1

INTRODUCTION

> In many ways, managing a large computer programming project is like managing any other large undertaking—in more ways than most programmers believe. But in many other ways it is different—in more ways than most professional managers expect.[1]
>
> —Fred Brooks

1.1 INTRODUCTION TO SOFTWARE PROJECT MANAGEMENT

When you become (or perhaps already are) the manager of a software project you will find that experience to be one of the most challenging and most rewarding endeavors of your career. You, as a project manager, will be (or are) responsible for (1) delivering an acceptable product, (2) on the specified delivery date, and (3) within the constraints of the specified budget, resources, and technology. In return you will have, or should have, authority to use the resources available to you in the ways you think best to achieve the project objectives within the constraints of acceptable product, delivery date, and budget, resources, and technology.

Unfortunately, software projects have the (often deserved) reputation of costing more than estimated, taking longer than planned, and delivering less in quantity and quality of product than expected or required. Avoiding this stereotypical situation is the challenge of managing and leading software projects.

There are four fundamental activities that you must accomplish if you are to be a successful project manager:

[1] *The Mythical Man-Month, Anniversary Edition*, Frederick P. Brooks Jr., Addison Wesley, 1995; p. x.

Managing and Leading Software Projects, by Richard E. Fairley

1. planning and estimating,
2. measuring and controlling,
3. communicating, coordinating, and leading, and
4. managing risk.

These are the major themes of this text.

1.2 OBJECTIVES OF THIS CHAPTER

After reading this chapter and completing the exercises, you should understand:

- why managing and leading software projects is difficult,
- the nature of project constraints,
- a workflow model for software projects,
- the work products of software projects,
- the organizational context of software projects,
- organizing a software development team,
- maintaining the project vision and product goals, and
- the nature of process frameworks, software engineering standards, and process guidelines.

Appendix 1A to this chapter provides an introduction to elements of the following frameworks, standards, and guidelines that are concerned with managing software projects: the SEI Capability Maturity Model® Integration CMMI-DEV-v1.2, ISO/IEC and IEEE/EIA Standards 12207, IEEE/EIA Standard 1058, and the Project Management Body of Knowledge (PMBOK®). Terms used in this chapter and throughout this text are defined in a glossary at the end of the text. Presentation slides for this chapter and other supporting material are available at the URL listed in the Preface.

1.3 WHY MANAGING AND LEADING SOFTWARE PROJECTS IS DIFFICULT

A *project* is a group of coordinated activities conducted within a specific time frame for the purpose of achieving specified objectives. Some projects are personal in nature, for example, building a dog house or painting a bedroom. Other projects are conducted by organizations. The focus of this text is on projects conducted within software organizations. In a general sense, all organizational projects are similar:

- objectives must be specified,
- a schedule of activities must be planned,
- resources allocated,
- responsibilities assigned,

- work activities coordinated,
- progress monitored,
- communication maintained,
- risk factors identified and confronted, and
- corrective actions applied as necessary.

In a specific sense, the methods, tools, and techniques used to manage a project depend on the nature of the work to be accomplished and the work products to be produced. Manufacturing projects are different from construction projects, which are different from agricultural projects, which are different from computer hardware projects, which are different from software engineering projects, and so on. Each kind of project, including software projects, adapts and tailors the general procedures of project management to accommodate the unique aspects of the development processes and the nature of the product to be developed.

Fred Brooks has famously observed that four essential properties of software differentiate it from other kinds of engineering artifacts and make software projects difficult[2]:

1. complexity,
2. conformity,
3. changeability, and
4. invisibility of software.

1.3.1 Software Complexity

Software is more complex, for the effort and the expense required to construct it, than most artifacts produced by human endeavor. Assuming it costs $50 (USD) per line of code to construct a one-million line program (specify, design, implement, verify, validate, and deliver it), the resulting cost will be $50,000,000. While this is a large sum of money, it is a small fraction of the cost of constructing a complex spacecraft, a skyscraper, or a naval aircraft carrier.

Brooks says, "Software entities are more complex for their *size* [emphasis added] than perhaps any other human construct, because no two parts are alike (at least above the statement level)." [3] It is difficult to visualize the size of a software program because software has no physical attributes; however, if one were to print a one-million line program the stack of paper would be about 10 feet (roughly 3 meters) high if the program were printed 50 lines per page. The printout would occupy a volume of about 6.5 cubic feet. Biological entities such as human beings are of similar volume and they are far more complex than computer software, but there are few, if any, human-made artifacts of comparable size that are as complex as software.

The complexity of software arises from the large number of unique, interacting parts in a software system. The parts are unique because, for the most part, they are encapsulated as functions, subroutines, or objects and invoked as needed rather

[2] *Ibid*, pp. 182–186.
[3] *Ibid*, p. 182.

than being replicated. Software parts have several different kinds of interactions, including serial and concurrent invocations, state transitions, data couplings, and interfaces to databases and external systems. Depiction of a software entity often requires several different representations to portray the numerous static structures, dynamic couplings, and modes of interaction that exist in computer software.

A seemingly "small" change in requirements is one of the many ways that complexity of the product may affect management of a project. Complexity within the parts and in the connections among parts may result in a large amount of evolutionary rework for the "small" change in requirements, thus upsetting the ability to make progress according to plan. For this reason many experienced project managers say there are no small requirements changes. Size and complexity can also hide defects that may not be discovered immediately and thus require additional, unplanned corrective rework later.

1.3.2 Software Conformity

Conformity is the second issue cited by Brooks. Software must conform to exacting specifications in the representation of each part, in the interfaces to other internal parts, and in the connections to the environment in which it operates. A missing semicolon or other syntactic error can be detected by a compiler but a defect in the program logic, or a timing error caused by failure to conform to the requirements may be difficult to detect until encountered in operation. Unlike software, tolerance among the interfaces of physical entities is the foundation of manufacturing and construction; no two physical parts that are joined together have, or are required to have, exact matches. Eli Whitney (of cotton gin fame) realized in 1798 that if musket parts were manufactured to specified tolerances, interchangeability of similar (but not identical) parts could be achieved.

There are no corresponding tolerances in the interfaces among software entities or between software entities and their environments. Interfaces among software parts must agree exactly in numbers and types of parameters and kind of couplings. There are no interface specifications for software stating that a parameter can be "an integer plus or minus 2%."

Lack of conformity can cause problems when an existing software component cannot be reused as planned because it does not conform to the needs of the product under development. Lack of conformity might not be discovered until late in a project, thus necessitating development and integration of an acceptable component to replace the one that cannot be reused. This requires unplanned allocation of resources and can delay product completion. Complexity may have made it difficult to determine that the reuse component lacked the necessary conformity until the components it would interact with were completed.

1.3.3 Software Changeability

Changeability is Brooks's third factor that makes software projects difficult. Software coordinates the operation of physical components and provides the functional-

ity in software-intensive systems.[4] Because software is the most easily changed element (i.e., the most malleable) in a software-intensive system, it is the most frequently changed element, particularly in the late stages of a project. Changes may occur because customers change their minds; competing products change; mission objectives change; laws, regulations, and business practices change; underlying hardware and software technology changes (processors, operating systems, application packages); and/or the operating environment of the software changes. If an early version of the final product is installed in the operating environment, it will change that environment and result in new requirements that will require changes to the product. Simply stated, now that the new system enables me to do A and B, I would like for it to also allow me to do C, or to do C instead of B.

Each proposed change in product requirements must be accompanied by an analysis of the impact of the change on project work activities:

- what work products will have to be changed?
- how much time and effort will be required?
- who is available to make the changes?
- how will the change affect your plans for schedule, budget, resources, technology, other product features, and the quality attributes of the product?

The goal of impact analysis is to determine whether a proposed change is "in scope" or "out of scope." In-scope changes to a software product are changes that can be accomplished with little or no disruption to planned work activities. Acceptance of an out-of-scope change to the product requirements must be accompanied by corresponding adjustments to the budget, resources, and/or schedule; and/or modification or elimination of other product requirements. These actions can bring a proposed out-of-scope requirement change into revised scope.

A commonly occurring source of problems in managing software projects is an out-of-scope product change that is not accompanied by corresponding changes to the schedule, resources, budget, and/or technology. The problems thus created include burn-out of personnel from excessive overtime, and reduction in quality because tired people make more mistakes. In addition reviews, testing, and other quality control techniques are often reduced or eliminated because of inadequate time and resources to accomplish the change and maintain these other activities.

