PART

Four

MANAGING SOFTWARE PROJECTS

In this part of *Software Engineering: A Practitioner's Approach* you'll learn the management techniques required to plan, organize, monitor, and control software projects. These questions are addressed in the chapters that follow:

- How must people, process, and problem be managed during a software project?
- How can software metrics be used to manage a software project and the software process?
- How does a software team generate reliable estimates of effort, cost, and project duration?
- What techniques can be used to assess the risks that can have an impact on project success?
- How does a software project manager select a set of software engineering work tasks?
- How is a project schedule created?
- Why are maintenance and reengineering so important for both software engineering managers and practitioners?

Once these questions are answered, you'll be better prepared to manage software projects in a way that will lead to timely delivery of a high-quality product.

CHAPTER

24 PROJECT MANAGEMENT CONCEPTS

KEY CONCEPTS

In the preface to his book on software project management, Meiler Page-Jones [Pag85] makes a statement that can be echoed by many software engineering consultants:

> I've visited dozens of commercial shops, both good and bad, and I've observed scores of data processing managers, again, both good and bad. Too often, I've watched in horror as these managers futilely struggled through nightmarish projects, squirmed under impossible deadlines, or delivered systems that outraged their users and went on to devour huge chunks of maintenance time.

What Page-Jones describes are symptoms that result from an array of management and technical problems. However, if a post mortem were to be conducted

QUICK LOOK

What is it? Although many of us (in our darker moments) take Dilbert's view of "management," it remains a very necessary activity when computer-based systems and products are built. Project management involves the planning, monitoring, and control of the people, process, and events that occur as software evolves from a preliminary concept to full operational deployment.

Who does it? Everyone "manages" to some extent, but the scope of management activities varies among people involved in a software project. A software engineer manages her day-to-day activities, planning, monitoring, and controlling technical tasks. Project managers plan, monitor, and control the work of a team of software engineers. Senior managers coordinate the interface between the business and software professionals.

Why is it important? Building computer software is a complex undertaking, particularly if it involves many people working over a relatively long time. That's why software projects need to be managed.

What are the steps? Understand the four P's—people, product, process, and project. People must be organized to perform software work effectively. Communication with the customer and other stakeholders must occur so that product scope and requirements are understood. A process that is appropriate for the people and the product should be selected. The project must be planned by estimating effort and calendar time to accomplish work tasks: defining work products, establishing quality checkpoints, and identifying mechanisms to monitor and control work defined by the plan.

What is the work product? A project plan is produced as management activities commence. The plan defines the process and tasks to be conducted, the people who will do the work, and the mechanisms for assessing risks, controlling change, and evaluating quality.

How do I ensure that I've done it right? You're never completely sure that the project plan is right until you've delivered a high-quality product on time and within budget. However, a project manager does it right when he encourages software people to work together as an effective team, focusing their attention on customer needs and product quality.

for every project, it is very likely that a consistent theme would be encountered: project management was weak.

In this chapter and Chapters 25 through 29, I'll present the key concepts that lead to effective software project management. This chapter considers basic software project management concepts and principles. Chapter 25 presents process and project metrics, the basis for effective management decision making. The techniques that are used to estimate cost are discussed in Chapter 26. Chapter 27 will help you to define a realistic project schedule. The management activities that lead to effective risk monitoring, mitigation, and management are presented in Chapter 28. Finally, Chapter 29 considers maintenance and reengineering and discusses the management issues that you'll encounter when you must deal with legacy systems.

24.1 The Management Spectrum

Effective software project management focuses on the four P's: people, product, process, and project. The order is not arbitrary. The manager who forgets that software engineering work is an intensely human endeavor will never have success in project management. A manager who fails to encourage comprehensive stakeholder communication early in the evolution of a product risks building an elegant solution for the wrong problem. The manager who pays little attention to the process runs the risk of inserting competent technical methods and tools into a vacuum. The manager who embarks without a solid project plan jeopardizes the success of the project.

24.1.1 The People

The cultivation of motivated, highly skilled software people has been discussed since the 1960s. In fact, the "people factor" is so important that the Software Engineering Institute has developed a *People Capability Maturity Model* (People-CMM), in recognition of the fact that "every organization needs to continually improve its ability to attract, develop, motivate, organize, and retain the workforce needed to accomplish its strategic business objectives" [Cur01].

The people capability maturity model defines the following key practice areas for software people: staffing, communication and coordination, work environment, performance management, training, compensation, competency analysis and development, career development, workgroup development, team/culture development, and others. Organizations that achieve high levels of People-CMM maturity have a higher likelihood of implementing effective software project management practices.

The People-CMM is a companion to the *Software Capability Maturity Model–Integration* (Chapter 30) that guides organizations in the creation of a mature

software process. Issues associated with people management and structure for software projects are considered later in this chapter.

24.1.2 The Product

Before a project can be planned, product objectives and scope should be established, alternative solutions should be considered, and technical and management constraints should be identified. Without this information, it is impossible to define reasonable (and accurate) estimates of the cost, an effective assessment of risk, a realistic breakdown of project tasks, or a manageable project schedule that provides a meaningful indication of progress.

As a software developer, you and other stakeholders must meet to define product objectives and scope. In many cases, this activity begins as part of the system engineering or business process engineering and continues as the first step in software requirements engineering (Chapter 5). Objectives identify the overall goals for the product (from the stakeholders' points of view) without considering how these goals will be achieved. Scope identifies the primary data, functions, and behaviors that characterize the product, and more important, attempts to bound these characteristics in a quantitative manner.

