems usually represent a departure from the way business is currently done; therefore, the analyst must provide

for a smooth transition from the old system to the new system and help users cope with normal start-up problems. Thus, the INSTALLATION AND DELIVERY phase serves to deliver the system into operation (sometimes called *production*).

In Figure 3-5, the FUNCTIONAL SYSTEM from the construction and testing phase is the key input to the INSTALLATION AND DELIVERY phase. The deliverable is an OPERATIONAL SYSTEM. SYSTEM BUILDERS install the system from its development environment into the production environment. SYSTEM ANALYSTS must train SYSTEM USERS, write various user and production control manuals, convert existing files and databases to the new databases, and perform final system testing. Any problems may initiate rework in previous phases thought to be complete. System users provide continuous feedback as new problems and issues arise. Essentially, the installation and delivery phase considers the same building blocks as the construction phase.

To provide a smooth transition to the new system, a conversion plan should be prepared. This plan may call for an abrupt cutover, where the old system is terminated and replaced by the new system on a specific date. Alternatively, the plan may run the old and new systems in parallel until the new system has been deemed acceptable to replace the old system.

The installation and delivery phase also involves training individuals who will use the final system and developing documentation to aid the system users. The implementation phase usually includes some form of POST-AUDIT REVIEW to gauge the success of the completed systems project. This activity promotes continuous improvement of the process and future project management.

System Operation and Maintenance Once the system is placed into operation, it will require ongoing **system support** for the remainder of its useful, productive lifetime. System support consists of the following ongoing activities:

system support the ongoing technical support for users of a system, as well as the maintenance required to deal with any errors, omissions, or new requirements that may arise.

- *Assisting users*—Regardless of how well the users have been trained and how thorough and clear the end-user documentation is, users will eventually require additional assistance as unanticipated problems arise, new users are added, and so forth.
- *Fixing software defects (bugs)*—Software defects are errors that slipped through the testing of software. These are inevitable, but they can usually be resolved, in most cases, by knowledgeable support.
- *Recovering the system*—From time to time, a system failure may result in a program "crash" and/or loss of data. Human error or a hardware or software failure may cause this. The systems analyst or technical support specialists may then be called on to recover the system—that is, to restore a system's files and databases and to restart the system.
- *Adapting the system to new requirements*—New requirements may include new business problems, new business requirements, new technical problems, or new technology requirements.

Eventually, we expect that the user feedback and problems, or changing business needs, will indicate that it is time to start over and reinvent the system. In other words, the system has reached entropy, and a new project to create an entirely new system development process should be initiated.

cross life-cycle activity any activity that overlaps multiple phases of the system development process. Examples include *fact-finding, documentation, presentation, estimation, feasibility analysis, project and process management, change management,* and *quality management.*

> Cross Life-Cycle Activities

System development also involves a number of **cross life-cycle activities.** These activities, listed in the margin definition, are not explicitly depicted in Figure 3-5, but they are vital to the success of any project. Let's briefly examine each of these activities.

fact-finding the formal process of using research, interviews, meetings, questionnaires, sampling, and other techniques to collect information about system problems, requirements, and preferences. It is also called *information gathering* or *data collection.*

Fact-Finding There are many occasions for **fact-finding** during a project. Fact-finding is most crucial to the early phases of a project. It is during these phases that

the project team learns about a business's vocabulary, problems, opportunities, constraints, requirements, and priorities. But fact-finding is also used during the decision analysis, physical design, construction and testing, and installation and delivery phases—only to a lesser extent. It is during these latter phases that the project team researches technical alternatives and solicits feedback on technical designs, standards, and working components.

documentation the ongoing activity of recording facts and specifications for a system for current and future reference.

Documentation and Presentation Communication skills are essential to the successful completion of any project. In fact, poor communication is frequently cited as the cause of project delays and rework. Two forms of communication that are common to systems development projects are **documentation** and **presentation.**

presentation the ongoing activity of communicating findings, recommendations, and documentation for review by interested users and managers. Presentations may be either written or verbal.

Clearly, documentation and presentation opportunities span all the phases. In Figure 3-7, the black arrows represent various instances of documentation of a phase. The red arrows represent instances where presentations are frequently required. Finally, the green arrows represent the storage of documentation and other artifacts of systems development in a **repository.** A repository saves documentation for reuse and rework as necessary.

repository a database and/or file directory where system developers store all documentation, knowledge, and artifacts for one or more information systems or projects. A repository is usually automated for easy information storage, retrieval, and sharing.

Feasibility Analysis Consistent with our creeping commitment approach to systems development, **feasibility analysis** is a cross life-cycle activity. Different measures of **feasibility** are applicable in different phases of the methodology. These measures include technical, operational, economic, schedule, and risk feasibility, as described when we introduced the decision analysis phase. Feasibility analysis requires good **estimation** techniques.

feasibility analysis the activity by which feasibility is measured and assessed.

Process and Project Management Recall that the CMM considers systems development to be a *process* that must be managed on a project-by-project basis. For this reason and others, process management and project management are ongoing, cross life-cycle activities. Both types of management were introduced earlier, but their definitions are repeated in the margin on page ••• for your convenience. **Process management** defines the methodology to be used on every project—think of it as the *recipe* for building a system. **Project management** is concerned with administering a single instance of the process as applied to a single project.

feasibility a measure of how beneficial the development of an information system would be to an organization.

Failures and limited successes of systems development projects often outnumber successful projects. Why is that? One reason is that many systems analysts are unfamiliar with, or undisciplined in how to properly apply, tools and techniques of systems development. But most failures are attributed to poor leadership and management. This mismanagement results in unfulfilled or unidentified requirements, cost overruns, and late delivery.

estimation the calculated prediction of the costs and effort required for system development. A somewhat facetious synonym is *guesstimation,* usually meaning that the estimation is based on experience or empirical evidence but is lacking in rigor—in other words, a *guess.*

> Sequential versus Iterative Development

The above discussion of phases might lead you to assume that systems development is a naturally sequential process, moving in a one-way direction from phase to phase. Such sequential development is, in fact, one alternative. This approach is depicted in part (a) of Figure 3-8. In the figure we have used the four classic phases rather than the eight *FAST* phases in the interest of simplicity. This strategy requires that each phase be "completed" one after the other until the information system is finished. In reality, the phases may somewhat overlap one another in time. For example, some system design can be started prior to the completion of system analysis. Given its waterfall-like visual appearance, this approach is often called the **waterfall development approach.**

process management an ongoing activity that documents, teaches, oversees the use of, and improves an organization's chosen methodology (the "process") for systems development. Process management is concerned with phases, activities, deliverables, and quality standards that should be consistently applied to all projects.

The waterfall approach has lost favor with most modern system developers. A more popular strategy, shown in part (b) of Figure 3-8, is commonly referred to as the **iterative development approach,** or incremental development process. This

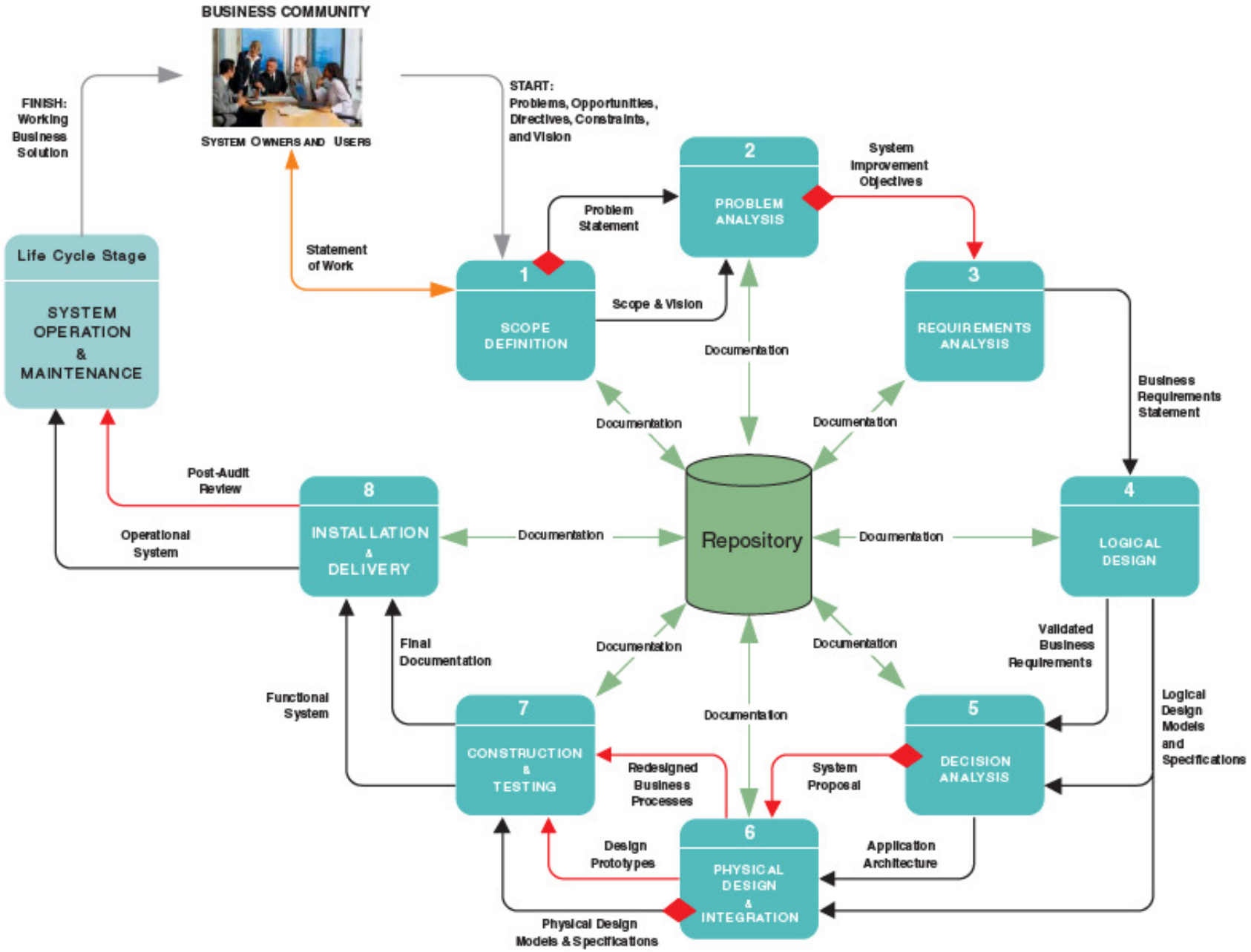


FIGURE 3-7 System Development Documentation, Repository, and Presentation

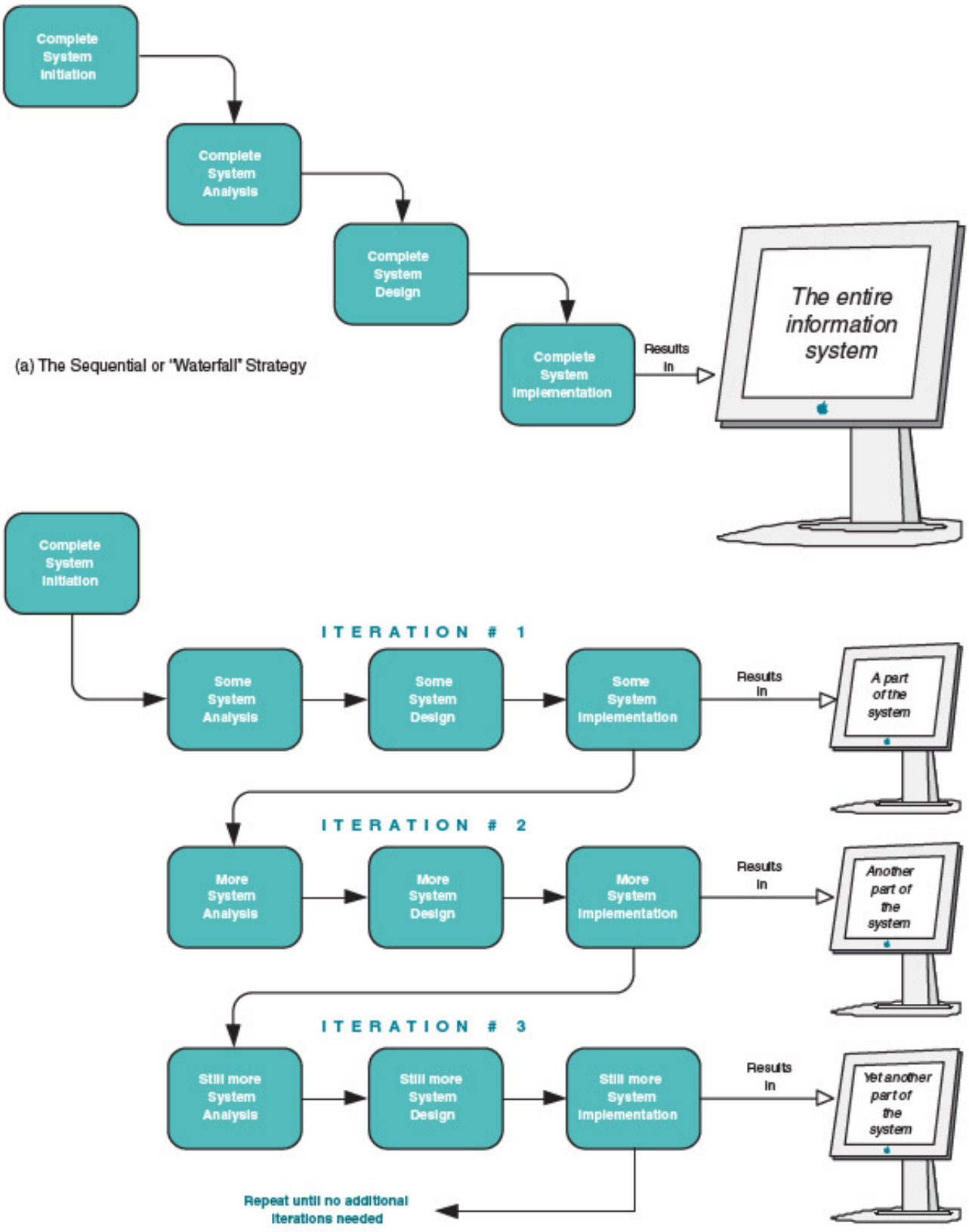


FIGURE 3-8 Sequential versus Iterative Systems Development Approach

project management the process of scoping, planning, staffing, organizing, directing, and controlling a project to develop an information system at minimum cost, within a specified time frame, and with acceptable quality.