1.3.4 Software Invisibility

The fourth of Brooks's factors is invisibility. Software is said to be invisible because it has no physical properties. While the effects of executing software on a digital computer are observable, software itself cannot be seen, tasted, smelled, touched, or heard. Our five human senses are incapable of directly sensing software; software is thus an intangible entity. Work products such as requirements specifications, design documents, source code, and object code are representations of software, but

[4] Software-intensive systems contain one or more digital devices and may include other kinds of hardware plus trained operators who perform manual functions. Nuclear reactors, modern aircraft, automobiles, network servers, and laptop computers are examples of software-intensive systems.

they are not the software. At the most elemental level, software resides in the magnetization and current flow in an enormous number of electronic elements within a digital device. Because software has no physical presence we use different representations, at different levels of abstraction, in an attempt to visualize the inherently invisible entity.

Because software cannot be directly observed as can, for example, a building under construction or an agricultural plot being prepared for planting, the techniques presented in this text can be used to determine the true state of progress of a software project. An unfortunate result of failing to use these techniques is that software products under development are often reported to be "almost complete" for long periods of time with no objective evidence to support or refute the claim; this is the well-known "90% complete syndrome" of software projects. Many software projects have been canceled after large investments of effort, time, and money because no one could objectively determine the status of the work products or provide a credible estimate of a completion date or the cost to complete the project. Sad but true, this will occur again. You do not want to be the manager of one of those projects.

1.3.5 Team-Oriented, Intellect-Intensive Work

In addition to the essential properties of software (complexity, conformity, changeability, and invisibility), one additional factor distinguishes software projects from other kinds of projects: *software projects are team-oriented, intellect-intensive endeavors*. In contrast, assembly-line manufacturing, construction of buildings and roads, planting of rice, and harvesting of fruit are labor-intensive activities; the work is arranged so that each person can perform a task with a high degree of autonomy and a small amount of interaction with others. Productivity increases linearly with the number of workers added; the work will proceed roughly twice as fast if the number of workers is doubled. Although labor-saving machines have increased productivity in some of these areas, the roles played by humans in these kinds of projects are predominantly labor-intensive.

Software is developed by teams of individuals who engage in creative problem solving. Teams are necessary because it would take too much time for one person to develop a modern software system and because it is unlikely that one individual would possess the necessary range of skills. Suppose, for example, that the total effort to develop a software product or system[5] results in a productivity level of 1000 lines of code per staff-month (more on this later). A one million line program would require 1000 staff-months. Because effort (staff-months) is the product of people and time, it would require 1 person 1000 months (about 83 years) to complete the project.

A feasible combination of people and time for a 1000 staff-month project might be a team of 50 people working for 20 months but not 1000 people working for 1 month or even 200 people working for 5 months. The later proposals (1000×1 and

[5] Software *products* are built by *vendors* for sale to numerous customers; software *systems* are built by *contractors* for specific customers on a contractual basis. The terms "system" and "product" are used interchangeably in this text unless the distinction is important; the distinction will be clarified in these cases.

200×5) are not feasible because scheduling constraints among work activities dictate that some activities cannot begin before other work activities are completed: you can't design (some part of a system) without some corresponding requirements, you should not write code without a design specification for (that part of) the system, you cannot review or test code until some code has been written, you cannot integrate software modules until they are available for integration, and so on.

Adding people to a software development team does not, as a rule, increase overall productivity in a linear manner because the increased overhead of communicating with and coordinating work activities among the added people decreases the productivity of the existing team. To cite Fred Brooks once again, the number of communication paths among n workers is $n(n-1)/2$, which is the number of links in a fully connected graph. Five workers have 20 communication paths, 10 have 45 paths, and 20 have 190. Increasing the size of a programming team from 5 to 10 members might, for example, might increase the production rate of the team from 5000 lines of code per week to 7500 lines of code per week, but not 10,000 lines of code per week as would occur with linear scaling. In *The Mythical Man-Month*, Brooks described this phenomenon as Brooks's law[6]:

Adding manpower to a late software project makes it later.

Brooks's law is based on three factors:

1. the time required for existing team members to indoctrinate new team members,
2. the learning curve for the new members, and
3. the increased communication overhead that results from the new and existing members working together.

Brooks's law would not be true if the work assigned to the new members did not invoke any of these three conditions.

A simile that illustrates the issues of team-oriented software development is that of a team of authors writing a book as a collaborative project; a team of authors is very much like a team of software developers. In the beginning, requirements analysis must be performed to determine the kind of book to be written and the constraints that apply to writing it. The number and skills of team members will constrain the kind and size of book that can be written by the available team of authors within a specified time frame. Constraints may include the number of people on the writing team, knowledge and skills of team members, the required completion date, and the word-processing hardware and software available to be used.

Next the structure of the book must be designed: the number of chapters, a brief synopsis of each, and the relationships (interfaces) among chapters must be specified. The book may be structured into sections that contain several chapters each (subsystems), or the text may be structured into multiple volumes (a system of systems). The dynamic structure of the text may flow linearly in time or it may move backward and forward in time between successive chapters; primary and

[6] *Ibid*, pp. 25 and 274.

secondary plot lines may be interleaved. An important constraint is to develop a design structure that will allow each team member to accomplish some work while other team members are accomplishing their work so that the work activities can proceed in parallel. Some books are cleverly structured to have multiple endings; readers choose the one they like.

Design details to be decided include the format of textual layout, fonts to be used, footnoting and referencing conventions, and stylistic guidelines (use of active and passive voice, use of dialects and idioms). Writing of the text occurs within a predetermined schedule of production that includes reviews by other team members (peer reviews) and independent reviews by copy editors (independent verification). Revisions determined by the reviews must be accomplished. The goal of the writing team is to produce a seamless text that appears to have been written by one person in a single setting.

A deviation from the planned narrative by one or more team members might produce a ripple effect that would require extensive revision of the text. If the completed book were software, a single punctuation or grammatical error in the text would render the book unreadable until the writers or their copy editor repaired the defect. An editor determines that each iteration of elements of the text satisfy the conditions placed on it by other elements (verification). Finally, reviews by critics and purchases by readers will determine the degree to which the book satisfies its intended purpose in its intended environment (validation).

The various development phases of writing (analysis, high-level design, detailed design, implementation, peer review, independent verification, revision, and validation) are creative activities and thus rarely occur in linear, sequential fashion. Conducting analysis, preparing and revising the design of the text, and production, review, and revision of the various parts may be overlapped, interleaved, and iterated. Team members must each do their assigned tasks, coordinate their work with other team members, and communicate ideas, problems, and changes on a continuous basis. The narrative above depicts a so-called Plan-driven approach to writing a book and, by analogy, to developing software. An alternative is to pursue an Agile approach by which the team members start with a basic concept and evolve the text in an iterative manner. This approach can be successful:

- if the team is small, say five or six members (to limit the complexity of communication);
- if all members have in mind a common understanding of the desired structure of the text (i.e., a “design metaphor”);
- if there is a strict page limit and a completion date (the project constraints);
- if each iteration occurs in one or a few days (to facilitate ongoing revisions in structure; known as “refactoring”); and
- if a knowledgeable reader (known as the “customer”) is available to review each iteration and provide guidance for the contents of the next iteration.

In some cases, the team members may work in pairs (“pair programming”) to enhance synergy of effort.

In reality, most software projects incorporate elements of a plan-driven approach and an agile approach. When pursuing an agile approach, the team members must

understand the nature of the desired product to be delivered, a design metaphor must be established, and the constraints on schedule, budget, resources, and technology that must be observed; thus some requirements definition, design, and project planning must be done. Those who pursue a plan-driven strategy often pursue an iterative (agile) approach to developing, verifying, and validating the product to be delivered; frequent demonstrations provide tangible evidence of progress and permit incorporation of changes in an incremental manner.

The approach taken in this text is to present a plan-driven strategy, based on iterative development, that is suitable for the largest and most complex projects, and to show how the techniques can be tailored and adapted to suit the needs of small, simple projects as well as large, complex ones. Process models for software development are presented in Chapter 2.

Over time humans have learned to conduct agricultural, construction, and manufacturing projects that employ teams of workers who accomplish their tasks efficiently and effectively.[7] Because software is characterized by complexity, conformity, changeability, and invisibility, and because software projects are conducted by teams of individuals engaged in intellect-intensive teamwork, we humans are not always as adept at conducting software projects as we are at conducting traditional kinds of projects in agriculture, construction, and manufacturing. Nevertheless, the techniques presented in this text will help you manage software projects efficiently and effectively, that is, with economical use of time and resources to achieve desired outcomes.

Your role as project manager is to plan and coordinate the work activities of your project team so that the team can accomplish more working in a coordinated manner than could be accomplished by each individual working with total autonomy.

1.4 THE NATURE OF PROJECT CONSTRAINTS

Many of the problems you will encounter, or have encountered, in software projects are caused by difficulties of management and leadership (i.e., planning, estimating, measuring, controlling, communicating, coordinating, and managing risk) rather than technical issues (i.e., analysis, design, coding, and testing). These difficulties arise from multiple sources; some you can control as a project manager and some you can't. Factors you can't control are called *constraints*, which are limitations imposed by external agents on some or all of the operational domain, operational requirements, product requirements, project scope, budget, resources, completion date, and platform technology. Table 1.1 lists some typical constraints for software projects and provides brief explanations.