Once the product objectives and scope are understood, alternative solutions are considered. Although very little detail is discussed, the alternatives enable managers and practitioners to select a "best" approach, given the constraints imposed by delivery deadlines, budgetary restrictions, personnel availability, technical interfaces, and myriad other factors.

24.1.3 The Process

Those who adhere to the agile process philosophy (Chapter 3) argue that their process is leaner than others. That may be true, but they still have a process, and agile software engineering still requires discipline.

A software process (Chapters 2 and 3) provides the framework from which a comprehensive plan for software development can be established. A small number of framework activities are applicable to all software projects, regardless of their size or complexity. A number of different task sets—tasks, milestones, work products, and quality assurance points—enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team. Finally, umbrella activities—such as software quality assurance, software configuration management, and measurement—overlay the process model. Umbrella activities are independent of any one framework activity and occur throughout the process.

24.1.4 The Project

We conduct planned and controlled software projects for one primary reason—it is the only known way to manage complexity. And yet, software teams still struggle. In a study of 250 large software projects between 1998 and 2004, Capers Jones [Jon04] found that "about 25 were deemed successful in that they achieved their schedule, cost, and quality objectives. About 50 had delays or overruns below

35 percent, while about 175 experienced major delays and overruns, or were terminated without completion." Although the success rate for present-day software projects may have improved somewhat, our project failure rate remains much higher than it should be.[1]

To avoid project failure, a software project manager and the software engineers who build the product must avoid a set of common warning signs, understand the critical success factors that lead to good project management, and develop a commonsense approach for planning, monitoring, and controlling the project. Each of these issues is discussed in Section 24.5 and in the chapters that follow.

24.2 People

In a study published by the IEEE [Cur88], the engineering vice presidents of three major technology companies were asked what was the most important contributor to a successful software project. They answered in the following way:

> **VP 1:** I guess if you had to pick one thing out that is most important in our environment, I'd say it's not the tools that we use, it's the people.
>
> **VP 2:** The most important ingredient that was successful on this project was having smart people . . . very little else matters in my opinion. . . . The most important thing you do for a project is selecting the staff. . . . The success of the software development organization is very, very much associated with the ability to recruit good people.
>
> **VP 3:** The only rule I have in management is to ensure I have good people—real good people—and that I grow good people—and that I provide an environment in which good people can produce.

Indeed, this is a compelling testimonial on the importance of people in the software engineering process. And yet, all of us, from senior engineering vice presidents to the lowliest practitioner, often take people for granted. Managers argue (as the preceding group had) that people are primary, but their actions sometimes belie their words. In this section I examine the stakeholders who participate in the software process and the manner in which they are organized to perform effective software engineering.

24.2.1 The Stakeholders

The software process (and every software project) is populated by stakeholders who can be categorized into one of five constituencies:

1. *Senior managers* who define the business issues that often have a significant influence on the project.

1 Given these statistics, it's reasonable to ask how the impact of computers continues to grow exponentially. Part of the answer, I think, is that a substantial number of these "failed" projects are ill conceived in the first place. Customers lose interest quickly (because what they've requested wasn't really as important as they first thought), and the projects are cancelled.

2. *Project (technical) managers* who must plan, motivate, organize, and control the practitioners who do software work.
3. *Practitioners* who deliver the technical skills that are necessary to engineer a product or application.
4. *Customers* who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
5. *End users* who interact with the software once it is released for production use.

Every software project is populated by people who fall within this taxonomy.[2] To be effective, the project team must be organized in a way that maximizes each person's skills and abilities. And that's the job of the team leader.

24.2.2 Team Leaders

Project management is a people-intensive activity, and for this reason, competent practitioners often make poor team leaders. They simply don't have the right mix of people skills. And yet, as Edgemon states: "Unfortunately and all too frequently it seems, individuals just fall into a project manager role and become accidental project managers" [Edg95].

In an excellent book of technical leadership, Jerry Weinberg [Wei86] suggests an MOI model of leadership:

? What do we look for when choosing someone to lead a software project?

> **Motivation.** The ability to encourage (by "push or pull") technical people to produce to their best ability.
>
> **Organization.** The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.
>
> **Ideas or innovation.** The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

Weinberg suggests that successful project leaders apply a problem-solving management style. That is, a software project manager should concentrate on understanding the problem to be solved, managing the flow of ideas, and at the same time, letting everyone on the team know (by words and, far more important, by actions) that quality counts and that it will not be compromised.

Quote:

"In simplest terms, a leader is one who knows where he wants to go, and gets up, and goes."

John Erskine

Another view [Edg95] of the characteristics that define an effective project manager emphasizes four key traits:

> **Problem solving.** An effective software project manager can diagnose the technical and organizational issues that are most relevant, systematically structure a solution or properly motivate other practitioners to develop the solution, apply lessons learned from past projects to new situations, and

2 When WebApps are developed, other nontechnical people may be involved in content creation.

remain flexible enough to change direction if initial attempts at problem solution are fruitless.

Managerial identity. A good project manager must take charge of the project. She must have the confidence to assume control when necessary and the assurance to allow good technical people to follow their instincts.

Achievement. A competent manager must reward initiative and accomplishment to optimize the productivity of a project team. She must demonstrate through her own actions that controlled risk taking will not be punished.

Influence and team building. An effective project manager must be able to "read" people; she must be able to understand verbal and nonverbal signals and react to the needs of the people sending these signals. The manager must remain under control in high-stress situations.