approach requires completing enough analysis, design, and implementation to be able to fully develop a *part* of the new system and place it into operation as quickly as possible. Once that version of the system is implemented, the strategy is to then perform some additional analysis, design, and implementation to release the next version of the system. These iterations continue until all parts of the entire information system have been implemented. The popularity of this iterative and incremental process can be explained simply: System owners and users have long complained about the excessive time required to develop and implement information systems using the waterfall approach. The iterative approach allows versions of useable information to be delivered in regular and shorter time frames. This results in improved customer (system owner and user) satisfaction.

Alternative Routes and Strategies

waterfall development approach an approach to systems analysis and design that completes each phase one after another and only once.

iterative development approach an approach to systems analysis and design that completes that entire information system in successive iterations. Each iteration does some analysis, some design, and some construction. Synonyms include incremental and spiral.

Given any destination, there are many routes to that destination and many modes of transport. You could take the superhighway, highways, or back roads, or you could fly. Deciding which route is best depends on your goals and priorities. Do you want to get there fast, or do you want to see the sights? How much are you willing to spend? Are you comfortable with the mode of travel? Just as you would pick your route and means to a travel destination, you can and should pick a route and means for a systems development destination.

So far, we've described a basic set of phases that comprise our *FAST* methodology. At one time, a "one size fits all" methodology was common for most projects; however, today a variety of types of projects, technologies, and development strategies exist—one size no longer fits all projects! Like many contemporary methodologies, *FAST* provides alternative routes and strategies to accommodate different types of projects, technology goals, developer skills, and development paradigms.

In this section, we will describe several *FAST* routes and strategies. Before we do so, examine Figure 3-9 on the following page. The figure illustrates a taxonomy or classification scheme for methodological strategies. Notice the following:

- Methodologies and routes can support the option of either *building software solutions* in-house or *buying a commercial software solution* from a software vendor. Generally, many of the same methods and techniques are applicable to both options.
- Methodologies may be either very *prescriptive* ("Touch all the bases; follow all the rules") or relatively *adaptive* ("Change as needed within certain guidelines").
- Methodologies can also be characterized as *model-driven* ("Draw pictures of the system") or *product-driven* ("Build the product and see how the users react").
- Model-driven methodologies are rapidly moving to a focus on the *object-oriented* technologies being used to construct most of today's systems (more about this later). Earlier model-driven approaches emphasized either process modeling *or* data modeling.
- Finally, product-driven approaches tend to emphasize either rapid *prototyping* or writing program *code* as soon as possible (perhaps you've heard the term *extreme programming*).

So many strategies! Which should you choose? A movement is forming known as *agile methods*. In a nutshell, advocates of agile methods suggest that system analysts and programmers should have a tool box of methods that include tools and techniques from all of the above methodologies. They should choose their tools and techniques based on the problem and situation. *FAST* is an agile methodology. It advocates the integrated use of tools and techniques from many methodologies, applied in the context of repeatable processes (as in CMM Level 3). That said, let's examine some of the route variations and strategies for the *FAST* process. As we

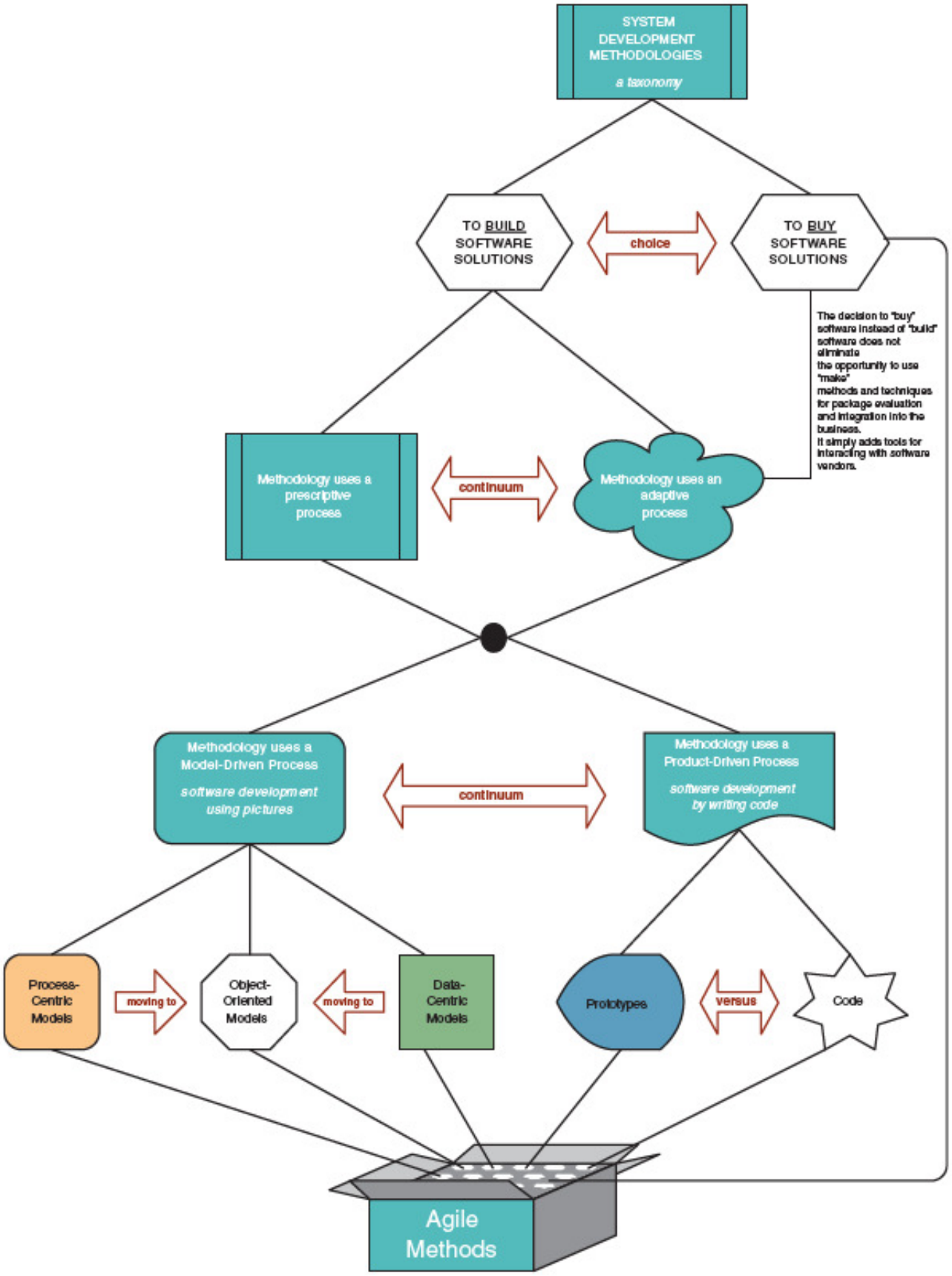


FIGURE 3-9 A Taxonomy for System Development Methodologies and Strategies

navigate through each route, we will use red typefaces and arrows to highlight those aspects of the route that differ from the basic route you've already learned.

> The Model-Driven Development Strategy

One of the oldest and most commonly used approaches to analyzing and designing information systems is based on system modeling. As a reminder, a system model is a picture of a system that represents reality or a desired reality. System models facilitate improved communication between system users, system analysts, system designers, and system builders. In the *FAST* methodology, system models are used to illustrate and communicate the KNOWLEDGE, PROCESS, or INTERFACE building blocks of information systems. This approach is called **model-driven development.**

model-driven development a system development strategy that emphasizes the drawing of system models to help visualize and analyze problems, define business requirements, and design information systems.

The model-driven development route for *FAST* is illustrated in Figure 3-10. The model-driven approach does not vary much from the basic phases we described earlier. We call your attention to the following notes that correspond to the numbered bullets:

1. System models may exist from the project that created the current system. Be careful! These models are notorious for being out of date. But they can still be useful as a point of departure.
2. Earlier you learned that it is important to define scope for a project. One of the simplest ways to communicate scope is by drawing MODELS THAT SHOW SCOPE DEFINITION. Scope models show which aspects of a problem are within scope and which aspects are outside scope. This is sometimes called a *context diagram* or context model.
3. Some system modeling techniques call for extensive MODELS OF THE EXISTING SYSTEM to identify problems and opportunities for system improvement. This is sometimes called the *as-is system model.* Modeling of the current system has waned in popularity today. Many project managers and analysts view it as counterproductive or of little value added. The exception is modeling of as-is business processes for the purpose of business process redesign.
4. The requirements statement is one of the most important deliverables of system development. It sometimes includes MODELS THAT DEPICT HIGH-LEVEL BUSINESS REQUIREMENTS. One of the most popular modeling techniques today is called *use case* (introduced in Chapter 7). Use cases identify requirements and track their fulfillment through the life cycle.
5. Most model-driven techniques require that analysts document business requirements with **logical models** (defined earlier). Business requirements are frequently expressed in LOGICAL MODELS THAT DEPICT MORE DETAILED USER REQUIREMENTS. They show only *what* a system must be or must do. They are implementation *in*dependent; that is, they depict the system independent of any possible technical implementation. Hence, they are useful for depicting and validating business requirements.
6. As a result of the decision analysis phase, the analyst may produce system MODELS THAT DEPICT APPLICATION ARCHITECTURE. Such models illustrate the planned technical implementation of a system.
7. Many model-driven techniques require that analysts develop MODELS THAT DEPICT PHYSICAL DESIGN SPECIFICATIONS (defined earlier in this chapter). Recall that **physical models** show not only what a system is or does but also *how* the system is implemented with technology. They are implementation *de*pendent because they reflect technology choices and the limitations of those technology choices. Examples include database schemas, structure charts, and flowcharts. They serve as a blueprint for construction of the new system.
8. New information systems must be interwoven into the fabric of an organization's business processes. Accordingly, the analyst and users may develop MODELS OF REDESIGNED BUSINESS PROCESSES.

logical model a pictorial representation that depicts *what* a system is or does. Synonyms include essential model, conceptual model, and business model.

physical model a technical pictorial representation that depicts what a system is or does and *how* the system is implemented. Synonyms include implementation model and technical model.