The *operational domain* is the environment in which the delivered software will be used. Operational domains include virtually every area of modern society, including health care, finance, transportation, communication, entertainment, business, and manufacturing environments. Understanding the operational domain in which the software will operate is essential to success. *Operational requirements* describe the

[7] To be *efficient* is to accomplish a task without wasting time or resources; to be *effective* is to obtain the desired result.

TABLE 1.1 Typical constraints on software projects

Constraint	Explanation
Operational domain	Environment of the users
Operational requirements	Users' needs and desires
Product requirements	Functional capabilities and quality attributes
Scientific knowledge	Algorithms and data structures
Process standards	Ways of conducting work activities
Project scope	Work activities to be accomplished
Resources	Assets available to conduct a project
Budget	Money used to acquire resources
Completion date	Delivery date for work products
Platform technology	Software tools and hardware/software base
Business goals	Profit, stability, growth
Ethical considerations	Serving best interests of humans and society

users' view (i.e., the external view) of the system to be delivered. Some desired features, as specified in the operational requirements, may be beyond the current state of scientific knowledge, either at large or within your organization. *Product requirements* are the developers' view (i.e., the internal view) of the system to be built; they include the functional capabilities and quality attributes the delivered product must possess in order to satisfy the operational requirements.

Process standards specify ways of conducting the work activities of software projects. Your organization may have standardized ways of conducting specific activities, such as planning and estimating projects, and measuring project factors such as conformance to the schedule, expenditure of resources, and measurement of quality attributes of the evolving product. In some cases the customer may specify standards and guidelines for conducting a project. Four of the most commonly used frameworks for process standards are the Capability Maturity Model Integration (CMMI), ISO/IEEE Standard 12207, IEEE Standard 1058, and the Project Management Body of Knowledge (PMBOK). Elements of these standards and guidelines are contained in appendixes to the chapters of this text.

The *scope* of a project is the set of activities that must be accomplished to deliver an acceptable product on schedule and within budget. *Resources* are the assets, both corporate and external, that can be applied to the project. Resources have both quality and quantity attributes; for example, you may have a sufficient number of software developers available (quantity of assets), but they may not have the necessary skills (quality of assets). The *budget* is the money available to acquire and use resources; the budget for your project may be constrained so that resources available within the organization cannot be utilized. The *completion date* is the day on which the product must be finished and ready for delivery. In some cases there may be multiple completion dates on which subsets of the final product must be delivered. The constrained delivery date(s) may be unrealistic.

Platform technology includes the set of methods, tools, and development environments used to produce or modify a software product. Examples include tools to develop and document requirements and designs, compilers and debuggers to gen-

erate and check the code, version control tools to track evolving versions of a project's work products, and testing tools to aid in verify the software. Platform technology also includes the hardware processors and operating systems on which the software is developed and on which it will operate (which may be the same or different). One or more aspects of the platform technology may be obsolete or otherwise inappropriate for the work to be done.

Business goals may constrain your project to complete the product as soon as possible (to maximize short-term revenue), or to produce the highest possible quality (to maintain credibility with existing customers). *Ethical considerations* may constrain your project from delivering a product with known defects or from incorporating knowledge of a competitor's product gained by unethical methods.

Some of the most difficult problems you will encounter in managing software projects arise from establishing and maintaining a balance among the constraints on project scope, budget, resources, technology, and the scheduled delivery date:

1. scope: the work to be done;
2. budget: the money to acquire resources;
3. resources: the assets to do the job;
4. technology: methods and tools to be used; and
5. delivery date: the date on which the system must be ready for delivery.

The initial balance among these factors is established in your initial project plan. The scope of your project may change during project execution because of changes to product requirements or other factors such as the budget or delivery date. The constraints on your budget, resources, and schedule may change because of internal factors in your organization, changes in the operational environment of the product to be delivered, or competitive pressures. Changes in project scope must always be accompanied by corresponding changes in schedule, budget, resources, and (perhaps) technology.

The constraints listed in Table 1.1 reduce the conceptual space available in which to plan and conduct your project. For example, it may not be possible to deliver a satisfactory product using 10 people for 12 months, but it might be possible if the schedule were extended to 15 months or if the number of people were increased from 10 to 15, or if the requirements for the product were reduced to the functionality that can be delivered with acceptable quality by 10 people in 12 months. In addition to the constraints listed in Table 1.1, there may be political and sociological factors that you cannot control.

Some of the first things you must do in managing a software project are:

1. establish the success criteria for your project,
2. clarify the constraints on the project and the product, and
3. determine whether there is a reasonable chance of meeting the success criteria within the constraints.

Constraints should be clarified to determine whether there is any flexibility or possibility of trade-offs among the constraints because fewer or looser constraints

increase the options for planning and executing your project. There may be priorities among the success criteria of delivering an acceptable product on schedule and within budget; for example, delivering on schedule may be more important than the number of features delivered, or features delivered may be more important than cost. There may be additional success criteria, such as establishing a working relationship with a new customer, or developing a product architecture that provides a basis for developing future products, that is, developing a product-line architecture that consists of base elements and configurable elements.

Factors you will have (or should have) some influence over include:

1. establishing the success criteria,
2. negotiating the project constraints,
3. obtaining consensus among project stakeholders on an initial set of operational requirements, and
4. obtaining consensus among project stakeholders on an initial set of product requirements.

Factors you will have responsibility for include:

5. making initial estimates and plans;
6. maintaining a balance among requirements, schedule, and resources as the project evolves;
7. measuring and controlling the progress of the work;
8. leading the project team and coordinating their work activities;
9. communicating with stakeholders; and
10. managing risk factors that might interfere with, or prevent achieving a successful outcome.

The major activities of project management are planning and estimating, measuring and controlling, communicating and leading, and managing risk factors. Planning and estimating are concerned with determining the scope of activities that must be accomplished, estimating effort and schedule for the overall project, and developing estimates and plans for each major work activity. Planning for measurement involves establishing a data collection and reporting system that will be used to determine and report the actual status of work activities and work products on a continuing basis. Controlling involves applying corrective actions when actual status, as indicated by the measurements, does not agree with planned status.

Communicating involves establishing and maintaining adequate communication channels among all involved parties so that everyone is aware of progress and problems, and so that they are constantly reminded of the goals and success criteria for the project. Leading is concerned with providing direction to, removing roadblocks for, and maintaining the morale of project personnel.

Risk management is concerned with identifying risk factors (potential problems), both initially and on a continuing basis; monitoring identified risk factors; and engaging in risk mitigation activities such as preparing contingency plans and executing them when necessary.

1.5 A WORKFLOW MODEL FOR MANAGING SOFTWARE PROJECTS

The primary objective of a software project is to develop and deliver one or more acceptable work products within the constraints of required features, quality attributes, project scope, budget, resources, completion date, technology, and other factors. The work products to be delivered (e.g., object code, training materials, and installation instructions) result from the flow of intermediate work products that are produced by and flow through the work processes (requirements, design, source code, and test scenarios).

The model of project workflow used in this text is presented in Figure 1.1. All models, including the one in Figure 1.1, are abstractions of real situations that emphasize some aspects of interest and suppress details that are unimportant to the purposes of the model. Important details may be expressed in subordinate models. Subordinate models to Figure 1.1 are presented throughout this text.

Figure 1.1 indicates some of the processes that support the primary activity of Product Development; they include Verification and Validation (V&V), Quality Assurance of work processes and work products (QA), Configuration Management (CM), and others. Some supporting processes and their purposes are listed in Table 1.2. Each supporting process must be accomplished in accordance with a well-defined model for accomplishing the work activities of that process.

The model in Figure 1.1 is called a *process model* because it emphasizes work activities and the flow of work products among work activities. Each work activity in a process model produces one or more work products that provide inputs to subsequent work activities. By *work product* we mean any document produced by a software project (including the source code). Some work products are delivered to the customer (called deliverable work products), while others are intermediate work products developed to advance the creative problem-solving process in an orderly manner. Some of the work products of software projects are listed in Table 1.3.