24.2.3 The Software Team

Quote:

"Not every group is a team, and not every team is effective."

Glenn Parker

There are almost as many human organizational structures for software development as there are organizations that develop software. For better or worse, organizational structure cannot be easily modified. Concern with the practical and political consequences of organizational change are not within the software project manager's scope of responsibility. However, the organization of the people directly involved in a new software project is within the project manager's purview.

The "best" team structure depends on the management style of your organization, the number of people who will populate the team and their skill levels, and the overall problem difficulty. Mantei [Man81] describes seven project factors that should be considered when planning the structure of software engineering teams:

? What factors should be considered when the structure of a software team is chosen?

- Difficulty of the problem to be solved
- "Size" of the resultant program(s) in lines of code or function points
- Time that the team will stay together (team lifetime)
- Degree to which the problem can be modularized
- Required quality and reliability of the system to be built
- Rigidity of the delivery date
- Degree of sociability (communication) required for the project

Constantine [Con93] suggests four "organizational paradigms" for software engineering teams:

? What options do we have when defining the structure of a software team?

1. A *closed paradigm* structures a team along a traditional hierarchy of authority. Such teams can work well when producing software that is quite similar to past efforts, but they will be less likely to be innovative when working within the closed paradigm.

2. A *random paradigm* structures a team loosely and depends on individual initiative of the team members. When innovation or technological breakthrough is required, teams following the random paradigm will excel. But such teams may struggle when "orderly performance" is required.
3. An *open paradigm* attempts to structure a team in a manner that achieves some of the controls associated with the closed paradigm but also much of the innovation that occurs when using the random paradigm. Work is performed collaboratively, with heavy communication and consensus-based decision making the trademarks of open paradigm teams. Open paradigm team structures are well suited to the solution of complex problems but may not perform as efficiently as other teams.
4. A *synchronous paradigm* relies on the natural compartmentalization of a problem and organizes team members to work on pieces of the problem with little active communication among themselves.

Quote:

"If you want to be incrementally better: Be competitive. If you want to be exponentially better: Be cooperative."

Author unknown

As an historical footnote, one of the earliest software team organizations was a closed paradigm structure originally called the *chief programmer team.* This structure was first proposed by Harlan Mills and described by Baker [Bak72]. The nucleus of the team was composed of a *senior engineer* (the chief programmer), who plans, coordinates, and reviews all technical activities of the team; *technical staff* (normally two to five people), who conduct analysis and development activities; and a *backup engineer,* who supports the senior engineer in his or her activities and can replace the senior engineer with minimum loss in project continuity. The chief programmer may be served by one or more specialists (e.g., telecommunications expert, database designer), support staff (e.g., technical writers, clerical personnel), and a software librarian.

As a counterpoint to the chief programmer team structure, Constantine's random paradigm [Con93] suggests a software team with creative independence whose approach to work might best be termed *innovative anarchy.* Although the free-spirited approach to software work has appeal, channeling creative energy into a high-performance team must be a central goal of a software engineering organization. To achieve a high-performance team:

- Team members must have trust in one another.
- The distribution of skills must be appropriate to the problem.
- Mavericks may have to be excluded from the team, if team cohesiveness is to be maintained.

Regardless of team organization, the objective for every project manager is to help create a team that exhibits cohesiveness. In their book, *Peopleware,* DeMarco and Lister [DeM98] discuss this issue:

> We tend to use the word team fairly loosely in the business world, calling any group of people assigned to work together a "team." But many of these groups just don't seem like

What is a "jelled" team?

> teams. They don't have a common definition of success or any identifiable team spirit. What is missing is a phenomenon that we call *jell.*
>
> A jelled team is a group of people so strongly knit that the whole is greater than the sum of the parts. . . .
>
> Once a team begins to jell, the probability of success goes way up. The team can become unstoppable, a juggernaut for success. . . . They don't need to be managed in the traditional way, and they certainly don't need to be motivated. They've got momentum.

DeMarco and Lister contend that members of jelled teams are significantly more productive and more motivated than average. They share a common goal, a common culture, and in many cases, a "sense of eliteness" that makes them unique.

Why is it that teams fail to jell?

But not all teams jell. In fact, many teams suffer from what Jackman [Jac98] calls "team toxicity." She defines five factors that "foster a potentially toxic team environment": (1) a frenzied work atmosphere, (2) high frustration that causes friction among team members, (3) a "fragmented or poorly coordinated" software process, (4) an unclear definition of roles on the software team, and (5) "continuous and repeated exposure to failure."

Quote:

"Do or do not. There is no try."

Yoda from *Star Wars*

To avoid a frenzied work environment, the project manager should be certain that the team has access to all information required to do the job and that major goals and objectives, once defined, should not be modified unless absolutely necessary. A software team can avoid frustration if it is given as much responsibility for decision making as possible. An inappropriate process (e.g., unnecessary or burdensome work tasks or poorly chosen work products) can be avoided by understanding the product to be built, the people doing the work, and by allowing the team to select the process model. The team itself should establish its own mechanisms for accountability (technical reviews[3] are an excellent way to accomplish this) and define a series of corrective approaches when a member of the team fails to perform. And finally, the key to avoiding an atmosphere of failure is to establish team-based techniques for feedback and problem solving.