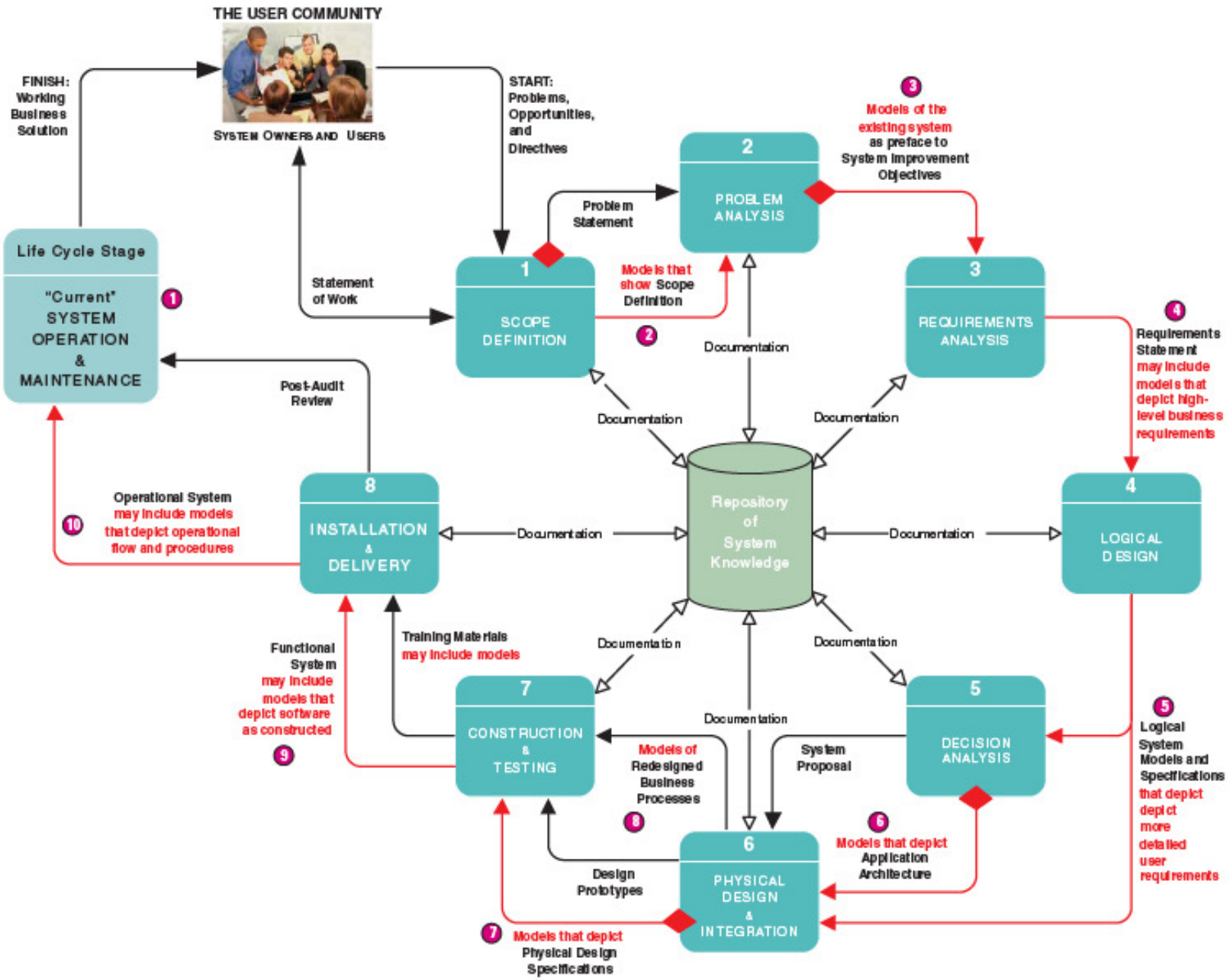


FIGURE 3-10 The Model-Driven System Development Strategy

9. Construction translates the physical system models into software. In some cases, automated tools exist to automatically translate software into PHYSICAL MODELS THAT DEPICT SOFTWARE CONSTRUCTED. This is called *reverse engineering*.
10. Finally, the operational system may include MODELS THAT DEPICT FLOW AND PROCEDURE. For example, system models may document backup and recovery procedures.

In summary, system models can be produced as a portion of the deliverables for most phases. Model-driven approaches emphasize system modeling. Once implemented, the system models serve as documentation for any changes that might be needed during the operation and support stage of the life cycle.

The model-driven approach is believed to offer several advantages and disadvantages, as listed below:

Advantages

- Requirements specification tends to be more thorough and better documented.
- Business requirements and system designs are easier to validate with pictures than words.
- It is easier to identify, conceptualize, and analyze alternative technical solutions.
- Design specifications tend to be more sound, stable, adaptable, and flexible because they are model based and more thoroughly analyzed *before* they are built.
- Systems can be constructed more correctly the first time when built from thorough and clear model based specifications. Some argue that code-generating software can automatically generate skeleton or near-complete code from good system models.

Disadvantages

- It is time-consuming. It takes time to collect the facts, draw the models, and validate those models. This is especially true if users are uncertain or imprecise about their system requirements.
- The models can only be as good as the users' understanding of those requirements.
- Pictures are not software—some argue that this reduces the users' role in a project to passive participation. Most users don't get excited about pictures. Instead, they want to see working software, and they gauge project progress by the existence of software (or its absence).
- The model-driven approach is considered by some to be inflexible—users must fully specify requirements before design, design must fully document technical specifications before construction, and so forth. Some view such rigidity as impractical.

process modeling a process-centered technique popularized by the *structured analysis and design* methodology that used models of business process requirements to derive effective software designs for a system. Structured analysis introduced a modeling tool called the *data flow diagram* to illustrate the flow of data through a series of business processes. Structured design converted data flow diagrams into a process model called *structure charts* to illustrate a top-down software structure that fulfills the business requirements.

Model-driven development is most effective for systems for which requirements are well understood and which are so complex that they require large project teams to complete. The approach also works well when fulfillment of user expectations and quality is more important than cost and schedule.

There are several different model-driven techniques. They differ primarily in terms of the types of models that they require the systems analyst to draw and validate. Let's briefly examine three of the most popular model-driven development techniques that will be taught in this book. Please note that we are introducing only the techniques here, not the models. We'll teach the models themselves later, in the "how to" chapters.

Process Modeling **Process modeling** was founded in the structured analysis and design methodologies in 1978. While structured analysis and design has lost favor as a

methodology, process modeling remains a viable and important technique. Recall that your information system building blocks include several possible focuses: KNOWLEDGE, PROCESSES, and INTERFACES. Process modeling focuses on the PROCESS column of building blocks. *Flowcharts* are one type of process model (used primarily by SYSTEM BUILDERS) that you may have encountered in a programming course. Process modeling has enjoyed something of a renaissance with the emergence of business process redesign (introduced in Chapter 1).

Data flow diagrams and structure charts have contributed significantly to reducing the communications gap that often exists between nontechnical system owners and users and technical system designers and builders. Process modeling is taught in this book.

Data Modeling Recall that KNOWLEDGE improvement is a fundamental goal and set of building blocks in your framework. Knowledge is the product of *information,* which in turn is the product of *data.* **Data modeling** methods emphasize the knowledge building blocks, especially data. In the data modeling approach, emphasis is placed on diagrams that capture business data requirements and translate them into database designs. Arguably, data modeling is the most widely practiced system modeling technique. Hence, it will be taught in this book.

data modeling a data-centered technique used to model business data requirements and design database systems that fulfill those requirements. The most frequently encountered data models are *entity relationship diagrams.*

Object Modeling Object modeling is the result of technical advancement. Today, most programming languages and methods are based on the emergence of object technology. While the concepts of object technology are covered extensively throughout this book, a brief but oversimplified introduction is appropriate here.

For the past 30 years, techniques like process and data modeling deliberately separated the concerns of PROCESSES from those of DATA. In other words, process and data models were separate and distinct. Because virtually all systems included processes *and* data, the techniques were frequently used in parallel and the models had to be carefully synchronized. Object techniques are an attempt to eliminate the separation of concerns, and hence the need for synchronization of data and process concerns. This has given rise to **object modeling** methods.

Business objects correspond to real things of importance in the business such as *customers* and the *orders* they place for *products.* Each object consists of *both* the data that describes the object and the processes that can create, read, update, and delete that object. With respect to your information system building blocks, object-oriented analysis and design (OOAD) significantly changes the paradigm. The DATA and PROCESS columns (and, arguably, the INTERFACE column as well) are essentially merged into a single OBJECT column. The models then focus on identifying objects, building objects, and assembling appropriate objects, as with *Legos,* into useful information systems.

object modeling a technique that attempts to merge the data and process concerns into singular constructs called *objects.* Object models are diagrams that document a system in terms of its objects and their interactions. Object modeling is the basis for *object-oriented analysis and design* methodologies.

The current popularity of object technology is driving the interest in object models and OOAD. For example, most of today's popular operating systems like Microsoft *Windows* and Apple *Mac/OS* have object-oriented user interfaces ("point and click," using objects such as windows, frames, drop-down menus, radio buttons, checkboxes, scroll bars, and the like). Web user interfaces like Microsoft *Internet Explorer* and Netscape *Navigator* are also based on object technology. Object programming languages such as *Java, C++, C#, Smalltalk,* and *Visual Basic .NET* are used to construct and assemble such object-oriented operating systems and applications. And those same languages have become the tools of choice for building next-generation information system applications. Not surprisingly, object modeling techniques have been created to express business and software requirements and designs in terms of objects. This edition of this book extensively integrates the most popular object modeling techniques to prepare you for systems analysis and design that ultimately produces today's object-based information systems and applications.

> The Rapid Application Development Strategy

rapid application development (RAD) a system development strategy that emphasizes speed of development through extensive user involvement in the rapid, iterative, and incremental construction of a series of functioning **prototypes** of a system that eventually evolves into the final system (or a version).

In response to the faster pace of the economy in general, **rapid application development (RAD)** has become a popular route for accelerating systems development. The basic ideas of RAD are:

- To more actively involve system users in the analysis, design, and construction activities.
- To organize systems development into a series of focused, intense workshops jointly involving SYSTEM OWNERS, USERS, ANALYSTS, DESIGNERS, and BUILDERS.
- To accelerate the requirements analysis and design phases through an iterative construction approach.
- To reduce the amount of time that passes before the users begin to see a working system.

prototype a small-scale, representative, or working model of the users' requirements or a proposed design for an information system. Any given prototype may omit certain functions or features until such time as the prototype has sufficiently evolved into an acceptable implementation of requirements.

The basic principle behind prototyping is that users know what they want when they see it working. In RAD, a **prototype** eventually evolves into the final information system. The RAD route for *FAST* is illustrated in Figure 3-11. Again, the red text and flows indicate the deviations from the basic *FAST* process. We call your attention to the following notes that correspond to the numbered bullets:

1. The emphasis is on reducing time in developing applications and systems; therefore, the initial problem analysis, requirements analysis, and decision analysis phases are consolidated and accelerated. The deliverables are typically abbreviated, again in the interest of time. The deliverables are said to be INITIAL, meaning "expected to change" as the project progresses.

   After the above initial analysis, the RAD uses an iterative approach, as discussed earlier in the chapter. Each iteration emphasizes only enough new functionality to be accomplished within a few weeks.
2. LOGICAL AND PHYSICAL DESIGN SPECIFICATIONS are usually significantly abbreviated and accelerated. In each iteration of the cycle, only some design specifications will be considered. While some system models may be drawn, they are selectively chosen and the emphasis continues to be on rapid development. The assumption is that errors can be caught and fixed in the next iteration.
3. In some, but rarely all, iterations, some business processes may need to be redesigned to reflect the likely integration of the evolving software application.
4. In each iteration of the cycle, SOME DESIGN PROTOTYPES OF SOME PARTIAL FUNCTIONAL SYSTEM elements are constructed and tested. Eventually, the completed application will result from the final iteration through the cycle.
5. After each prototype or partial functional subsystem is constructed and tested, system users are given the opportunity to experience working with that prototype. The expectation is that users will clarify requirements, identify new requirements, and provide BUSINESS FEEDBACK on design (e.g., ease of learning, ease of use) for the next iteration through the RAD cycle.
6. After each prototype or functioning subsystem is constructed and tested, system analysts and designers will review the application architecture and design to provide TECHNICAL FEEDBACK and direction for the next iteration through the RAD cycle.
7. Based on the feedback, systems analysts will identify REFINED SYSTEM IMPROVEMENT OBJECTIVES and/or BUSINESS REQUIREMENTS. This analysis tends to focus on revising or expanding objectives and requirements and identifying user concerns with the design.
8. Based on the feedback, systems analysts and system designers will identify a REFINED APPLICATION ARCHITECTURE and/or DESIGN CHANGES.
9. Eventually, the system (or a version of the system) will be deemed worthy of implementation. This CANDIDATE RELEASE VERSION OF THE FUNCTIONAL SYSTEM is system tested and placed into operation. The next version of the system may continue iterating through the RAD cycle.