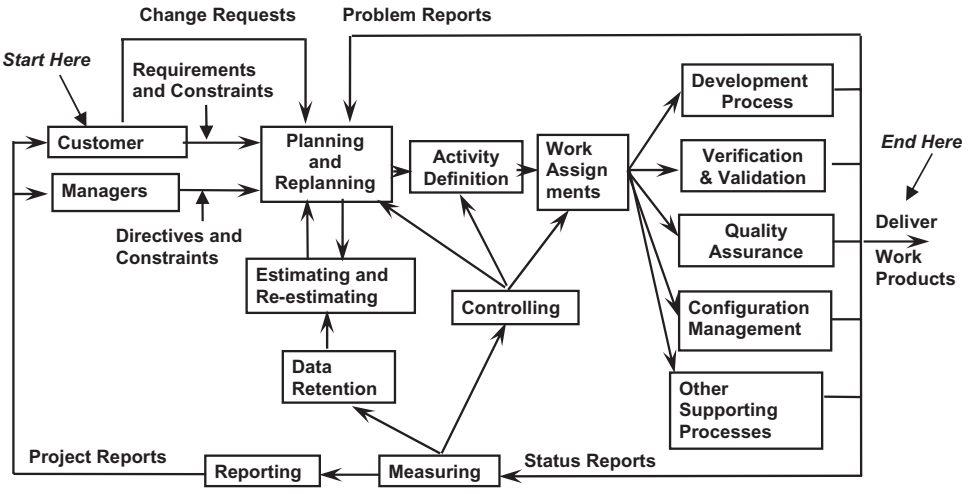

FIGURE 1.1 A workflow model for managing software projects

TABLE 1.2 Some supporting processes for software development

Supporting Process	Purpose
Configuration management	Change control, baseline management, product audits, product builds
Verification	Determining the degree to which work products satisfy the conditions placed on them by other work products and work processes
Validation	Determining the degree of fitness of work products for their intended use in their intended environments
Quality Assurance	Determining conformance of work processes and work products to policies, plans, and procedures
Documentation	Preparation and updating of intermediate and deliverable work products
Developer training	Maintaining adequate and appropriate skills
User and operator training	Imparting skills needed to effectively use and operate systems

TABLE 1.3 Some work-product documents produced by software projects

Document	Content of Document
Project plan	Roadmap for conducting the project
Status reports	State of progress, cost, schedule, and quality
Memos and meeting minutes	Issues, problems, recommendations, and resolutions
e-Mail messages	Ongoing communications
Operational requirements	User needs, desires, and expectations
Technical specification	Product features and quality attributes
Architectural design document	Components and interfaces
Detailed design specification	Algorithms, data structures, and interface details of individual modules
Source code	Product implementation
Test plan	Product verification criteria, test scenarios, and facilities
Reference manual	Product encyclopedia
Help messages	Guidance for users
Release notes	Known issues, hints, and guidelines
Installation instructions	Guidance for operators
Maintenance guide	Guidance for maintainers

As Michael Jackson has observed, the entire description of a software system or product is usually too complex for the entire description to be written directly in a programming language, so we must prepare different descriptions at different levels of abstraction, and for different purposes [Jack02]. Note that each of the work products listed in Table 1.3 is a document; software developers and software project managers do not produce physical artifacts other than documents, which may exist in printed or electronic form.

As illustrated in the workflow model depicted in Figure 1.1, a software project is initiated by customer and managers. A *customer* is the person or organization that

provides the requirements for and accepts the deliverable work products. Customers may place constraints on a project, such as specifying a required database interface (a product constraint) or the date when the delivered system must be available for use (a process constraint). Managers include your management and you, the project manager. Managers specify constraints and directives. A process constraint from your manager might place a limit on the number of people available to conduct the project; a management directive might require that all software projects in the organization perform a design activity. You, the project manager, might issue directives requiring that the design be documented using UML (the Universal Modeling Language) and that one or more design reviews be held.

Requirements, constraints, and directives provide the inputs to the planning process, which is (or should be) a group activity led by you, the project manager. You should involve the customer, selected members of the development team, and other primary stakeholders in the planning process. Planning involves estimation. Factors to be initially estimated include a schedule for conducting the major work activities; kinds and numbers of resources needed, when they will be needed, and for how long; and the project milestones (points in time when progress is assessed). Estimation is best accomplished by using historical data from a data repository. Data at the completion of your project can be placed in a repository to aid in estimation of future projects. Intermediate data can be retained to assess progress and prepare completion estimates, which may result in replanning.

The output of your planning process will include identification of the roles to be played in conducting the project, which results in assignment of personnel to those roles. During initial planning, the major work activities to be planned include software development and the various supporting processes such as configuration management, process and product quality assurance, verification, validation, user training; plus other necessary activities that constitute the scope of your project. Detailed plans for these activities will evolve as the project evolves.

During execution of the project, data are collected and status reports are prepared on a periodic basis by you and your staff. The status reports will be used by you (the project manager), your customer, your managers, support groups, and other project stakeholders. Status reports compare planned progress to actual progress; they may cause you and your customer, working together, to revise plans and requirements, or you might, for example, reassign some personnel to different project roles (e.g., a software designer might be moved to the independent validation team). Status data are also used to provide a basis for estimating future progress based on progress to date (which may result in replanning), and is retained to provide a basis of estimation for future projects.

Problem reports are generated to document defects discovered in work products that must be reworked. Status reports, new requirements, and changes to requirements, constraints, directives, and problem reports provide the data needed to continually update, elaborate, and revise your project plan.

Every organization that develops and maintains software, including yours, should have one or more workflow models of software development that depicts the major work activities and flow of work products. Each member of the organization should be familiar with the workflow model(s) and understand the ways in which their work activities and work products fit into the model(s). Everyone in your software development organization should be able to sketch and describe the workflow model(s)

used in the organization. If there is more than one workflow model, everyone should understand the kinds of projects for which the various models are appropriate.

1.6 ORGANIZATIONAL STRUCTURES FOR SOFTWARE PROJECTS

Projects are one-time, transient events that are initiated to accomplish a specific purpose and are terminated when the project objectives are achieved (and are sometimes cancelled before achieving the objectives). A project exists within the context of the organization in which it is conducted; each project must adhere to the structural model of the organization. Departments that conduct engineering projects, including software projects, are typically organized in one of four ways: functional structure, project structure, matrix structure, or hybrid structure.

1.6.1 Functional Structures

As the name implies, workers in a functional organization are grouped by the functions they perform. Functional groups can be process-oriented or product-oriented. One process-oriented functional group might, for example, specialize in requirements engineering, another in design of user interfaces, another in design and implementation of code, another in product validation, and yet another in user training. When organized by product specialty, one group might specialize in data communication, another in database systems, another in user interfaces, and yet another in numerical algorithms. Figure 1.2 illustrates a process-oriented functional organization, and Figure 1.3 illustrates a product-oriented functional group.

Each functional group has a functional manager whose job is to acquire and maintain the quantity and quality of workers needed to support the projects within the organization, train them as necessary, provide the necessary tools, and coordinate their work activities on various projects. Different group members apply their

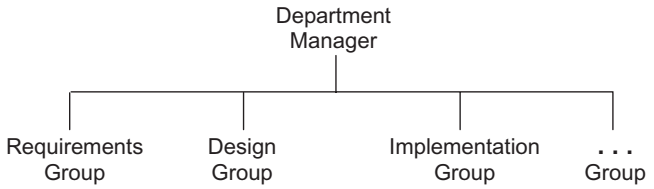

FIGURE 1.2 A process-oriented functional organization

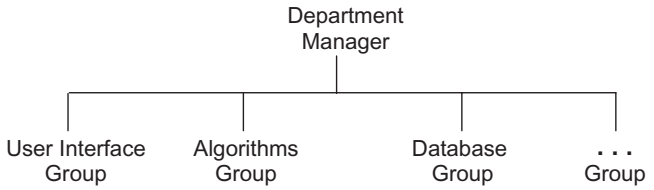

FIGURE 1.3 A product-oriented functional organization

expertise to different projects as needed. As a project manager in a functional organization, responsible for delivering an acceptable product on schedule and within budget, your ability to successfully conduct your project will depend on your skill in working with the functional managers and their team members to complete the various work activities and develop the various work products for your project.

1.6.2 Project Structures

In a purely project-structured organization, you, as project manager, have full authority and responsibility for managing budget and resources. You acquire the kinds of workers you need to conduct your project and all project members report directly to you; you might acquire your workers from functional groups or you might hire them from outside. You, the project manager, have the authority to acquire staff members within the constraints of your budget and to remove them when they are no longer needed or are not performing up to your expectations. Your ability to successfully conduct your project depends on acquiring the quantity and quality of workers needed, training them as necessary, providing the necessary tools, and coordinating their work activities. A project-structured organization is illustrated in Figure 1.4.

1.6.3 Matrix Structures

The goal of a matrix organization is to obtain the advantages of both functional and project structures; functional specialists are assigned to projects as needed and work for you, the project manager, while applying their expertise to your project. When their tasks are completed, they return to their function groups and are assigned, as needed, to other projects. Workers in a matrix organization thus have two bosses: their functional manager and their project manager.