In addition to the five toxins described by Jackman, a software team often struggles with the differing human traits of its members. Some team members are extroverts; others are introverts. Some people gather information intuitively, distilling broad concepts from disparate facts. Others process information linearly, collecting and organizing minute details from the data provided. Some team members are comfortable making decisions only when a logical, orderly argument is presented. Others are intuitive, willing to make a decision based on "feel." Some practitioners want a detailed schedule populated by organized tasks that enable them to achieve closure for some element of a project. Others prefer a more spontaneous environment in which open issues are okay. Some work hard to get things done long before a milestone date, thereby avoiding stress as the date approaches, while others are

3 Technical reviews are discussed in detail in Chapter 15.

energized by the rush to make a last-minute deadline. A detailed discussion of the psychology of these traits and the ways in which a skilled team leader can help people with opposing traits to work together is beyond the scope of this book.[4] However, it is important to note that recognition of human differences is the first step toward creating teams that jell.

24.2.4 Agile Teams

Over the past decade, agile software development (Chapter 3) has been suggested as an antidote to many of the problems that have plagued software project work. To review, the agile philosophy encourages customer satisfaction and early incremental delivery of software, small highly motivated project teams, informal methods, minimal software engineering work products, and overall development simplicity.

The small, highly motivated project team, also called an *agile team,* adopts many of the characteristics of successful software project teams discussed in the preceding section and avoids many of the toxins that create problems. However, the agile philosophy stresses individual (team member) competency coupled with group collaboration as critical success factors for the team. Cockburn and Highsmith [Coc01a] note this when they write:

> If the people on the project are good enough, they can use almost any process and accomplish their assignment. If they are not good enough, no process will repair their inadequacy—"people trump process" is one way to say this. However, lack of user and executive support can kill a project—"politics trump people." Inadequate support can keep even good people from accomplishing the job.

KEY POINT

An agile team is a self-organizing team that has autonomy to plan and make technical decisions.

To make effective use of the competencies of each team member and to foster effective collaboration through a software project, agile teams are *self-organizing.* A self-organizing team does not necessarily maintain a single team structure but instead uses elements of Constantine's random, open, and synchronous paradigms discussed in Section 24.2.3.

Many agile process models (e.g., Scrum) give the agile team significant autonomy to make the project management and technical decisions required to get the job done. Planning is kept to a minimum, and the team is allowed to select its own approach (e.g., process, methods, tools), constrained only by business requirements and organizational standards. As the project proceeds, the team self-organizes to focus individual competency in a way that is most beneficial to the project at a given point in time. To accomplish this, an agile team might conduct daily team meetings to coordinate and synchronize the work that must be accomplished for that day.

Based on information obtained during these meetings, the team adapts its approach in a way that accomplishes an increment of work. As each day passes, continual self-organization and collaboration move the team toward a completed software increment.

---

4 An excellent introduction to these issues as they relate to software project teams can be found in [Fer98].

Quote:

"Collective ownership is nothing more than an instantiation of the idea that products should be attributable to the [agile] team, not individuals who make up the team."

Jim Highsmith

24.2.5 Coordination and Communication Issues

There are many reasons that software projects get into trouble. The scale of many development efforts is large, leading to complexity, confusion, and significant difficulties in coordinating team members. Uncertainty is common, resulting in a continuing stream of changes that ratchets the project team. Interoperability has become a key characteristic of many systems. New software must communicate with existing software and conform to predefined constraints imposed by the system or product.

These characteristics of modern software—scale, uncertainty, and interoperability—are facts of life. To deal with them effectively, you must establish effective methods for coordinating the people who do the work. To accomplish this, mechanisms for formal and informal communication among team members and between multiple teams must be established. Formal communication is accomplished through "writing, structured meetings, and other relatively non-interactive and impersonal communication channels" [Kra95]. Informal communication is more personal. Members of a software team share ideas on an ad hoc basis, ask for help as problems arise, and interact with one another on a daily basis.

SAFEHOME

Team Structure

The scene: Doug Miller's office prior to the initiation of the *SafeHome* software project.

The players: Doug Miller (manager of the *SafeHome* software engineering team) and Vinod Raman, Jamie Lazar, and other members of the product software engineering team.

The conversation:

Doug: Have you guys had a chance to look over the preliminary info on *SafeHome* that marketing has prepared?

Vinod (nodding and looking at his teammates): Yes. But we have a bunch of questions.

Doug: Let's hold on that for a moment. I'd like to talk about how we're going to structure the team, who's responsible for what . . .

Jamie: I'm really into the agile philosophy, Doug. I think we should be a self-organizing team.

Vinod: I agree. Given the tight time line and some of the uncertainty, and that fact that we're all really competent [laughs], that seems like the right way to go.

Doug: That's okay with me, but you guys know the drill.

Jamie (smiling and talking as if she was reciting something): "We make tactical decisions, about who does what and when, but it's our responsibility to get product out the door on time.

Vinod: And with quality.

Doug: Exactly. But remember there are constraints. Marketing defines the software increments to be produced—in consultation with us, of course.

Jamie: And?

Doug: And, we're going to use UML as our modeling approach.

Vinod: But keep extraneous documentation to an absolute minimum.

Doug: Who is the liaison with me?

Jamie: We decided that Vinod will be the tech lead—he's got the most experience, so Vinod is your liaison, but feel free to talk to any of us.

Doug (laughing): Don't worry, I will.

24.3 The Product

A software project manager is confronted with a dilemma at the very beginning of a software project. Quantitative estimates and an organized plan are required, but solid information is unavailable. A detailed analysis of software requirements would provide necessary information for estimates, but analysis often takes weeks or even months to complete. Worse, requirements may be fluid, changing regularly as the project proceeds. Yet, a plan is needed "now!"