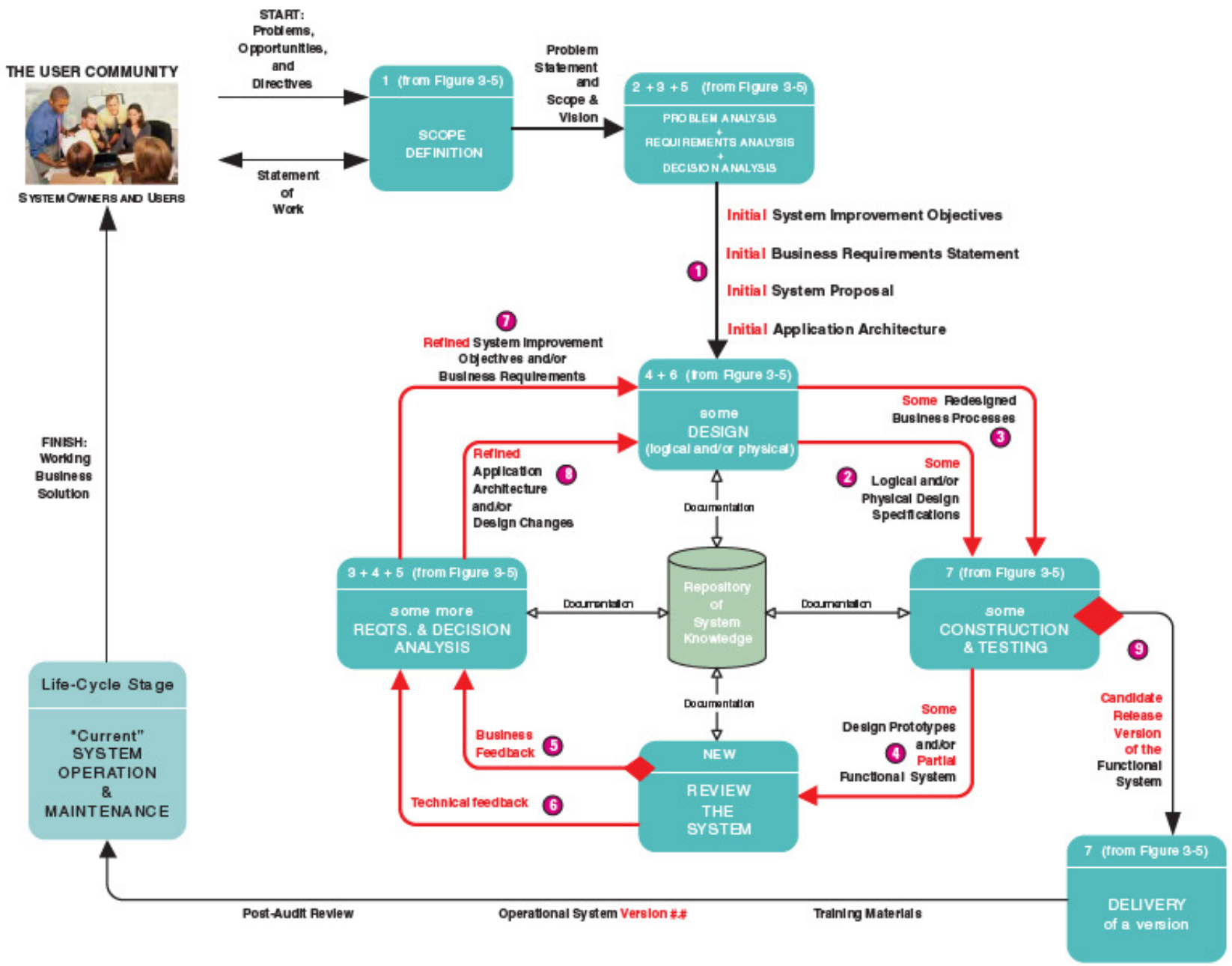


FIGURE 3-11 The Rapid Application Development (RAD) Strategy

timeboxing the imposition of a nonextendable period of time, usually 60 to 90 days, by which the first (or next) version of a system must be delivered into operation.

Although not a rigid requirement for RAD, the duration of the prototyping loop can be limited using a technique called **timeboxing.** Timeboxing seeks to deliver an operational system to users and management on a regular, recurring basis. Advocates of timeboxing argue that management and user enthusiasm for a project can be enhanced and sustained because a working version of the system is implemented on a regular basis.

The RAD approach offers several advantages and disadvantages:

Advantages

- It is useful for projects in which the user requirements are uncertain or imprecise.
- It encourages active user and management participation (as opposed to a passive reaction to nonworking system models). This increases end-user enthusiasm for the project.
- Projects have higher visibility and support because of the extensive user involvement throughout the process.
- Users and management see working, software-based solutions more rapidly than they do in model-driven development.
- Errors and omissions tend to be detected earlier in prototypes than in system models.
- Testing and training are natural by-products of the underlying prototyping approach.
- The iterative approach is a more natural process because change is an expected factor during development.

Disadvantages

- Some argue that RAD encourages a "code, implement, and repair" mentality that increases lifetime costs required to operate, support, and maintain the system.
- RAD prototypes can easily solve the wrong problems since problem analysis is abbreviated or ignored.
- A RAD-based prototype may discourage analysts from considering other, more worthy technical alternatives.
- Sometimes it is best to throw a prototype away, but stakeholders are often reluctant to do so because they see this as a loss of time and effort in the current product.
- The emphasis on speed can adversely impact quality because of ill-advised shortcuts through the methodology.

RAD is most popular for small- to medium-size projects. We will demonstrate prototyping and RAD techniques in appropriate chapters of this book.

> The Commercial Application Package Implementation Strategy

commercial application package a software application that can be purchased and customized (within limits) to meet the business requirements of a large number of organizations or a specific industry. A synonym is *commercial off-the-shelf (COTS) system*.

Sometimes it makes more sense to buy an information system solution than to build one in-house. In fact, many organizations increasingly expect to build software in-house only when there is a competitive advantage to be gained. And for many core applications such as human resources, financials, procurement, manufacturing, and distribution, there is little competitive value in building your own system—hence a **commercial application package** is purchased. Accordingly, our *FAST* methodology includes a commercial software package route.

The ultimate commercial solution is enterprise resource planning, or ERP (defined in Chapter 1). ERP solutions provide *all* of the core information system applications for an entire business. For most organizations, an ERP implementation represents the single largest information system project ever undertaken by the organization. It can cost tens or hundreds of millions of dollars and require a small army of managers, users, analysts, technical specialists, programmers, consultants, and contractors.

The *FAST* methodology's route for commercial application package integration is not really intended for ERP projects. Indeed, most ERP vendors provide their own implementation methodology (and consulting partners) to help their customers implement such a massive software solution. Instead, our *FAST* methodology provides a route for implementing all other types of information system solutions that may be purchased by a business. For example, an organization might purchase a commercial application package for a single business function such as accounting, human resources, or procurement. The package must be selected, installed, customized, and integrated into the business and its other existing information systems.

The basic ideas behind our commercial application package implementation route are:

- Packaged software solutions must be carefully selected to fulfill business needs—"You get what you ask and pay for."
- Packaged software solutions not only are costly to purchase but can be costly to implement. In fact, the package route can actually be more expensive to implement than an in-house development route.
- Software packages must usually be customized for and integrated into the business. Additionally, software packages usually require the redesign of existing business processes to adapt to the software.
- Software packages rarely fulfill all business requirements to the users' complete satisfaction. Thus, some level of in-house systems development is necessary in order to meet the unfulfilled requirements.

The commercial application package implementation route is illustrated in Figure 3-12. Once again, the red typeface and arrows indicate differences from the basic *FAST* process. We call your attention to the following notes that correspond to the numbered bullets:

1. It should be noted that the decision to purchase a package is determined in the problem analysis phase. The red diamond represents the "make versus buy" decision. The remainder of this discussion assumes that a decision to buy has been approved.
2. Problem analysis usually includes some initial TECHNOLOGY MARKET RESEARCH to identify what package solutions exist, what features are in the software, and what criteria should be used to evaluate such application packages. This research may involve software vendors, IT research services (such as the Gartner Group), or consultants.
3. After defining business requirements, the requirements must be communicated to the software vendors who offer viable application solutions. The business (and technical) requirements are formatted and communicated to candidate software vendors as either a **REQUEST FOR PROPOSAL (RFP)** or a **REQUEST FOR QUOTATION (RFQ).** The double-ended arrow implies that there may need to be some clarification of requirements and criteria.
4. Vendors submit PROPOSALS or QUOTATIONS for their application solutions. These proposals are evaluated against the business and technical requirements specified in the RFP. The double-ended arrow indicates that claimed features and capabilities must be validated and in some instances clarified. This occurs during the decision analysis phase.
5. A CONTRACT AND ORDER is negotiated with the winning vendor for the software and possibly for services necessary to install and maintain the software.
6. The vendor provides the BASELINE COMMERCIAL APPLICATION software and documentation. Services for installation and implementation of the software are frequently provided by the vendor or its service providers (certified consultants).
7. When an application package is purchased, the organization must nearly always change its business processes and practices to work efficiently with the package. The need for REDESIGNED BUSINESS PROCESSES is rarely greeted with enthusiasm,

request for proposal (RFP) a formal document that communicates business, technical, and support requirements for an application software package to vendors that may wish to compete for the sale of that application package and services.

request for quotation (RFQ) a formal document that communicates business, technical, and support requirements for an application software package to a single vendor that has been determined as being able to supply that application package and services.

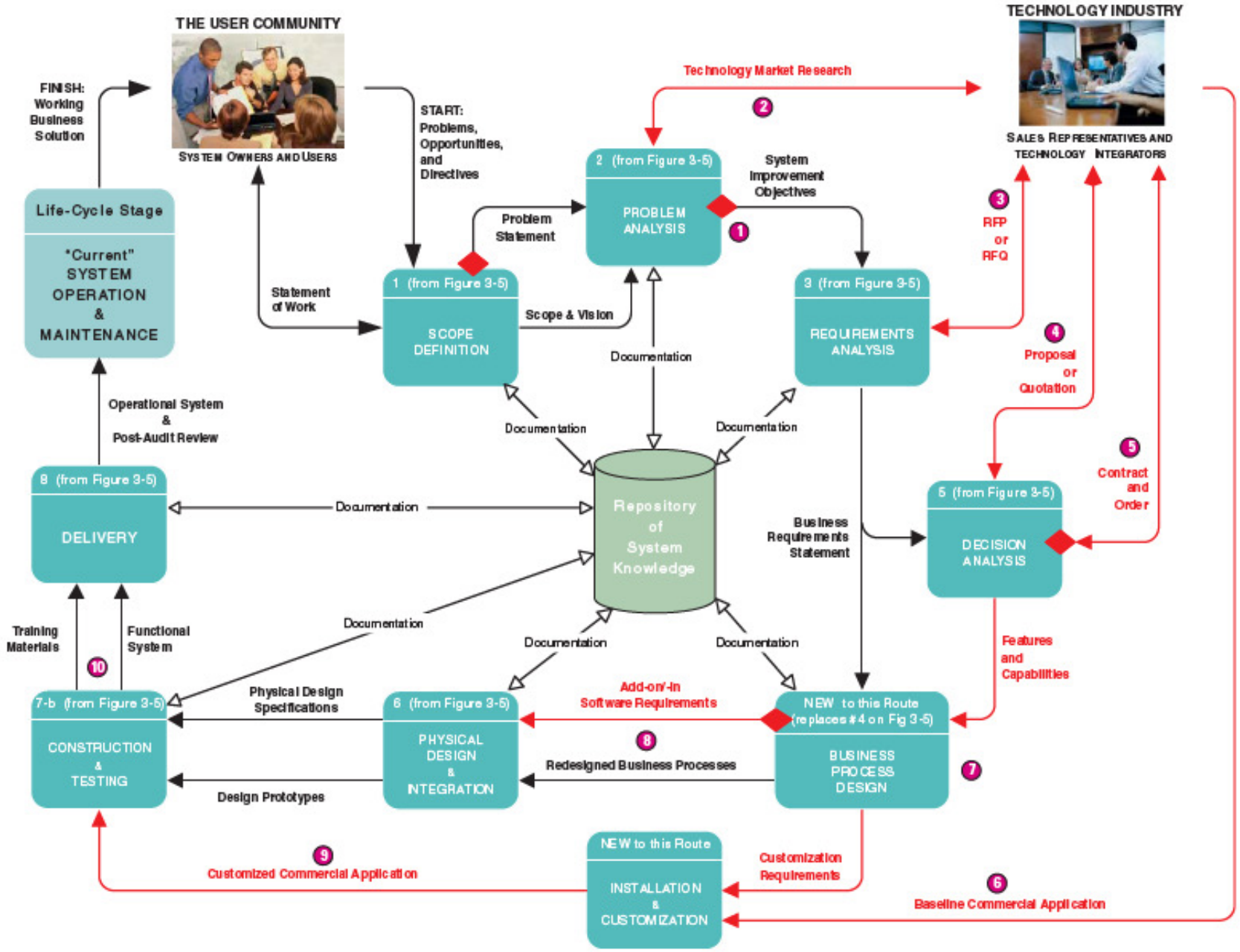


FIGURE 3-12 The Commercial Application Package Implementation Strategy

but they are usually necessary. In many cases, the necessary changes are not wrong—they are just different and unfamiliar. System users tend to be uncomfortable with changing the way they have always done something.