An example of a matrix organization is illustrated in Figure 1.5. The functional groups might be, for example, a user interface group, an algorithms group, a database group, and a communications protocol group. The numbers in the matrix indicate the number of workers of each functional type assigned to each project; for example, project #1 has 10 members: 2 of functional type #1 (user interface), 5 of functional type #3 (database), and 2 of functional type #4 (communications). Project #3 is the largest; it has 23 members. Currently 6 members of the user interface group are assigned to this project, 8 from the algorithms group, 2 from the database group, and 7 from communications.

Matrix organizations can be characterized as weak or strong, depending on the relative authority of the functional managers and the project managers. In a strong

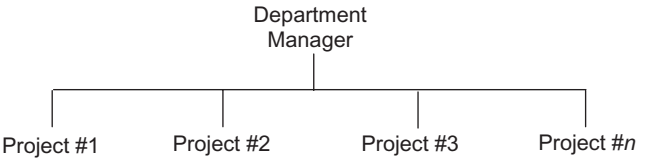

FIGURE 1.4 A project-oriented organization

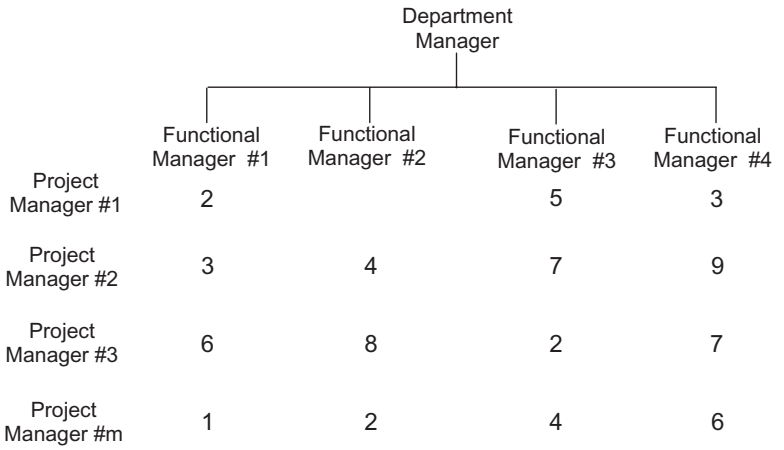

FIGURE 1.5 A matrix-structured organization

matrix, the functional managers have authority to assign workers to projects, and project managers must accept the workers assigned to them. In a weak matrix, the project manager controls the project budget, can reject workers from functional groups and hire outside workers if functional groups do not have sufficient quantities or qualities of workers.

When a matrix organization performs as intended, functional workers apply their specialties to different projects, under the direction of project managers, over time while retaining membership in a group of like-minded experts. Two problems that can occur in matrix organizations are (1) conflicts between functional managers and project managers over the allocation of worker resources (which puts the workers in untenable situations), and (2) frequent shifting of workers from project to project as crises occur (know as "firefighting" mode).

1.6.4 Hybrid Structures

Few, if any, organizations are purely functional, project, or matrix in nature. In a purely functional organization, there would be no project managers; a coordinator at the department level would assign tasks to the functional groups and work products would be passed from group to group as they become available. In a purely project organization, the project would be an entirely separate organization. The project manager would be responsible for physical facilities, janitorial service, human resources (i.e., hiring, firing, payroll, health insurance, and conflict resolution), and other organizational functions. Similarly projects organized in matrix format do not operate in isolation but are dependent on other functional elements of the organization to provide physical facilities, payroll processing, and janitorial service. Figure 1.6 illustrates the organizational continuum from pure function to pure project with matrix organizations occupying the middle region [Youk77].

You, as project manager, will have fewer or more responsibilities and more or fewer constraints on your authority depending on whether your organization has predominantly a functional, matrix, or project structure.

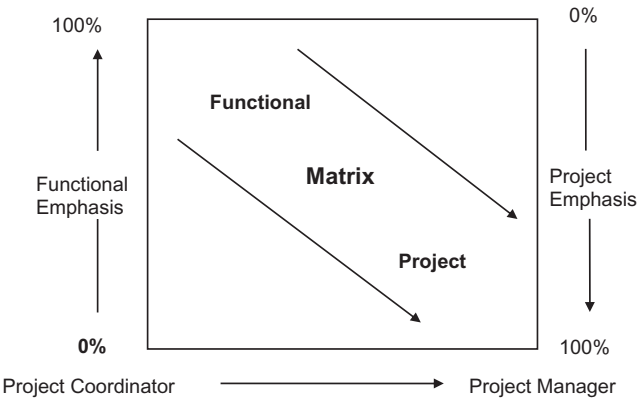

FIGURE 1.6 The organizational continuum [Youk77]

1.7 ORGANIZING THE PROJECT TEAM

The way in which your organization is structured determines the way in which you acquire your project members. It is your job to organize your project team, and to participate, as appropriate as a member of other teams such as the system engineering team.

1.7.1 The System Engineering Team

The responsibilities of systems engineers include:

- defining the operational requirements;
- specifying system requirements;
- developing the system design;
- allocating system requirements to system components;
- integrating the system components as they become available;
- verifying that the system to be delivered is correct, complete, and consistent with respect to its technical specifications; and
- validating operation of the system with its intended users in its intended operational environment.

System engineering, when it exists as a separate entity, is typically a specialty function in an organization. System engineers may be assigned to projects from a functional group within a matrix organization, or they may provide internal consulting to projects while remaining in their functional group. System engineers must be experts in their customer domains and knowledgeable of their organization's capabilities; they are more likely to be long-term organizational members than to be hired from outside the organization by a project manager.

Note that system engineers are not component specialists; they are generalists who understand (must understand) the operational domains of their customers and users and the capabilities of their organizations to develop systems for those domains.

System engineers work with component specialists to specify collections of components that will satisfy user needs and customer expectations.

A system engineering team for a complex, software-intensive system should include hardware, software, and human factors specialists as appropriate for the various kinds of hardware, software, and manual operations of the envisioned system. You, as manager of the software project for a software-intensive system, should be (must be) a member of the system engineering team. In addition the lead technical person on your software team (if you are not that person) and a representative of the group that will maintain the software portion of the system (if that is not your team) should also be members of the system engineering team.

1.7.2 The Software Engineering Team

Every software project, whether stand-alone or a subproject of a system-level program, should include a project manager, a lead designer/software architect, and one or more small development teams, each with a designated team leader. On a small project (up to 10 members), the roles of team leader, project manager, and lead designer may be played by a single individual (you). Or, a project manager may be assigned on a part-time basis with another individual playing the roles of lead designer and team leader. For intermediate-size projects (11 to 20 members), there will be (must be) separate people playing the roles of lead designer and full-time project manager. On large projects (more than 20 members), there may be a design team with a designated chief architect, staff members to support the project manager, and multiple development teams.

Figure 1.7 illustrates a hierarchical model for organizing software projects that can be expanded or contracted to accommodate various sizes of software projects.

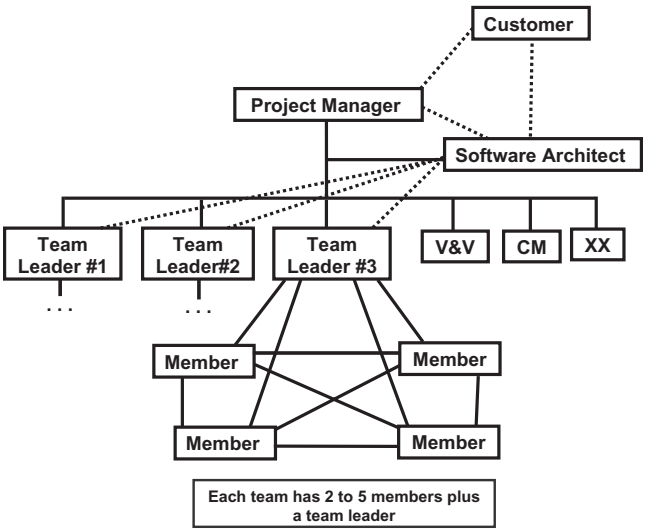

FIGURE 1.7 An organizational model for software projects

A very small project (5 or fewer members) may have only one team whose leader is the project manager and software architect; a project having 5 to 10 members may include two teams and a project manager/software architect. Intermediate-size projects will have one individual playing the role of project manager and another as lead designer; a project having 20 software developers might have 4 teams of 5 members, with one member of each team playing the role of team leader. For projects of more than 50 members, the team leaders depicted in Figure 1.7 will be subsystem managers and subsystem designers with team leaders and their teams reporting to them; a project having 100 software developers might be decomposed into 4 subsystems with, for example, 5 teams of 5 assigned to each subsystem.