Like it or not, you must examine the product and the problem it is intended to solve at the very beginning of the project. At a minimum, the scope of the product must be established and bounded.

24.3.1 Software Scope

The first software project management activity is the determination of *software scope.* Scope is defined by answering the following questions:

If you can't bound a characteristic of the software you intend to build, list the characteristic as a project risk (Chapter 25).

Context. How does the software to be built fit into a larger system, product, or business context, and what constraints are imposed as a result of the context?

Information objectives. What customer-visible data objects are produced as output from the software? What data objects are required for input?

Function and performance. What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?

Software project scope must be unambiguous and understandable at the management and technical levels. A statement of software scope must be bounded. That is, quantitative data (e.g., number of simultaneous users, target environment, maximum allowable response time) are stated explicitly, constraints and/or limitations (e.g., product cost restricts memory size) are noted, and mitigating factors (e.g., desired algorithms are well understood and available in Java) are described.

24.3.2 Problem Decomposition

Problem decomposition, sometimes called *partitioning* or *problem elaboration,* is an activity that sits at the core of software requirements analysis (Chapters 6 and 7). During the scoping activity no attempt is made to fully decompose the problem. Rather, decomposition is applied in two major areas: (1) the functionality and content (information) that must be delivered and (2) the process that will be used to deliver it.

In order to develop a reasonable project plan, you must decompose the problem. This can be accomplished using a list of functions or with use cases.

Human beings tend to apply a divide-and-conquer strategy when they are confronted with a complex problem. Stated simply, a complex problem is partitioned into smaller problems that are more manageable. This is the strategy that applies as project planning begins. Software functions, described in the statement of scope, are evaluated and refined to provide more detail prior to the beginning of estimation (Chapter 26). Because both cost and schedule estimates are functionally oriented,

some degree of decomposition is often useful. Similarly, major content or data objects are decomposed into their constituent parts, providing a reasonable understanding of the information to be produced by the software.

As an example, consider a project that will build a new word-processing product. Among the unique features of the product are continuous voice as well as virtual keyboard input via a multitouch screen, extremely sophisticated "automatic copy edit" features, page layout capability, automatic indexing and table of contents, and others. The project manager must first establish a statement of scope that bounds these features (as well as other more mundane functions such as editing, file management, and document production). For example, will continuous voice input require that the product be "trained" by the user? Specifically, what capabilities will the copy edit feature provide? Just how sophisticated will the page layout capability be and will it encompass the capabilities implied by a multitouch screen?

As the statement of scope evolves, a first level of partitioning naturally occurs. The project team learns that the marketing department has talked with potential customers and found that the following functions should be part of automatic copy editing: (1) spell checking, (2) sentence grammar checking, (3) reference checking for large documents (e.g., Is a reference to a bibliography entry found in the list of entries in the bibliography?), (4) the implementation of a style sheet feature that imposed consistency across a document, and (5) section and chapter reference validation for large documents. Each of these features represents a subfunction to be implemented in software. Each can be further refined if the decomposition will make planning easier.

24.4 The Process

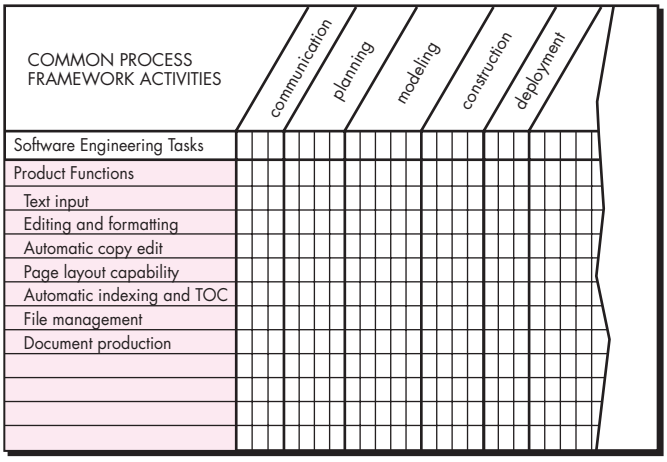
The framework activities (Chapter 2) that characterize the software process are applicable to all software projects. The problem is to select the process model that is appropriate for the software to be engineered by your project team.

Your team must decide which process model is most appropriate for (1) the customers who have requested the product and the people who will do the work, (2) the characteristics of the product itself, and (3) the project environment in which the software team works. When a process model has been selected, the team then defines a preliminary project plan based on the set of process framework activities. Once the preliminary plan is established, process decomposition begins. That is, a complete plan, reflecting the work tasks required to populate the framework activities must be created. We explore these activities briefly in the sections that follow and present a more detailed view in Chapter 26.

24.4.1 Melding the Product and the Process

Project planning begins with the melding of the product and the process. Each function to be engineered by your team must pass through the set of framework activities that have been defined for your software organization.

FIGURE 24.1
Melding the problem and the process



Assume that the organization has adopted the generic framework activities—**communication, planning, modeling, construction,** and **deployment**—discussed in Chapter 2. The team members who work on a product function will apply each of the framework activities to it. In essence, a matrix similar to the one shown in Figure 24.1 is created. Each major product function (the figure notes functions for the word-processing software discussed earlier) is listed in the left-hand column. Framework activities are listed in the top row. Software engineering work tasks (for each framework activity) would be entered in the following row.[5] The job of the project manager (and other team members) is to estimate resource requirements for each matrix cell, start and end dates for the tasks associated with each cell, and work products to be produced as a consequence of each task. These activities are considered in Chapter 26.