8. An application package rarely meets all business requirements upon installation. Typically, a **gap analysis** must be performed to determine which business requirements are not fulfilled by the package's capabilities and features. For requirements that will not be fulfilled, the following options exist:

   - Request customizations of the package within allowable limits as specified by the software vendor. Most commercial application packages allow the purchaser to set specific options, preferences, and defined values and ranges for certain parameters. Within limits, these customizations allow you to "personalize" the package to the business accounting and business practices. Such necessary CUSTOMIZATION REQUIREMENTS need to be specified.
   - Define ADD-ON SOFTWARE REQUIREMENTS. Add-on software requirements specify programs that must be designed and constructed to augment the commercial application package and deliver additional functionality. It should be noted that add-on programs carry some risk that they may have to be modified in the future when a new version of the software becomes available. But this risk is nominal, and most organizations take the risk in order to provide additional functionality.
   - Define ADD-IN SOFTWARE REQUIREMENTS. Add-in software requirements are very dangerous! They specify changes to the actual commercial application package to meet business requirements. In other words, users are requesting that changes be made to the purchased software, its database, or its user interfaces. At best, such changes can make version upgrades extremely difficult and prohibitively expensive. At worst, such changes can invalidate technical support from the vendor. (And most vendors encourage keeping versions relatively current by canceling technical support on older versions.) Changing program code and database structures should be discouraged. Insistence by users is often a symptom of unwillingness to adapt business processes to work with the application. Many organizations prohibit changes to application packages and force users to adapt in order to preserve their upgrade path.

9. The BASELINE COMMERCIAL APPLICATION is installed and tested. Allowable changes based on options, preferences, and parameters are completed and tested. Note: These customizations within the limits specified by the software vendor will typically carry forward to version upgrades. In most instances the vendor has provided for this level of CUSTOMIZED COMMERCIAL APPLICATION.
10. Any add-on (or add-in) software changes are designed and constructed to meet additional business requirements. The system is subsequently tested and placed into operation using the same activities described in the basic *FAST* process.

gap analysis a comparison of business and technical requirements for a commercial application package against the capabilities and features of a specific commercial application package for the purpose of defining the requirements that cannot be met.

The commercial application package strategy offers its own advantages and disadvantages:

Advantages

- New systems can usually be implemented more quickly because extensive programming is not required.
- Many businesses cannot afford the staffing and expertise required to develop in-house solutions.
- Application vendors spread their development costs across all customers that purchase their software. Thus, they can invest in continuous

Disadvantages

- A successful COTS implementation is dependent on the long-term success and viability of the application vendor—if the vendor goes out of business, you can lose your technical support and future improvements.
- A purchased system rarely reflects the ideal solution that the business might achieve with an in-house-developed system that could be

improvements in features, capabilities, and usability that individual businesses cannot always afford.
- The application vendor assumes responsibility for significant system improvements and error corrections.
- Many business functions are more similar than dissimilar for all businesses in a given industry. For example, business functions across organizations in the health care industry are more alike than different. It does not make good business sense for each organization to "reinvent the wheel."

customized to the precise expectations of management and the users.
- There is almost always at least some resistance to changing business processes to adapt to the software. Some users will have to give up or assume new responsibilities. And some people may resent changes they perceive to be technology-driven instead of business-driven.

Regardless, the trend toward purchased commercial application packages cannot be ignored. Today, many businesses require that a package alternative be considered prior to engaging in any type of in-house development project. Some experts estimate that by the year 2005 businesses will purchase 75 percent of their new information system applications. For this reason, we will teach systems analysis tools and techniques needed by system analysts to function in this environment.

> Hybrid Strategies

The *FAST* routes are not mutually exclusive. Any given project may elect to or be required to use a combination of, or variation of, more than one route. The route to be used is always selected during the *scope definition* phase and is negotiated as part of the *statement of work.* One strategy that is commonly applied to both model-driven and rapid application development routes is an incremental strategy. Figure 3-13 illustrates one possible implementation of an incremental strategy in combination with rapid application development. The project delivers the information system into operation in four stages. Each stage implements a version of the final system using a RAD route. Other variations on routes are possible.

> System Maintenance

All routes ultimately result in placing a new system into operation. System maintenance is intended to guide projects through the operation and support stage of their life cycle—which could last decades! Figure 3-14 places system maintenance into perspective. System maintenance in *FAST* is not really a unique route. As illustrated in the figure, it is merely a smaller-scale version of the same *FAST* process (or route) that was used to originally develop the system. The figure demonstrates that the starting point for system maintenance depends on the problem to be solved. We call your attention to the following numbered bullets in the figure:

1. Maintenance and reengineering projects are triggered by some combination of user and technical feedback. Such feedback may identify new problems, opportunities, or directives.
2. The maintenance project is initiated by a SYSTEM CHANGE REQUEST that indicates the problems, opportunities, or directives.
3. The simplest fixes are SOFTWARE BUGS (errors). Such a project typically jumps right into a RECONSTRUCTION AND RETESTING phase and is solved relatively quickly.
4. Sometimes a DESIGN FLAW in the system becomes apparent after implementation. For example, users may frequently make the same mistake due to a confusing screen design. For this type of maintenance project, the PHYSICAL DESIGN AND INTEGRATION phase would need to be revisited, followed of course by the construction and delivery phases.

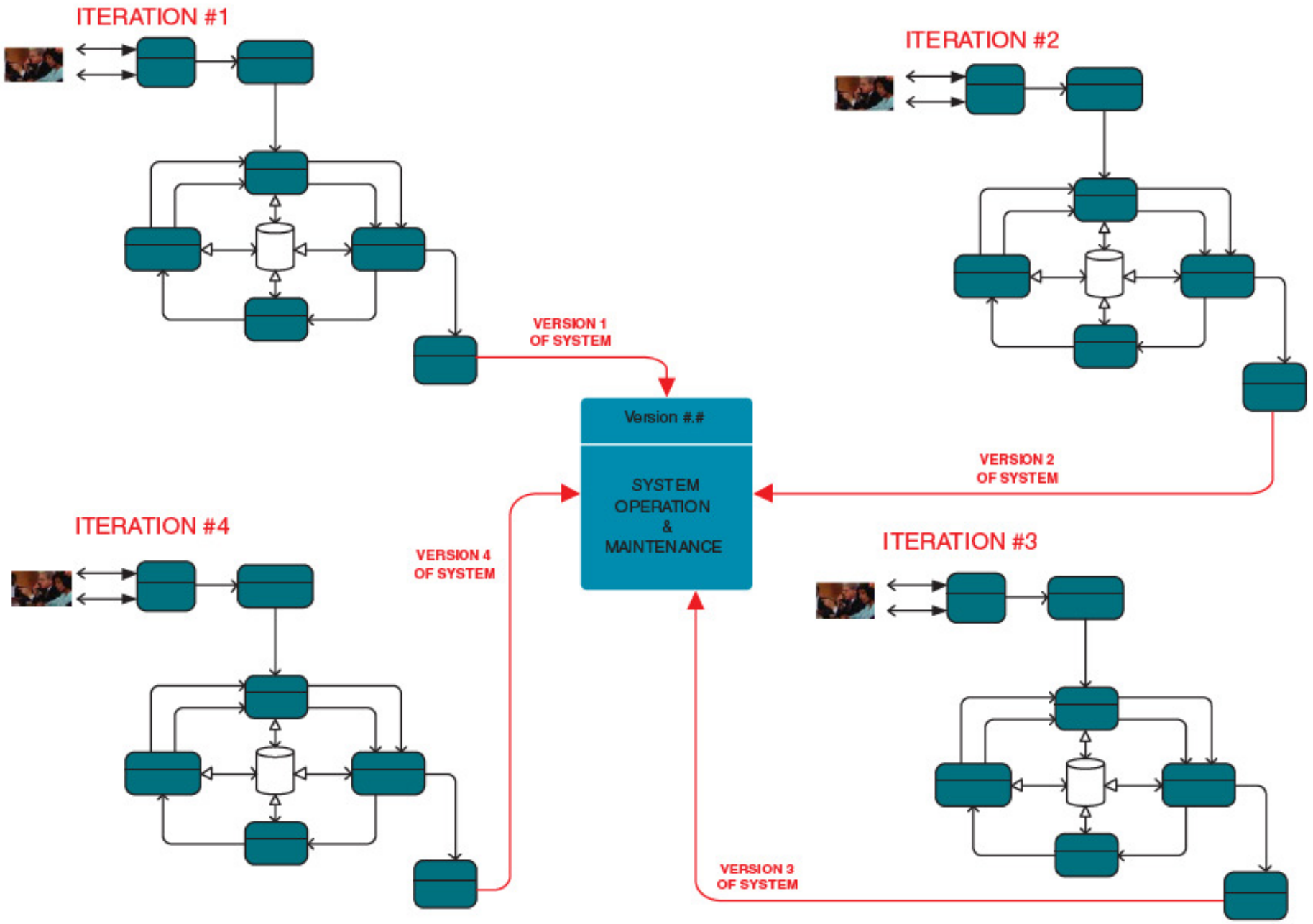


FIGURE 3-13 An Incremental Implementation Strategy

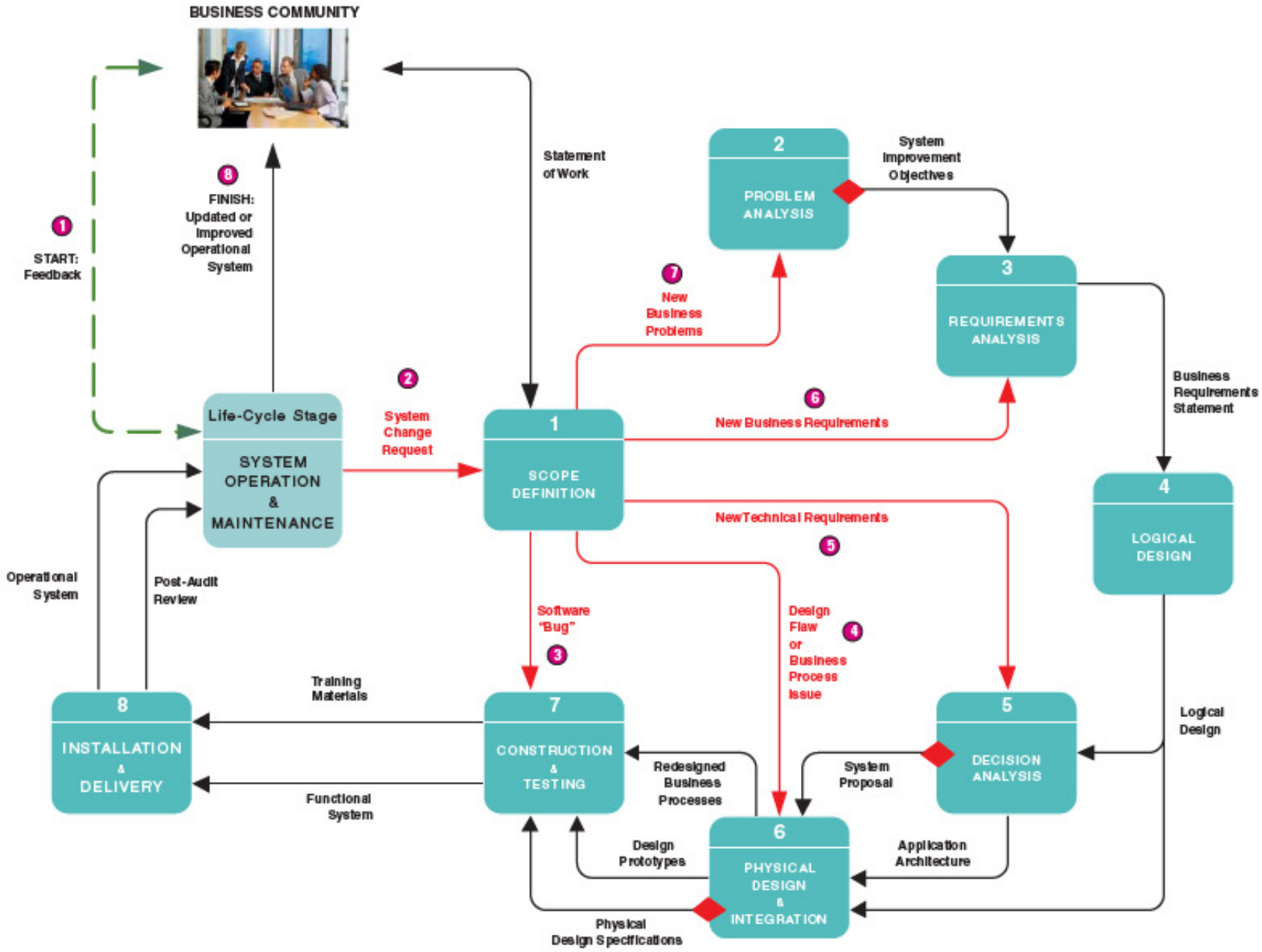


FIGURE 3-14 A System Maintenance Perspective

In some cases a BUSINESS PROCESS ISSUE may become apparent. In this case, only the business process needs to be redesigned.