A hierarchical project structure, as depicted in Figure 1.7, thus provides a flexible model that can be expanded and contracted as the needs of various projects dictate. The purpose of hierarchical structures is not to restrict the flow of communication within the project but rather to provide well-defined work activities, roles, authorities, and responsibilities at each level in the hierarchy that minimizes the need for communication among different groups. Communication paths among teams are not restricted to the hierarchy; the communication paths are informal networks that are dynamically established and disbanded as appropriate.

To facilitate communication, a fundamental principle of software analysis and design is that the requirements must be partitioned and the design structured so that the work of each small team can proceed concurrently with the work of other teams. The reason for limiting the size of each team is to control the number of intensive communication paths among software developers who are engaged in closely coordinated work activities. As previously mentioned, communication paths can be modeled as links in a fully connected graph where each team member is a node in the graph. The number of links in a fully connected graph of n nodes is $n(n-1)/2$. Five members thus have 10 paths; 10 members have 45.

The need to partition the work into well-defined work activities for multiple teams either by process function (e.g., design, coding, testing) or product function (e.g., database, algorithms, user interface) is particularly important if the team members reside in functional groups or are geographically distributed. In these cases the work to be done must be partitioned so that each functional group or geographic group can proceed with their work activities with a large degree of autonomy from the other groups.

1.8 MAINTAINING THE PROJECT VISION AND THE PRODUCT VISION

Every software project, large or small, simple or complex, must maintain the process vision (the project roadmap) and the product vision (the goals for the product) from beginning to end; otherwise, it is easy to lose sight of vision and goals in the midst of the daily work activities of a project. You, as the project manager, are the keeper of the process vision, which is documented in the project plan (and is updated as the project evolves). The software architect is the keeper of the product vision,

which is documented in the requirements and architectural design specifications (and is updated as the product evolves).[8]

The project manager can be likened to a movie producer and the software architect to a movie director. The producer has overall responsibility for schedules, budgets, resources, customer relations, and delivery of a satisfactory product on time and within budget. The director is responsible for the content of the product. Producer and director must work together to maintain and constantly communicate the process vision and the product vision to the cast of developers and supporting personnel as well as all other project stakeholders.

Fred Brooks observes that producer and director can be the same person on a small project (five to seven developers), but they must be different individuals on larger projects because of the differing skills required and the number of tasks to be performed. As Brooks points out, if you, as project manager (producer) are not also the director (i.e., lead designer), you must "proclaim the director's technical authority. . . . For this to be possible, the producer and director must see alike on fundamental technical philosophy; they must talk out the main technical issues privately, before they really become timely; and the producer must have a high respect for the director's technical prowess."[9] We should add that, conversely, the director must have a high respect for the producer's managerial prowess.

1.9 FRAMEWORKS, STANDARDS, AND GUIDELINES

A *process framework* is a generic process model that can be tailored and adapted to fit the needs of particular projects and organizations. An *engineering standard* is a codification of methods, practices, and procedures that is usually developed and endorsed by a professional society or independent agency. *Guidelines* are pragmatic statements of practices that have been found to be effective in many practical situations.

Some well-known frameworks, standards, and guidelines for software engineering and the associated URLs are:

- the Capability Maturity Model® Integration for development (CMMI-DEV-v1.2) [www.sei.cmu.edu/cmmi/models];
- ISO/IEC and IEEE/EIA Standards 12207 [www.iso.org], [standards.ieee.org/software];
- IEEE/EIA Standard 1058 [standards.ieee.org/software]; and
- the Project Management Body of Knowledge (PMBOK®) [www.pmibookstore.org].

Elements of these models that are relevant to managing and leading software projects are presented in appendixes to the chapters of this text, including Appendix 1A to this chapter.

[8] *Ibid*, pp. 79–83.
[9] *Ibid,* p. 79.

1.10 KEY POINTS OF CHAPTER 1

- A project is a coordinated set of activities that occur within a specific time frame to achieve specific objectives.
- The primary activities of software project management are planning and estimating; measuring and controlling; communicating, coordinating and leading; and managing risk.
- Software projects are inherently difficult because software is complex, changeable, conformable, and invisible.
- Software projects are conducted by teams of individuals who engage in intellect-intensive teamwork.
- Project constraints consist of limitations imposed by external agents on some or all of the operational domain, operational requirements, product requirements, project scope, budget, resources, completion date, and platform technology.
- A workflow model depicts the work activities and the flow of work products among work activities in a software project.
- The entire description of a software system or product is usually too complex for the entire description to be written directly in a programming language, so we must prepare different descriptions at different levels of abstraction, and for different purposes.
- Organizations that conduct software projects use functional, project, weak matrix, and strong matrix structures.
- Software projects organized in a hierarchical manner provide well-defined work activities, roles, authorities, and responsibilities at each level in the hierarchy; hierarchies can expand and shrink to fit the needs of each project.
- Requirements must be allocated and the design structured so that the work of each small team can proceed concurrently with the work of other teams.
- The project manager maintains the project vision, as documented in the project plan, and the software architect maintains the product goals, as documented in the requirements and architectural design.
- A software process framework is a generic process model that can be tailored and adapted to fit the needs of particular projects and organizations.
- A software engineering standard is a codification of methods, practices, and procedures, usually developed and endorsed by a professional society or independent agency.
- SEI, ISO, IEEE, and PMI provide process frameworks, standards, and guidelines that contain information relevant to managing software projects (see Appendix 1A to this chapter).

1.11 OVERVIEW OF THE TEXT

This text is organized into 11 chapters. The first 3 chapters present the context in which software projects are conducted. This chapter provides an overview of and an introduction to managing software projects. Chapter 2 presents commonly used

process models for software development and the project management considerations for each of the models. Chapter 3 describes product and process foundations for software projects. Product foundations include operational requirements, system requirements and system design, design constraints, and software requirements. Process foundations include the workflow model, the software development model, the contractual agreement, and the project plan.

Chapters 4, 5, and 6 are concerned with planning and estimation. Chapter 4 describes the planning process and the format and contents of project management plans. Chapter 5 presents planning techniques, including work breakdown structures, work packages, activity networks (critical paths and PERT), Gantt charts, and resource-loading histograms. Chapter 6 is concerned with estimation techniques, including pragmatic, theory-based, and regression-based techniques.

Chapter 7 presents an introduction to measures and measurement, and measurement and control of work products, including techniques to measure and analyze software defects. Chapter 8 presents measurement and control of work processes, including techniques for measuring and controlling schedule, budget, progress, and risk. Chapter 9 covers risk management, including risk identification, analysis and prioritization, mitigation strategies, action plans and action items, contingency plans and contingent actions, and crisis management.

Chapter 10 covers teamwork, motivation, personality styles, and leadership styles. Chapter 11 covers organizational issues; it concludes with 15 guidelines for organizing and leading software engineering teams.

Each chapter provides exercises; completing them will further your understanding of the topics covered in the chapter. An appendix to each chapter of this text includes relevant topics, keyed to that chapter, from the SEI Capability Maturity Model® Integration CMMI-DEV-v1.2, ISO/IEC and IEEE/EIA Standards 12207, IEEE/EIA Standard 1058, and the PMI Project Management Body of Knowledge (PMBOK®).

Appendix A to this text provides a glossary of terms used throughout the text. Appendix B describes some topics for term projects and a schedule of assignments for a term project to develop a software project management plan. Presentation slides for each chapter and other supporting material are available at the URL listed in the Preface.

REFERENCES

[Brooks95] Brooks, F. P. *The Mythical Man-Month*. Addison Wesley, 1995.

[CMMI06] SEI. *CMMI® Models and Modules*. http://www.sei.cmu.edu/cmmi/models/, 2006.

[IEEE1058] IEEE Std 1058™—1998 *IEEE Standard for Software Project Management Plans*. IEEE Press, New York, 1998. Also in Engineering Standards Collection. IEEE Product: SE113. Institute of Electrical and Electronic Engineers, August 2003.

[IEEE12207] *Industry Implementation of International Standard ISO/IEC 12207:1995 Standard for Information Technology–Software Life Cycle Processes*. IEEE/EIA 12207.0/.1/.2-1996 (March), IEEE Press, New York, 1996. Also in Engineering

Standards Collection. IEEE Product: SE113. Institute of Electrical and Electronic Engineers, August 2003.

[Jack02] Jackson, M. *Descriptions in Software Development*. Lecture Notes in Computer Science. Springer Verlag, 2002.

[PMI04] PMI, *A Guide to the Project Management Body of Knowledge*, 3rd ed. (PMBOK® Guide). Project Management Institute, 2004.

[Youk77] Youker, R. Organizational alternatives for project managers. *Project Management* Quarterly, Vol. VIII, No. 1, (March 1977).

URLs

SEI Capability Maturity Model Integration (CMMI®) [www.sei.cmu.edu/cmmi/models].