24.4.2 Process Decomposition

KEY POINT
The process framework establishes a skeleton for project planning. It is adapted by allocating a task set that is appropriate to the project.

A software team should have a significant degree of flexibility in choosing the software process model that is best for the project and the software engineering tasks that populate the process model once it is chosen. A relatively small project that is similar to past efforts might be best accomplished using the linear sequential approach. If the deadline is so tight that full functionality cannot reasonably be delivered, an incremental strategy might be best. Similarly, projects with other characteristics (e.g., uncertain requirements, breakthrough technology, difficult customers, significant reuse potential) will lead to the selection of other process models.[6]

5 It should be noted that work tasks must be adapted to the specific needs of the project based on a number of adaptation criteria.

6 Recall that project characteristics also have a strong bearing on the structure of the software team (Section 24.2.3).

Once the process model has been chosen, the process framework is adapted to it. In every case, the generic process framework discussed earlier can be used. It will work for linear models, for iterative and incremental models, for evolutionary models, and even for concurrent or component assembly models. The process framework is invariant and serves as the basis for all work performed by a software organization.

But actual work tasks do vary. Process decomposition commences when the project manager asks, "How do we accomplish this framework activity?" For example, a small, relatively simple project might require the following work tasks for the communication activity:

1. Develop list of clarification issues.
2. Meet with stakeholders to address clarification issues.
3. Jointly develop a statement of scope.
4. Review the statement of scope with all concerned.
5. Modify the statement of scope as required.

These events might occur over a period of less than 48 hours. They represent a process decomposition that is appropriate for the small, relatively simple project.

Now, consider a more complex project, which has a broader scope and more significant business impact. Such a project might require the following work tasks for the **communication**:

1. Review the customer request.
2. Plan and schedule a formal, facilitated meeting with all stakeholders.
3. Conduct research to specify the proposed solution and existing approaches.
4. Prepare a "working document" and an agenda for the formal meeting.
5. Conduct the meeting.
6. Jointly develop mini-specs that reflect data, functional, and behavioral features of the software. Alternatively, develop use cases that describe the software from the user's point of view.
7. Review each mini-spec or use case for correctness, consistency, and lack of ambiguity.
8. Assemble the mini-specs into a scoping document.
9. Review the scoping document or collection of use cases with all concerned.
10. Modify the scoping document or use cases as required.

Both projects perform the framework activity that we call **communication**, but the first project team performs half as many software engineering work tasks as the second.

24.5 The Project

In order to manage a successful software project, you have to understand what can go wrong so that problems can be avoided. In an excellent paper on software projects, John Reel [Ree99] defines 10 signs that indicate that an information systems project is in jeopardy:

? What are the signs that a software project is in jeopardy?

1. Software people don't understand their customer's needs.
2. The product scope is poorly defined.
3. Changes are managed poorly.
4. The chosen technology changes.
5. Business needs change [or are ill defined].
6. Deadlines are unrealistic.
7. Users are resistant.
8. Sponsorship is lost [or was never properly obtained].
9. The project team lacks people with appropriate skills.
10. Managers [and practitioners] avoid best practices and lessons learned.

Quote:

"We don't have time to stop for gas, we're already late."

M. Cleron

Jaded industry professionals often refer to the 90–90 rule when discussing particularly difficult software projects: The first 90 percent of a system absorbs 90 percent of the allotted effort and time. The last 10 percent takes another 90 percent of the allotted effort and time [Zah94]. The seeds that lead to the 90–90 rule are contained in the signs noted in the preceding list.

But enough negativity! How does a manager act to avoid the problems just noted? Reel [Ree99] suggests a five-part commonsense approach to software projects:

1. *Start on the right foot.* This is accomplished by working hard (very hard) to understand the problem that is to be solved and then setting realistic objectives and expectations for everyone who will be involved in the project. It is reinforced by building the right team (Section 24.2.3) and giving the team the autonomy, authority, and technology needed to do the job.
2. *Maintain momentum.* Many projects get off to a good start and then slowly disintegrate. To maintain momentum, the project manager must provide incentives to keep turnover of personnel to an absolute minimum, the team should emphasize quality in every task it performs, and senior management should do everything possible to stay out of the team's way.[7]

7 The implication of this statement is that bureaucracy is reduced to a minimum, extraneous meetings are eliminated, and dogmatic adherence to process and project rules is deemphasized. The team should be self-organizing and autonomous.

Quote:

"A project is like a road trip. Some projects are simple and routine, like driving to the store in broad daylight. But most projects worth doing are more like driving a truck off-road, in the mountains, at night."

Cem Kaner, James Bach, and Bret Pettichord

3. *Track progress.* For a software project, progress is tracked as work products (e.g., models, source code, sets of test cases) are produced and approved (using technical reviews) as part of a quality assurance activity. In addition, software process and project measures (Chapter 25) can be collected and used to assess progress against averages developed for the software development organization.
4. *Make smart decisions.* In essence, the decisions of the project manager and the software team should be to "keep it simple." Whenever possible, decide to use commercial off-the-shelf software or existing software components or patterns, decide to avoid custom interfaces when standard approaches are available, decide to identify and then avoid obvious risks, and decide to allocate more time than you think is needed to complex or risky tasks (you'll need every minute).
5. *Conduct a postmortem analysis.* Establish a consistent mechanism for extracting lessons learned for each project. Evaluate the planned and actual schedules, collect and analyze software project metrics, get feedback from team members and customers, and record findings in written form.