5. On occasion, NEW TECHNICAL REQUIREMENTS might dictate a change. For example, an organization may standardize on the newest version of a particular database management system such as *SQL Server* or *Oracle*. For this type of project, the DECISION ANALYSIS phase may need to be revisited to first determine the risk and feasibility of converting the existing, operational database to the new version. As appropriate, such a project would subsequently proceed to the physical design, construction, and delivery phases as necessary.
6. Businesses constantly change; therefore, business requirements for a system also change. One of the most common triggers for a reengineering project is a NEW (or revised) BUSINESS REQUIREMENT. Given the requirement, the REQUIREMENTS ANALYSIS phase must be revisited with a focus on the impact of the new requirement on the existing system. Based on requirements analysis, the project would then proceed to the logical design, decision analysis, physical design, construction, and delivery phases.

   Time out! By now, you've noticed that maintenance and reengineering projects initiate in different phases of the basic methodology. You might be concerned that repeating these phases would require excessive time and effort. Keep in mind, however, that the scope of maintenance and reengineering projects is much, much smaller than the original project that created the operational system. Thus, the work required in each phase is much less than you might have guessed.
7. Again, as businesses change, significant NEW BUSINESS PROBLEMS, opportunities, and constraints can be encountered. In this type of project, work begins with the PROBLEM ANALYSIS phase and proceeds as necessary to the subsequent phases.
8. In all cases, the final result of any type of maintenance or reengineering project is an updated operational business system that delivers improved value to the system users and owners. Updates may include revised programs, databases, interfaces, or business processes.

As described earlier in the chapter, we expect all systems to eventually reach entropy. The business and/or technical problems and requirements have become so troublesome as to warrant "starting over."

That completes our introduction to the systems development methodology and routes. Before we end this chapter, we should introduce the role of automated tools that support systems development.

Automated Tools and Technology

You may be familiar with the old fable of the cobbler (shoemaker) whose own children had no shoes. That situation is not unlike the one faced by some systems developers. For years we've been applying information technology to solve our users' business problems; however, we've sometimes been slow to apply that same technology to our own problem—developing information systems. In the not-too-distant past, the principal tools of the systems analyst were paper, pencil, and flowchart template.

Today, entire suites of automated tools have been developed, marketed, and installed to assist systems developers. While system development methodologies do not always require automated tools, most methodologies do benefit from such technology. Some of the most commonly cited benefits include:

- Improved productivity—through automation of tasks.
- Improved quality—because automated tools check for completeness, consistency, and contradictions.
- Better and more consistent documentation—because the tools make it easier to create and assemble consistent, high-quality documentation.

- Reduced lifetime maintenance—because of the aforementioned system quality improvements combined with better documentation.
- Methodologies that really work—through rule enforcement and built-in expertise.

Chances are that your future employer is using or will be using this technology to develop systems. We will demonstrate the use of various automated tools throughout this textbook. There are three classes of automated tools for developers:

- Computer-aided systems modeling.
- Application development environments.
- Project and process managers.

Let's briefly examine each of these classes of automated tools.

REPRESENTATIVE CASE TOOLS

Computer Associates' *Erwin*
Oracle's *Designer 2000*
Popkin's *System Architect*
Rational ROSE
Visible Systems' *Visible Analyst*

> Computer-Assisted Systems Engineering

Systems developers have long aspired to transform information systems and software development into engineering-like disciplines. The terms *systems engineering* and *software engineering* are based on a vision that systems and software development can and should be performed with engineering-like precision and rigor. Such precision and rigor are consistent with the model-driven approach to systems development. To help systems analysts better perform system modeling, the industry developed automated tools called **computer-assisted software engineering (CASE)** tools. Think of CASE technology as software that is used to design and implement other software. This is very similar to the computer-aided design (CAD) technology used by most contemporary engineers to design products such as vehicles, structures, machines, and so forth. Representative modeling products are listed in the margin. Most modeling products run on personal computers, as depicted in Figure 3-15.

computer-assisted software engineering (CASE) the use of automated software tools that support the drawing and analysis of system models and associated specifications. Some CASE tools also provide prototyping and code generation capabilities.

CASE Repositories At the center of any true CASE tool's architecture is a developer's database called a **CASE repository.** The repository concept was introduced earlier (see Figure 3-7).

Around the CASE repository is a collection of tools, or facilities, for creating system models and documentation.

CASE repository a system developers' database where developers can store system models, detailed descriptions and specifications, and other products of systems development. Synonyms data include data *dictionary* and *encyclopedia*.

CASE Facilities To use the repository, the CASE tools provide some combination of the following facilities, illustrated in Figure 3-16:

- *Diagramming tools* are used to draw the system models required or recommended in most system development methodologies. Usually, the shapes on one system model can be linked to other system models and to detailed descriptions (see next item below).
- *Dictionary tools* are used to record, delete, edit, and output detailed documentation and specifications. The descriptions can be associated with shapes appearing on system models that were drawn with the diagramming tools.
- *Design tools* can be used to develop mock-ups of system components such as inputs and outputs. These inputs and outputs can be associated with both the aforementioned system models and the descriptions.
- *Quality management tools* analyze system models, descriptions and specifications, and designs for completeness, consistency, and conformance to accepted rules of the methodologies.
- *Documentation tools* are used to assemble, organize, and report on system models, descriptions and specifications, and prototypes that can be reviewed by system owners, users, designers, and builders.
- *Design and code generator tools* automatically generate database designs and application programs or significant portions of those programs.
- *Testing tools* simulate transactions and data traffic, measure performance, and provide configuration management of test plans and test scripts.

forward engineering a CASE tool capability that can generate initial software or database code directly from system models.

Forward and Reverse Engineering As previous stated, CASE technology automates system modeling. Today's CASE tools provide two distinct ways to develop system models—**forward engineering** and **reverse engineering.** Think of reverse

reverse engineering a CASE tool capability that can automatically generate initial system models from software or database code.

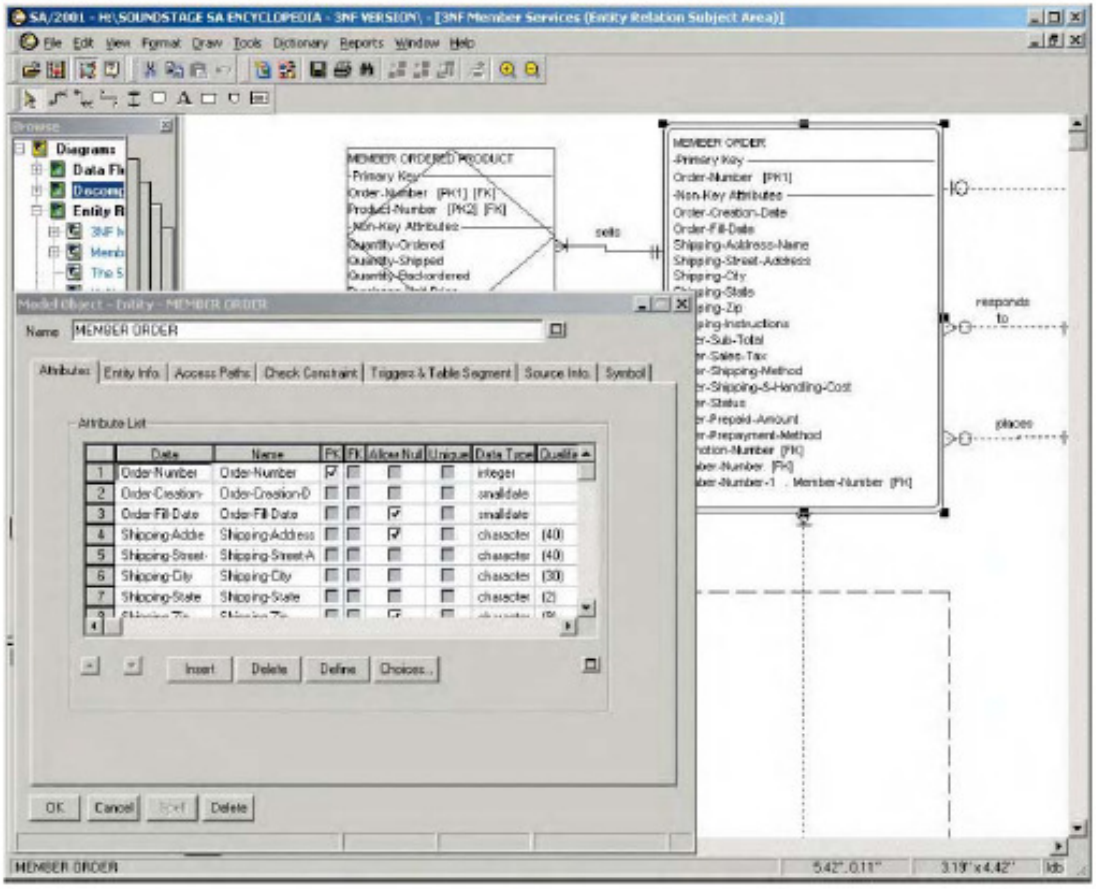


FIGURE 3-15 Screen Capture of *System Architect* CASE Tool

engineering as allowing you to generate a flowchart from an existing program and of forward engineering as allowing you to generate a program directly from a flowchart. CASE tools that allow for bidirectional, forward, and reverse engineering are said to provide for "round-trip engineering." For example, you reverse engineer a poorly designed system into a system model, edit and improve that model, and then forward engineer the new model into an improved system.

> Application Development Environments

The emphasis on speed and quality in software development has resulted in RAD approaches. The potential for RAD has been amplified by the transformation of programming language compilers into complete **application development environments (ADEs).** ADEs make programming simpler and more efficient. Indeed, most programming language compilers are now integrated into an ADE. Examples of ADEs (and the programming languages they support, where applicable) are listed in the margin.

Application development environments provide a number of productivity and quality management facilities. The ADE vendor provides some of these facilities. Third-party vendors provide many other facilities that can integrate into the ADE.

- *Programming languages* or *interpreters* are the heart of an ADE. Powerful debugging features and assistance are usually provided to help programmers quickly identify and solve programming problems.
- *Interface construction tools* help programmers quickly build the user interfaces using a component library.

REPRESENTATIVE ADEs

IBM's *Websphere (Java)*
Borland's *J Builder (Java)*
Macromedia's *Cold Fusion*
Microsoft's *Visual Studio.NET (VB.NET, C#, C++.NET)*
Oracle's *Developer*
Sybase's *Powerbuilder*

application development environment (ADE) an integrated software development tool that provides all the facilities necessary to develop new application software with maximum speed and quality. A common synonym is *integrated development environment (IDE).*

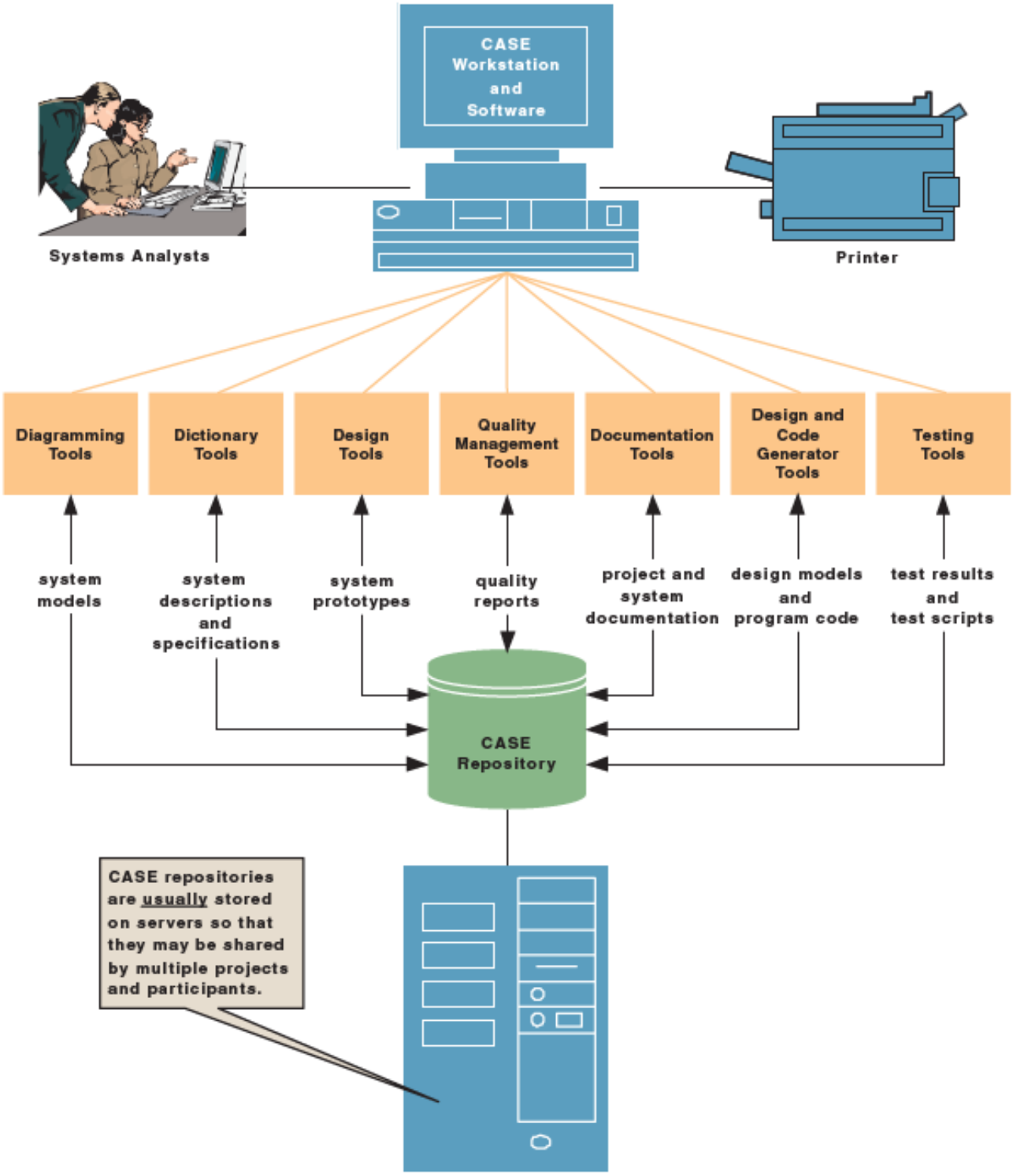


FIGURE 3-16 CASE Tool Architecture

- *Middleware* is software that helps programmers integrate the software being developed with various databases and computer networks.
- *Testing tools* are used to build and execute test scripts that can consistently and thoroughly test software.
- *Version control* tools help multiple programmer teams manage multiple versions of a program, both during development and after implementation.
- *Help authoring tools* are used to write online help systems, user manuals, and online training.
- *Repository links* permit the ADE to integrate with CASE tool products as well as other ADEs and development tools.