ISO/IEC Standard 12207–1995 [www.iso.org].

IEEE/EIA Software Engineering standards, including IEEE/EIA Standard 12207–1996 and IEEE/EIA Standard 1058 [standards.ieee.org/software].

PMI Project Management Body of Knowledge (PMBOK® Guide), 3rd Ed., 2004 [www.pmibookstore.org].

EXERCISES

1.1. A project is a collection of coordinated work activities conducted within a specific time frame that utilizes resources to achieve specified objectives.

- **a.** Briefly describe a project from your personal life that you have recently completed. State the nature of the project, the initial objectives, and planned the starting and ending dates and the actual starting and ending dates of the project. List any resources used (money, tools, materials, labor).
- **b.** List and compare the outcome of your project to the initial objectives.

1.2. Different kinds of projects tailor and adapt the generic techniques of project management (planning, estimating, measuring, controlling, communicating, coordinating, leading, managing risk) to fit the needs of the projects. For each of the following kinds of projects, list some factors that would influence the way you would plan, estimate, measure, control, communicate, coordinate, lead, and managing risk those projects:

- **a.** Building construction
- **b.** Restaurant kitchen
- **c.** Fruit picking
- **c.** Handcrafting of race cars

1.3. A 1,000,000 line of code program, when printed at 50 lines per page, results in stack of paper about 10 feet high (3 meters). Show the calculation of this result. List any assumptions made.

1.4. In the text *The Mythical Man-Month*, Fred Brooks differentiates accidental difficulties from essential difficulties in software engineering. Accidental

difficulties are those that arise because of the current state of our knowledge, processes, tools, and technology. Essential difficulties arise from the inherent complexity, conformity, changeability, and invisibility of software.

a. List and briefly describe five accidental difficulties that make software development difficult.

b. Compare and contrast the current state of your five accidental difficulties to the state of those difficulties in 1960.

1.5. Describe a circumstance in which adding more people to a software project would not invoke Brooks's law; that is, a situation where the 3 factors listed in the text would not apply.

1.6. The text describes the ways in which a team of people writing a book is like a team of people writing software. Read the description and develop a two-column table in which the activities of writing a book are listed in the first column and comparable activities of writing software are listed in the rows of the second column.

1.7. Describe three ways in which a team effort to develop software is not similar to a team effort to write a book.

1.8. Describe a circumstance in which a software team would be:

a. efficient but not effective and

b. effective but not efficient.

1.9. Briefly describe an example of each kind of constraint listed in Table 1.1.

1.10. In an example in the text, it is stated that the project discussed might be successfully completed by 10 developers in 12 months if the 10 were outstanding *team members*. List five attributes of an outstanding team member; include some individual and some team membership skills.

1.11. Table 1.2 lists some supporting processes for software development. List and briefly describe three additional supporting processes that might be needed for some software projects.

1.12. Authority and responsibility are major issues for project managers.

a. Briefly state what is meant by authority.

b. Briefly state what is meant by responsibility.

c. Can authority be delegated? If not, why not? If so, give an example.

d. Can responsibility be delegated? If not, why not? If so, give an example.

e. Briefly explain why authority must be commensurate with responsibility.

1.13. Briefly describe the work environment of a software developer working in a software department organized as:

a. a functional organization

b. a project organization

c. a matrix organization

1.14. Figure 1.3 illustrates an organizational model for software projects. List the kind of work each of the three teams might do if the project is organized:

a. by process component

b. by product component

1.15. In the text, software project managers are compared to movie producers and software architects to movie directors. Briefly explain the roles comparable to project manager and software architect if software projects are compared to:

a. symphony orchestras

b. sports teams (baseball, soccer)

c. an army platoon

1.16. ISO and IEEE standards 12207 include five activities for managing software projects: initiation and scope definition, planning, execution and control, review and evaluation, and closure. Consult a copy of either ISO 12207 or IEEE 12207 and briefly summarize the topics included in each of these five activities.

1.17. The seven processes included in level 2 of the staged representation of CMMI-DEV-v1.2 are some of the most important processes for managing software projects. Access CMMI-DEV-v1.2 at www.sei.cmu.edu/cmmi/models.

a. Briefly summarize, in your own words, the purpose of each of these seven processes.

b. Briefly summarize, in your own words, the Introductory Notes for each of these seven processes.

c. Briefly summarize, in your own words, the related process areas for each of these seven processes.

d. Briefly explain why and how the related process areas are important for the purposes of managing software projects.

APPENDIX 1A

FRAMEWORKS, STANDARDS, AND GUIDELINES FOR MANAGING SOFTWARE PROJECTS

1A.1 THE CMMI-DEV-v1.2 PROCESS FRAMEWORK

CMMI process frameworks are developed and supported by the Software Engineering Institute, which is an affiliate of Carnegie Mellon University [CMMI06]. As stated on the home page for CMMI [http://www.sei.cmu.edu/cmmi/general/general.html]:

> Capability Maturity Model® Integration (CMMI) is a process improvement approach that provides organizations with the essential elements of effective processes. It can be used to guide process improvement across a project, a division, or an entire organization. CMMI helps integrate traditionally separate organizational functions, set process improvement goals and priorities, provide guidance for quality processes, and provide a point of reference for appraising current processes.

This text is not primarily focused on process improvement. However, understanding the goals and adopting the specific practices of the process areas for project management in the CMMI frameworks will improve your ability, and your organization's ability to manage software projects. Thereby your chances of delivering acceptable products on schedule and within budget will be enhanced.

Version 1.2 of CMMI is structured as a framework from which various "constellations" can be derived. CMMI-DEV-v1.2 is the first constellation; see www.sei.cmu.edu/cmmi/models. CMMI-ACQ-v1.2 for acquisition processes has just been released at the time of writing this text. Other constellations of the version 1.2 framework are under development. It is important to note that the v1.2 constellations are not process models but rather frameworks for developing and improving processes that satisfy the goals of the CMMI frameworks.

This text is primarily concerned with the process areas related to managing software and systems projects in CMMI-DEV-v1.2, which contains 22 process areas. Both staged and continuous representations are provided. The staged representa-

tion places each process area into one of five *maturity levels* numbered 1 through 5 and the continuous representation provides *capability levels* for each process area on a scale of 0 to 5. In the staged representation each higher level adds more processes. The maturity levels and their names are listed in Table 1A.1.

The 22 process areas in the staged representation of CMMI-DEV-v1.2 are illustrated in Figure 1A.1. The purposes of each process in Figure 1A.1 are listed in Table 1A.4 of this appendix. In the continuous representation of CMMI-DEV-v1.2 a capability level is determined for each individual process area selected for assessment. All the CMMI processes or any subset of them can be assessed and improved, as determined by business needs of the organization. There are six capability levels, numbered 0 through 5 and named as indicated in Table 1A.2.

In the continuous representation the CMMI processes are grouped into four categories. Categories are not levels; they are a way of grouping related process areas. The process areas in each category are as follows:

TABLE 1A.1 CMMI maturity levels

Maturity Level	Name
Level 1	Initial
Level 2	Managed
Level 3	Defined
Level 4	Quantitatively managed
Level 5	Optimizing

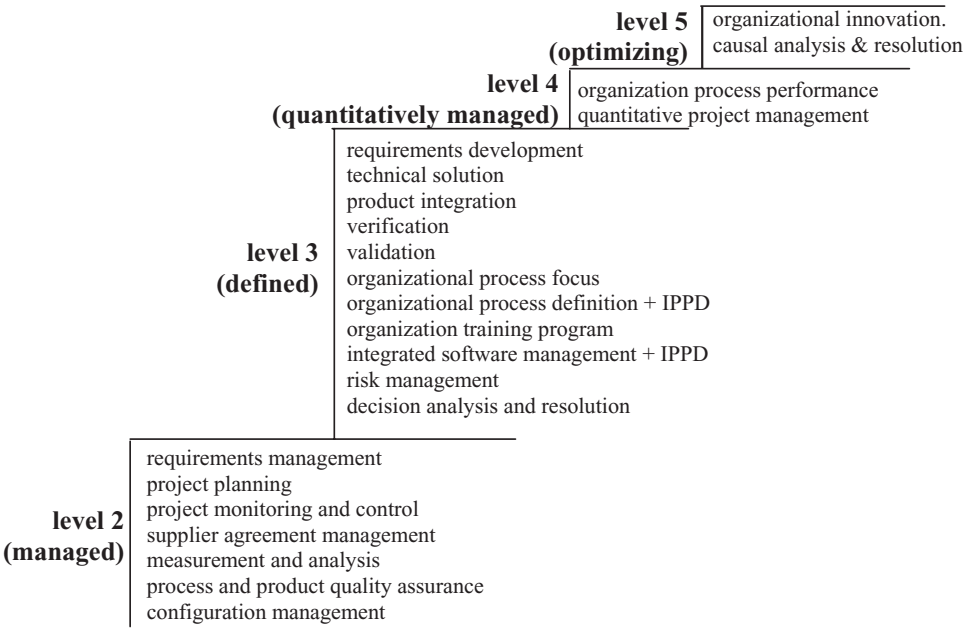

FIGURE 1A.1 Staged representation of the CMMI-DEV-v1.2

TABLE 1A.2 Capability levels in the CMMI continuous representations

Capability Level	Name
Level 0	Incomplete
Level 1	Performed
Level 2	Managed
Level 3	Defined
Level 4	Quantitatively managed
Level 5	Optimizing