24.6 THE W^{5}HH PRINCIPLE

In an excellent paper on software process and projects, Barry Boehm [Boe96] states: "you need an organizing principle that scales down to provide simple [project] plans for simple projects." Boehm suggests an approach that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches, and required resources. He calls it the *W^{5}HH Principle,* after a series of questions that lead to a definition of key project characteristics and the resultant project plan:

? How do we define key project characteristics?

Why is the system being developed? All stakeholders should assess the validity of business reasons for the software work. Does the business purpose justify the expenditure of people, time, and money?

What will be done? The task set required for the project is defined.

When will it be done? The team establishes a project schedule by identifying when project tasks are to be conducted and when milestones are to be reached.

Who is responsible for a function? The role and responsibility of each member of the software team is defined.

Where are they located organizationally? Not all roles and responsibilities reside within software practitioners. The customer, users, and other stakeholders also have responsibilities.

How will the job be done technically and managerially? Once product scope is established, a management and technical strategy for the project must be defined.

How much of each resource is needed? The answer to this question is derived by developing estimates (Chapter 26) based on answers to earlier questions.

Boehm's W[5]HH Principle is applicable regardless of the size or complexity of a software project. The questions noted provide you and your team with an excellent planning outline.

24.7 Critical Practices

The Airlie Council[8] has developed a list of "critical software practices for performance-based management." These practices are "consistently used by, and considered critical by, highly successful software projects and organizations whose 'bottom line' performance is consistently much better than industry averages" [Air99].

Critical practices[9] include: metric-based project management (Chapter 25), empirical cost and schedule estimation (Chapters 26 and 27), earned value tracking (Chapter 27), defect tracking against quality targets (Chapters 14 though 16), and people aware management (Section 24.2). Each of these critical practices is addressed throughout Parts 3 and 4 of this book.

Software Tools

Software Tools for Project Managers

The "tools" listed here are generic and apply to a broad range of activities performed by project managers. Specific project management tools (e.g., scheduling tools, estimating tools, risk analysis tools) are considered in later chapters.

Representative Tools:[10]

The Software Program Manager's Network (**www.spmn.com**) has developed a simple tool called *Project Control Panel*, which provides project managers with an direct indication of project status. The tool has "gauges" much like a dashboard and is implemented with Microsoft Excel. It is available for download at **www.spmn.com/products_software.html**.

Ganthead.com (**www.ganttthead.com/**) has developed a set of useful *checklists for project managers*.

Ittoolkit.com (**www.ittoolkit.com**) provides "a collection of planning guides, process templates and smart worksheets" available on CD-ROM.

8 The Airlie Council was comprised of a team of software engineering experts chartered by the U.S. Department of Defense to help develop guidelines for best practices in software project management and software engineering. For more on best practices, see **www.swqual.com/newsletter/vol1/no3/vol1no3.html**.

9 Only those critical practices associated with "project integrity" are noted here.

10 Tools noted here do not represent an endorsement, but rather a sampling of tools in this category. In most cases, tool names are trademarked by their respective developers.

24.8 Summary

Software project management is an umbrella activity within software engineering. It begins before any technical activity is initiated and continues throughout the modeling, construction, and deployment of computer software.

Four P's have a substantial influence on software project management—people, product, process, and project. People must be organized into effective teams, motivated to do high-quality software work, and coordinated to achieve effective communication. Product requirements must be communicated from customer to developer, partitioned (decomposed) into their constituent parts, and positioned for work by the software team. The process must be adapted to the people and the product. A common process framework is selected, an appropriate software engineering paradigm is applied, and a set of work tasks is chosen to get the job done. Finally, the project must be organized in a manner that enables the software team to succeed.

The pivotal element in all software projects is people. Software engineers can be organized in a number of different team structures that range from traditional control hierarchies to "open paradigm" teams. A variety of coordination and communication techniques can be applied to support the work of the team. In general, technical reviews and informal person-to-person communication have the most value for practitioners.

The project management activity encompasses measurement and metrics, estimation and scheduling, risk analysis, tracking, and control. Each of these topics is considered in the chapters that follow.

Problems and Points to Ponder

24.1. Based on information contained in this chapter and your own experience, develop "ten commandments" for empowering software engineers. That is, make a list of 10 guidelines that will lead to software people who work to their full potential.

24.2. The Software Engineering Institute's People Capability Maturity Model (People-CMM) takes an organized look at "key practice areas" that cultivate good software people. Your instructor will assign you one KPA for analysis and summary.

24.3. Describe three real-life situations in which the customer and the end user are the same. Describe three situations in which they are different.

24.4. The decisions made by senior management can have a significant impact on the effectiveness of a software engineering team. Provide five examples to illustrate that this is true.

24.5. Review a copy of Weinberg's book [Wei86], and write a two- or three-page summary of the issues that should be considered in applying the MOI model.

24.6. You have been appointed a project manager within an information systems organization. Your job is to build an application that is quite similar to others your team has built, although this one is larger and more complex. Requirements have been thoroughly documented by the customer. What team structure would you choose and why? What software process model(s) would you choose and why?

24.7. You have been appointed a project manager for a small software products company. Your job is to build a breakthrough product that combines virtual reality hardware with state-of-the-art software. Because competition for the home entertainment market is intense, there is significant pressure to get the job done. What team structure would you choose and why? What software process model(s) would you choose and why?