process manager application an automated tool that helps to document and manage a methodology and routes, its deliverables, and quality management standards. An emerging synonym is *methodware*.

> Process and Project Managers

A third class of automated tools helps us manage the system development methodology and projects that use the methodology. While CASE tools and ADEs are intended to support analysis, design, and construction of new information systems and software, **process manager application** and **project manager application** tools are intended to support cross life-cycle activities. Microsoft's *Project* and Niku's *Open Workbench* and *Project Manager* are examples of automated project management tools. Because process and project management is the subject of the next chapter, you'll learn more about these automated tools in that chapter.

project manager application an automated tool that helps to plan system development activities (preferably using the approved methodology), estimate and assign resources (including people and costs), schedule activities and resources, monitor progress against schedule and budget, control and modify schedule and resources, and report project progress.

Learning Roadmap

This chapter, along with the first two, completes the minimum context for studying systems analysis and design. We have described that context in terms of the three Ps—the participants (the stakeholders; Chapter 1), the product (the information system; Chapter 2), and the process (the system development; Chapter 3). Armed with this understanding, you are now ready to study systems analysis and/or design methods.

For many of you, Chapter 4, "Project Management," will provide a more complete context for studying systems analysis and design. It builds on Chapter 3 by emphasizing a variety of management issues related to the system development process. These include methodology management, resource management, management of expectations, change management, and others.

For those of you who skip the project and process management chapter, your next assignment will depend on whether your goal is:

- A comprehensive survey of systems development—We recommend you continue your sequential path to Chapter 5, "Systems Analysis." In that chapter you will study in greater depth the scope definition, problem analysis, requirements analysis, and logical design phases that were introduced in Chapter 3.
- The study of systems analysis techniques—Again, we recommend you continue your sequential path to Chapter 5, "Systems Analysis." Chapter 5 will provide a context for subsequently studying the tools and techniques of systems analysis.
- The study of systems design techniques—You might still want to quickly skim or review Chapter 5, "Systems Analysis," for context. Then continue your detailed study in Chapter 12, "System Design." In that chapter, you will learn more about the process and strategies for system design and construction of information systems.

Summary

1. A systems development process is a set of activities, methods, best practices, deliverables, and automated tools that stakeholders use to develop and continuously improve information systems and software.
2. The Capability Maturity Model (CMM) is a framework for assessing the maturity level of an organization's information systems development and management processes and products. It defines the need for a system development process.
3. A system life cycle divides the life of an information system into two stages, *systems development* and *systems operation and maintenance.*
4. A systems development methodology is a process for the system development stage. It defines a set of activities, methods, best practices, deliverables, and automated tools that systems developers and project managers are to use to develop and maintain information systems and software.
5. The following principles should underlie all systems development methodologies:
   a. Get the system users involved.
   b. Use a problem-solving approach.
   c. Establish phases and activities.
   d. Document throughout development.
   e. Establish standards.
   f. Manage the process and projects.
   g. Justify information systems as capital investments.
   h. Don't be afraid to cancel or revise scope.
   i. Divide and conquer.
   j. Design systems for growth and change.
6. System development projects are triggered by problems, opportunities, and directives:
   a. Problems are undesirable situations that prevent the organization from fully achieving its purpose, goals, and/or objectives.
   b. Opportunities are chances to improve the organization even in the absence of specific problems.
   c. Directives are new requirements that are imposed by management, government, or some external influence.
7. Wetherbe's PIECES framework is useful for categorizing problems, opportunities, and directives. The letters of the PIECES acronym correspond to Performance, Information, Economics, Control, Efficiency, and Service.
8. Traditional, basic systems development phases include:
   a. Scope definition
   b. Problem analysis
   c. Requirements analysis
   d. Logical design
   e. Decision analysis
   f. Physical design and integration
   g. Construction and testing
   h. Installation and delivery
9. Cross life-cycle activities are activities that overlap many or all phases of the methodology. They may include:
   a. Fact-finding, the formal process of using research, interviews, meetings, questionnaires, sampling, and other techniques to collect information about systems, requirements, and preferences.
   b. Documentation, the activity of recording facts and specifications for a system for current and future reference. Documentation is frequently stored in a repository, a database where systems developers store all documentation, knowledge, and products for one or more information systems or projects.
   c. Presentation, the activity of communicating findings, recommendations, and documentation for review by interested users and managers. Presentations may be either written or verbal.
   d. Feasibility analysis, the activity by which feasibility, a measure of how beneficial the development of an information system would be to an organization, is measured and assessed.
   e. Process management, the ongoing activity that documents, manages the use of, and improves an organization's chosen methodology (the "process") for systems development.
   f. Project management, the activity of defining, planning, directing, monitoring, and controlling a project to develop an acceptable system within the allotted time and budget.
10. There are different routes through the basic systems development phases. An appropriate route is selected during the scope definition phase. Typical routes include:
    a. Model-driven development strategies, which emphasize the drawing of diagrams to help visualize and analyze problems, define

business requirements, and design information systems. Alternative model-driven strategies include:

i) Process modeling
ii) Data modeling
iii) Object modeling

b. Rapid application development (RAD) strategies, which emphasize extensive user involvement in the rapid and evolutionary construction of working prototypes of a system to accelerate the system development process.
c. Commercial application package implementation strategies, which focus on the purchase and integration of a software package or solution to support one or more business functions and information systems.
d. System maintenance, which occurs after a system is implemented and lasts throughout the system's lifetime. Essentially, system maintenance executes a smaller-scale version of the development process with different starting points depending on the type of problem to be solved.

11. Automated tools support all systems development phases:

a. Computer-aided systems engineering (CASE) tools are software programs that automate or support the drawing and analysis of system models and provide for the translation of system models into application programs.
i) A CASE repository is a system developers' database. It is a place where developers can store system models, detailed descriptions and specifications, and other products of systems development.
ii) Forward engineering requires that the systems analyst draw system models, either from scratch or from templates. The resulting models are subsequently transformed into program code.
iii) Reverse engineering allows a CASE tool to read existing program code and transform that code into a representative system model that can be edited and refined by the systems analyst.

b. Application development environments (ADEs) are integrated software development tools that provide all the facilities necessary to develop new application software with maximum speed and quality.
c. Process management tools help us document and manage a methodology and routes, its deliverables, and quality management standards.
d. Project management tools help us plan system development activities (preferably using the approved methodology), estimate and assign resources (including people and costs), schedule activities and resources, monitor progress against schedule and budget, control and modify schedule and resources, and report project progress.

Review Questions

1. Explain why having a standardized system development process is important to an organization.
2. How are system life cycle and system development methodology related?
3. What are the 10 underlying principles for systems development?
4. Why is documentation important throughout the development process?
5. Why are process management and project management necessary?
6. What is risk management? Why is it necessary?
7. Which stakeholders initiate most projects? What is the impetus for most projects?
8. Who are the main participants in the scope definition? What are their goals in the scope definition?
9. What are the three most important deliverables in scope definition?
10. Who are the main participants in the requirements analysis phase? Why are they the main participants?
11. What feasibility analyses are made in the decision analysis?
12. What is model-driven development?
13. Why is model-driven development popular?
14. What is rapid application development (RAD)?
15. What benefits can RAD bring to the system development process?
16. What is computer-assisted software engineering (CASE)? List some examples of CASE.

Problems and Exercises

1. The Capability Maturity Model (CMM) was developed by the Software Engineering Institute at Carnegie Mellon, and is widely used by both the private and public sectors. What is the purpose of the CMM framework and how does it achieve this?
2. List the five maturity levels, and briefly describe each of them.
3. Table 3-1 in the textbook illustrates the difference in a typical project's duration, person-months, quality, and cost, depending upon whether an organization's system development process is at CMM level 1, 2, or 3. Between which two CMM levels does an organization gain the greatest benefit in terms of percentage of improvement? What do you think is the reason for this?
4. *Systems development methodology* and *system life cycle* are two terms that are frequently used and just as frequently misused. What is the difference between the two terms?
5. Describe how using a systems development methodology is in line with CMM goals and can help an organization increase its maturity level.
6. A number of underlying principles are common to all systems development methodologies. Identify these underlying principles and explain them.
7. The PIECES framework was developed by James Wetherbe as a means to classify problems. Identify the categories, then categorize the following problems using the PIECES framework:

   a. Duplicate data is stored throughout the system.
   b. There is a need to port an existing application to PDA devices.
   c. Quarterly sales reports need to be generated automatically.
   d. Employees can gain access to confidential portions of the personnel system.
   e. User interfaces for the inventory system are difficult and confusing, resulting in a high frequency of incorrect orders.

8. Each phase of a project includes specific deliverables that must be produced and delivered to the next phase. Using the textbook's hypothetical *FAST* methodology, what are the deliverables for the requirements analysis, logical design, and physical design/integration phases?
9. Scope definition is the first phase of the *FAST* methodology, and it is either the first phase or part of the first phase of most methodologies. What triggers the scope phase, which stakeholders are involved in this phase, what two essential questions need to be answered, and what three important deliverables come out of this phase?
10. The requirements analysis phase is an essential part of a system development methodology. According to the *FAST* methodology, which stakeholders typically participate in this phase? What is the primary focus of requirements analysis? What is *not* the focus? How should each proposed requirement be evaluated? What critical error must be avoided?
11. In the *FAST* methodology, as well as most system methodologies, system owners and system designers do not participate in the requirements analysis phase. What do you think the reason is for this?
12. What is the essential purpose of the logical design phase? How does it accomplish this? How are technological solutions incorporated in this phase? What are some common synonyms for this phase used by other methodologies? Who are the typical participants in this phase? What is agile modeling and what is its purpose? What are the deliverables coming out of this phase? In terms of the development team, what critical transition takes place by the end of this phase?
13. What is the essential purpose of the physical design phase? Who must be involved in this phase, and who may be involved? What are the two philosophies of physical design on the different ends of the continuum, and how are they different? Is this a likely phase in which a project might be canceled? With what other phase is there likely to be overlap, and what do you think is the reason for this?
14. A customer has engaged your software development company to develop a new order-processing system. However, the time frames are very tight and inflexible for delivery of at least the basic part of the new system. Further, user requirements are sketchy and unclear. What are two system development strategies that might be advantageous to use in this engagement?
15. What is the potential downside to using the strategies described in the preceding question?

Projects and Research

1. The Software Engineering Institute (SEI) at Carnegie Mellon University has developed a series of related Capability Maturity Models (CMMs). You can read about these different CMM products at SEI's Web site (http://www.sei.cmu.edu).

   a. Identify the current CMM products being maintained or developed by SEI.
   b. What are their differences and similarities?
   c. If you were to rate your organization, or an organization with which you are familiar, using the CMM described in the textbook, at which level would it be? Why?
   d. What steps would you recommend that your organization take in order to advance to the next CMM level?
   e. Do you feel that the time, cost, and resources to advance to the next level would be worth the perceived benefits for your organization? Why or why not?