24.8. You have been appointed a project manager for a major software products company. Your job is to manage the development of the next-generation version of its widely used word-processing software. Because competition is intense, tight deadlines have been established and announced. What team structure would you choose and why? What software process model(s) would you choose and why?

24.9. You have been appointed a software project manager for a company that services the genetic engineering world. Your job is to manage the development of a new software product that will accelerate the pace of gene typing. The work is R&D oriented, but the goal is to produce a product within the next year. What team structure would you choose and why? What software process model(s) would you choose and why?

24.10. You have been asked to develop a small application that analyzes each course offered by a university and reports the average grade obtained in the course (for a given term). Write a statement of scope that bounds this problem.

24.11. Do a first-level functional decomposition of the page layout function discussed briefly in Section 24.3.2.

Further Readings and Information Sources

The Project Management Institute (*Guide to the Project Management Body of Knowledge,* PMI, 2001) covers all important aspects of project management. Bechtold (*Essentials of Software Project Management,* 2d ed., Management Concepts, 2007), Wysocki (*Effective Software Project Management,* Wiley, 2006), Stellman and Greene (*Applied Software Project Management,* O'Reilly, 2005), and Berkun (*The Art of Project Management,* O'Reilly, 2005) teach basic skills and provide detailed guidance for all software project management tasks. McConnell (*Professional Software Development,* Addison-Wesley, 2004) offers pragmatic advice for achieving "shorter schedules, higher quality products, and more successful projects." Henry (*Software Project Management,* Addison-Wesley, 2003) offers real-world advice that is useful for all project managers.

Tom DeMarco and his colleagues (*Adrenaline Junkies and Template Zombies,* Dorset House, 2008) have written an insightful treatment of the human patterns that are encountered in every software project. An excellent four-volume series written by Weinberg (*Quality Software Management,* Dorset House, 1992, 1993, 1994, 1996) introduces basic systems thinking and management concepts, explains how to use measurements effectively, and addresses "congruent action," the ability to establish "fit" between the manager's needs, the needs of technical staff, and the needs of the business. It will provide both new and experienced managers with useful information. Futrell and his colleagues (*Quality Software Project Management,* Prentice-Hall, 2002) present a voluminous treatment of project management. Brown and his colleagues (*Antipatterns in Project Management,* Wiley, 2000) discuss what not to do during the management of a software project.

Brooks (*The Mythical Man-Month,* Anniversary Edition, Addison-Wesley, 1995) has updated his classic book to provide new insight into software project and management issues. McConnell (*Software Project Survival Guide,* Microsoft Press, 1997) presents excellent pragmatic guidance for those who must manage software projects. Purba and Shah (*How to Manage a Successful Software Project,* 2d ed., Wiley, 2000) present a number of case studies that indicate why some projects succeed and others fail. Bennatan (*On Time Within Budget,* 3d ed., Wiley, 2000) presents useful tips and guidelines for software project managers. Weigers (*Practical Project Initiation,* Microsoft Press, 2007) provides practical guidelines for getting a software project off the ground successfully.

It can be argued that the most important aspect of software project management is people management. Cockburn (*Agile Software Development,* Addison-Wesley, 2002) presents one of

the best discussions of software people written to date. DeMarco and Lister [DeM98] have written the definitive book on software people and software projects. In addition, the following books on this subject have been published in recent years and are worth examining:

Cantor, M., *Software Leadership: A Guide to Successful Software Development,* Addison-Wesley, 2001.

Carmel, E., *Global Software Teams: Collaborating Across Borders and Time Zones,* Prentice Hall, 1999.

Constantine, L., *Peopleware Papers: Notes on the Human Side of Software,* Prentice Hall, 2001.

Garton, C., and K. Wegryn, *Managing Without Walls,* McPress, 2006.

Humphrey, W. S., *Managing Technical People: Innovation, Teamwork, and the Software Process,* Addison-Wesley, 1997.

Humphrey, W. S., *TSP-Coaching Development Teams,* Addison-Wesley, 2006.

Jones, P. H., *Handbook of Team Design: A Practitioner's Guide to Team Systems Development,* McGraw-Hill, 1997.

Karolak, D. S., *Global Software Development: Managing Virtual Teams and Environments,* IEEE Computer Society, 1998.

Peters, L., *Getting Results from Software Development Teams,* Microsoft Press, 2008.

Whitehead, R., *Leading a Software Development Team,* Addison-Wesley, 2001.

Even though they do not relate specifically to the software world and sometimes suffer from oversimplification and broad generalization, best-selling "management" books by Kanter (*Confidence,* Three Rivers Press, 2006), Covy (*The 8th Habit,* Free Press, 2004), Bossidy (*Execution: The Discipline of Getting Things Done,* Crown Publishing, 2002), Drucker (*Management Challenges for the 21st Century,* Harper Business, 1999), Buckingham and Coffman (*First, Break All the Rules: What the World's Greatest Managers Do Differently,* Simon and Schuster, 1999), and Christensen (*The Innovator's Dilemma,* Harvard Business School Press, 1997) emphasize "new rules" defined by a rapidly changing economy. Older titles such as *Who Moved My Cheese?*, *The One-Minute Manager,* and *In Search of Excellence* continue to provide valuable insights that can help you to manage people and projects more effectively.

A wide variety of information sources on the software project management are available on the Internet. An up-to-date list of World Wide Web references relevant to software project management can be found at the SEPA website: **www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm**.