2. You are a new project manager and have been assigned responsibility for an enterprise information systems project that touches every division in your organization. The chief executive officer stated at project initiation that successfully implementing this project was the number 1 priority of your organization. The project is in midst of the requirements analysis phase. While it is on schedule, you notice that attendance of the system users and owners at meetings on requirements has been dropping. A more experienced project manager has told you not to worry, that this is normal. Should you be concerned?
3. There are many different systems development methodologies in use, each with its own terminology, and number and scope of phases. Search the Web for information on two or three of these other systems development methodologies, then do the following:

   a. Note the systems development methodologies that you found. When, by whom, and why were they developed?
   b. What phases and terminology do they employ?
   c. Draw a matrix comparing their phases to the textbook's *FAST* methodology.
   d. What significant differences did you find?
   e. Do you see any advantages or disadvantages in any of the methodologies that you found compared to the *FAST* methodology?

4. The *PIECES* framework, which was developed by James Wetherbe and is described in the textbook, is intended to be a framework for classifying problems, opportunities, and directives.

   Contact one of the systems analysts for your organization, your school, and/or another organization. Ask them about the information systems used in their organization and to describe what the problems are in general terms. Select three of these systems:

   a. Describe the systems you selected, their problems, and the organizations that use them.
   b. Use the *PIECES* framework in the textbook to categorize each system's problems.
   c. Describe the *PIECES* category or categories you found for each problem. Did each problem generally have one or more categories associated with it?
   d. For the systems used by different organizations, what commonality did you find in the categories of problems? If you found a great deal of commonality of categories, do you think this is significant or just coincidental?
   e. Where in the systems development life cycle do you think the PIECES framework would be of the greatest value?

5. Computer-assisted software engineering (CASE) tools can significantly help developers improve productivity, quality, and documentation. Conduct an informal survey of about a dozen information technology departments regarding whether they use CASE tools, and if so, what type. Also, find out how long they have been using the CASE tools and whether they are used for all or just some projects. Add any other questions you may find meaningful. Try to split your survey between public and private sector agencies, and/or large and small organizations.

   a. What types of organizations did you survey?
   b. What did you ask?
   c. What were the results?
   d. Given the limited and informal nature of this survey, were you able to find any patterns or trends?
   e. Based upon your readings and your survey, what are your feelings regarding the use of CASE tools?

6. Projects at times are canceled or abandoned, sometime by choice, sometimes not. Research the Web for articles on project abandonment strategies, and select two of them.

a. What articles did you select?
b. What are their central themes, findings, and recommendations regarding project abandonment?
c. Compare and contrast their findings and their recommendations. Which strategy would you choose, if any?
d. Do you think that abandoning a project is always avoidable and/or always represents a failure?

Minicases

1. Interview at least two project managers. What are their experiences with *scope creep?*
2. George is the CEO of a major corporation that has been trying to develop a program that captures the keystrokes of employees on their computers. The project is currently $100,000 over budget and behind schedule and will require at least another $50,000 and six months to complete. The CEO wants to continue the project because so much has already been invested in it. What is your recommendation? Why?
3. Beatrice is an excellent manager—she is very capable of managing the bureaucratic process and following the business rules in her corporation. She is a "by the book" person who can always be counted on to do things "right." Will Beatrice make a good *project manager?* Why or why not?
4. A company is trying to decide between using an off-the-shelf program or developing a custom program for inventory management. The off-the-shelf product is less expensive than the custom solution and still has most of the needed functionality. The CEO believes that the missing capability can be addressed through tweaking the program once it is purchased. As the CIO of the company, what are your concerns and recommendations to the CEO?

Team and Individual Exercises

1. (Team) Hold team meetings using different communications mediums. Examples: phone, e-mail, virtual environment. What did you notice about the impact of the technology on the meeting? Was the productivity of the meetings the same for each medium? How did the team feel about the impact of the technologies on the team relationships?
2. (Individual) The analysis and design of an information system, done well, often eliminates jobs in a company. Economic theory strongly supports the creation of new jobs when this happens, but generally there is a time lag between the structural loss of jobs and the creation of new jobs. How do you feel about that?
3. (Team or Individual) Visit a neonatal intensive care unit at a highly regarded medical center (such as UC San Francisco). Write a short paper on your thoughts of the involved technology and the impact it has on people's lives. At the professor's discretion, share with the class.

Suggested Readings

Ambler, Scott. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process.* New York: John Wiley & Sons, 2002. This is the definitive book on agile methods and modeling.

Application Development Trends (monthly periodical). Framingham, MA: Software Productivity Group, Inc. This is our favorite periodical for keeping up with the latest trends in methodology and automated tools. Each month features several articles on different topics and products.

DeMarco, Tom. *Structured Analysis and System Specification.* Englewood Cliffs, NJ: Prentice Hall, 1978. This is the classic book on structured systems analysis, a process-centered, model-driven methodology.

Gane, Chris. *Rapid Systems Development.* Englewood Cliffs, NJ: Prentice Hall, 1989. This book presents a nice overview of RAD that combines model-driven development and prototyping in the correct balance.

Gildersleeve, Thomas. *Successful Data Processing Systems Analysis,* 2nd ed. Englewood Cliffs, NJ: Prentice Hall,

1985. We are indebted to Gildersleeve for the creeping commitment approach. The classics never become obsolete.

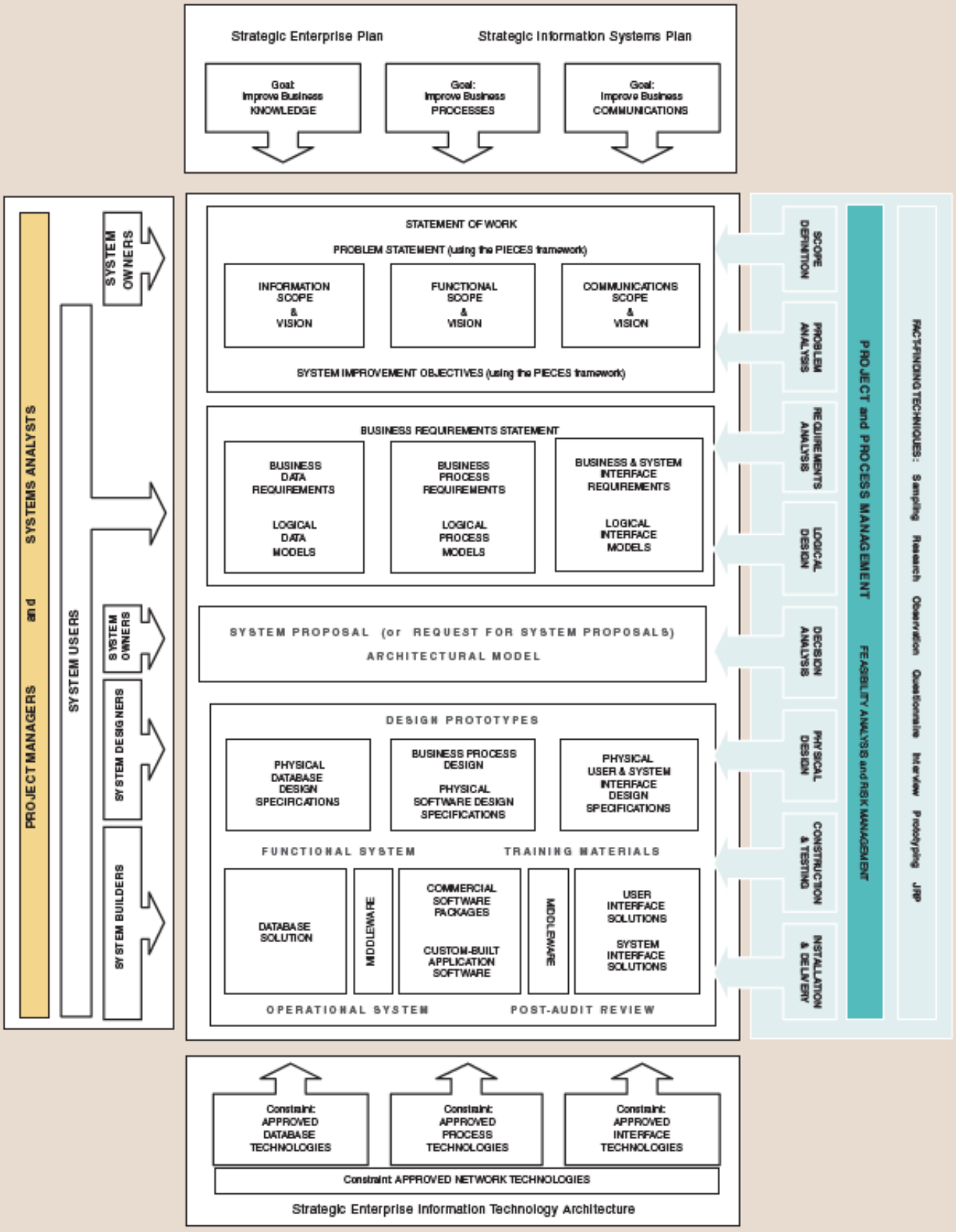
Jacobson, Ivar; Grady Booch; and James Rumbaugh. *The Unified Software Development Process.* Reading, MA: Addison-Wesley, 1999. The Rational Unified Process is a currently popular example of a model-driven, object-oriented methodology.

McConnell, Steve. *Rapid Development.* Redmond, WA: Microsoft Press, 1996. Chapter 7 of this excellent reference book provides what may be the definitive summary of system development life cycle and methodology variations that we call "routes" in our book.

Orr, Ken. *The One Minute Methodology.* New York: Dorsett House, 1990. Must reading for those interested in exploring the need for methodology. This very short book can be read in one sitting. It follows the satirical story of an analyst's quest for the development silver bullet, "the one minute methodology."

Paulk, Mark C.; Charles V. Weber; Bill Curtis; and Mary Beth Chrissis. *The Capability Maturity Model: Guidelines for Improving the Software Process.* Reading, MA: Addison-Wesley, 1995. This book fully describes version 1.1 of the Capability Maturity Model. Note that version 2.0 was under development at the time we were writing this chapter.

Wetherbe, James. *Systems Analysis and Design: Best Practices,* 4th ed. St. Paul, MN: West, 1994. We are indebted to Wetherbe for the PIECES framework.


Strategic Enterprise Plan
Strategic Information Systems Plan
Goal: Improve Business KNOWLEDGE
Goal: Improve Business PROCESSES
Goal: Improve Business COMMUNICATIONS
PROJECT MANAGERS and SYSTEMS ANALYSTS
SYSTEM USERS
SYSTEM OWNERS
SYSTEM OWNERS
SYSTEM DESIGNERS
SYSTEM BUILDERS
STATEMENT OF WORK
PROBLEM STATEMENT (using the PIECES framework)
INFORMATION SCOPE & VISION
FUNCTIONAL SCOPE & VISION
COMMUNICATIONS SCOPE & VISION
SYSTEM IMPROVEMENT OBJECTIVES (using the PIECES framework)
BUSINESS REQUIREMENTS STATEMENT
BUSINESS DATA REQUIREMENTS
LOGICAL DATA MODELS
BUSINESS PROCESS REQUIREMENTS
LOGICAL PROCESS MODELS
BUSINESS & SYSTEM INTERFACE REQUIREMENTS
LOGICAL INTERFACE MODELS
SYSTEM PROPOSAL (or REQUEST FOR SYSTEM PROPOSALS)
ARCHITECTURAL MODEL
DESIGN PROTOTYPES
PHYSICAL DATABASE DESIGN SPECIFICATIONS
BUSINESS PROCESS DESIGN
PHYSICAL SOFTWARE DESIGN SPECIFICATIONS
PHYSICAL USER & SYSTEM INTERFACE DESIGN SPECIFICATIONS
FUNCTIONAL SYSTEM
TRAINING MATERIALS
DATABASE SOLUTION
MIDDLEWARE
COMMERCIAL SOFTWARE PACKAGES
CUSTOM-BUILT APPLICATION SOFTWARE
MIDDLEWARE
USER INTERFACE SOLUTIONS
SYSTEM INTERFACE SOLUTIONS
OPERATIONAL SYSTEM
POST-AUDIT REVIEW
SCOPE DEFINITION
PROBLEM ANALYSIS
REQUIREMENTS ANALYSIS
LOGICAL DESIGN
DECISION ANALYSIS
PHYSICAL DESIGN
CONSTRUCTION & TESTING
INSTALLATION & DELIVERY
PROJECT and PROCESS MANAGEMENT
FEASIBILITY ANALYSIS and RISK MANAGEMENT
FACT-FINDING TECHNIQUES: Sampling Research Observation Questionnaire Interview Prototyping JRP
Constraint: APPROVED DATABASE TECHNOLOGIES
Constraint: APPROVED PROCESS TECHNOLOGIES
Constraint: APPROVED INTERFACE TECHNOLOGIES
Constraint: APPROVED NETWORK TECHNOLOGIES
Strategic Enterprise Information Technology Architecture