- Project management
  - Project planning
  - Project monitoring and control
  - Supplier agreement management
  - Integrated project management + IPPD
  - Risk management
  - Quantitative project management
- Engineering
  - Requirements development
  - Requirements management
  - Technical solution
  - Product integration
  - Verification
  - Validation
- Support
  - Configuration management
  - Process and product quality assurance
  - Measurement and analysis
  - Decision analysis and resolution
  - Causal analysis and resolution
- Process management
  - Organizational process focus
  - Organizational process definition + IPPD
  - Organizational training
  - Organizational process performance
  - Organizational innovation and deployment

Each of the four process categories is divided into basic and advanced process areas. The basic and advanced process areas of project management are listed in Table 1A.3. Note that the basic process areas in Table 1A.3 are level 2 processes in the staged representation of CMMI-DEV-v1.2 and the advanced processes are at level 3 in the staged representation.

Each of the 22 process areas in CMMI-DEV-V1.2 has:

- generic and specific goals (required components),
- generic and specific practices (expected components), and
- informative components, which include typical work products, examples, notes, and references

TABLE 1A.3 Process areas for project management in the continuous representation of CMMI-DEV-v1.2

Basic process areas for project management	• Project planning • Project monitoring and control • Supplier agreement management
Advanced process areas for project management	• Integrated project management + IPPD • Risk management • Quantitative project management

The generic goals and specific goals for a given level, plus all of the goals for lower levels, must be satisfied to reach that level. It is expected that the generic and specific practices will be implemented unless you can demonstrate that you are using equivalent or superior processes. Informative components are illustrative in nature; they are neither required nor expected.

Generic goals and generic practices apply to each process area; their purpose is to institutionalize the process areas so that they are embedded in the corporate memory and corporate procedures. Generic goal 2 (GG2), for example, must be satisfied for level 2 (managed) processes. The generic practices of GG2 are as follows:

GG 2 Institutionalize a managed process
- GP 2.1 Establish an organizational policy
- GP 2.2 Plan the process
- GP 2.3 Provide resources
- GP 2.4 Assign responsibility
- GP 2.5 Train people
- GP 2.6 Manage configurations
- GP 2.7 Identify and involve relevant stakeholders
- GP 2.8 Monitor and control the process
- GP 2.9 Objectively evaluate adherence
- GP 2.10 Review status with higher level management

Satisfying GG3 for a process area assumes that a standard organizational process exists and that you have tailored it to suit the needs of your project. At level 3 (managed) each process is documented (at the organizational level) to specify:

- purpose
- inputs
- entry criteria
- activities
- roles
- measures
- verification steps
- outputs
- exit criteria

At level 2, each project can satisfy the generic and specific goals using different practices, but at level 3, all projects in an organization implement the process areas in a uniform manner so that consistent data can be collected from projects across the organization. Levels 4 and 5 are concerned with analyzing process and product data and using the results to make improvements in processes and technology.

Specific goals and specific practices are, as the name implies, specific to each process area. For example, the specific goals and specific practices of project planning are as follows:

SG 1 Establish estimates
  SP 1.1 Estimate the scope of the project
  SP 1.2 Establish estimates of work product and task attributes
  SP 1.3 Define project life cycle
  SP 1.4 Determine estimates of effort and cost
SG 2 Develop a project plan
  SP 2.1 Establish the budget and schedule
  SP 2.2 Identify project risks
  SP 2.3 Plan for data management
  SP 2.4 Plan for project resources
  SP 2.5 Plan for needed knowledge and skills
  SP 2.6 Plan stakeholder involvement
  SP 2.7 Establish the project plan
SG 3 Obtain commitment to the plan
  SP 3.1 Review plans that affect the project
  SP 3.2 Reconcile work and resource levels
  SP 3.3 Obtain plan commitment

The purpose of the quantitative project management (QPM) process area (a level 3 process in the staged representation) is to quantitatively manage the project's defined process to achieve the project's specified quality and process-performance objectives, namely to manage projects "by the numbers." This involves defining measures for each project phase and each kind of work process, collecting quantita-

TABLE 1A.4 Purposes of the CMMI-DEV-v1.2 processes

Process Area	Purpose
Requirements management	Control requirements and maintain consistency of requirements with plans and work products
Project planning	Establish and maintain the plans that define the project work activities
Project monitoring and control	Compare progress to plans and apply corrective actions as needed
Supplier agreement management	Manage acquisition of product elements from vendors and subcontractors
Measurement and analysis	Supply status information needed to support decisions
Process and product quality assurance	Evaluate processes and work products to identify areas of noncompliance
Configuration management	Establish and maintain control of work products
Requirements development	Obtain, analyze, and develop customer, product, and product-component requirements
Technical solution	Design, develop, and implement solutions that satisfy requirements
Product integration	Integrate components, validate overall functionality, and deliver the product
Verification	Ensure that selected work products meet their specified requirements
Validation	Ensure that selected work products satisfy their intended use when placed in their intended environments
Organizational process focus	Plan and implement organizational process improvement
Organizational process definition + IPPD	Establish and maintain a usable set of organizational process assets
Organizational training	Develop skills and knowledge so that people can perform their jobs efficiently and effectively
Integrated project management + IPPD	Develop and use an integrated and defined set of processes that are tailored from the organization's set of standard processes
Risk management	Identify potential problems; develop and implement strategies and techniques for mitigating them
Decision analysis and resolution	Identify possible decisions using a formal evaluation process that evaluates alternatives against established criteria
Quantitative project management	Use quantified data to manage each project's quality and process-performance objectives
Organizational process performance	Provide process performance data and quantitative models to understand the organization's standard processes
Organizational innovation and deployment	Select and deploy incremental and innovative improvements that measurably improve the organization's processes and technologies
Causal analysis and resolution	Identify causes of defects and other problems and take action to prevent them from occurring in the future

tive data, performing statistical analyses, and comparing results to plans and expectation on an ongoing basis.

A staged maturity level cannot be attained until all of the generic and specific goals of all processes at lower levels plus the generic and specific goals for the processes in that level are satisfied. A higher capability level for an individual process cannot be attained until all of the generic and specific goals of the lower levels plus the generic and specific goals for that level have been attained for that process.

In general, staged representations provide a systematic approach to building process maturity, level by level. Continuous representations allow different organizations to choose the processes to be improved according to the priorities established by those organizations.

Note that levels 4 and 5 in both the staged and continuous representations are termed "quantitatively managed and optimizing." Quantitatively managed process areas are those for which uniformly defined and measured data are collected from all projects across an organization and analyzed for strengths and weaknesses. At level 5 the results of level 4 data analysis are used to improve process areas and to introduce new technologies in support of the process areas. Level 5 is "optimizing" and not "optimized." The latter term (optimized) implies that the organization's processes are as good as possible. In contrast, the former term (optimizing) implies that the organization's processes are being continuously improved but are not optimum; there is always room for improvement.

The purpose of each of the 22 processes in CMMI-DEV-v1.2 is briefly summarized in Table 1A.4. Relevant elements of CMMI-DEV-v1.2 are presented in appendixes to the chapters of this text.

1A.2 ISO/IEC AND IEEE/EIA STANDARDS 12207

ISO/IEC Standard 12207 is a framework for organizing and conducting software life cycle processes. ISO/IEC 12207 was published in 1995 and amended in 2002 and 2004. Amendments 1 and 2 revise 12207 to incorporate lessons learned in using 12207 and to more closely align it with ISO Standard 15504, which is a standard for assessing the software processes within an organization to determine areas of strength and weakness.

ISO/IEC Standard 12207 provides a comprehensive set of life cycle processes for acquisition, supply, development, operation, and maintenance of software. It includes 17 processes:

- 5 primary life cycle processes,
- 8 supporting processes, and
- 4 organizational processes.

The five primary processes are:

- acquisition,
- supply,
- development,

- operation, and
- maintenance.

The acquisition and supply processes are concerned with the relationships between a customer and a supplier. In ISO/IEC 12207, the development process consists of 13 activities:

1. Process implementation
2. System requirements analysis
3. System architectural design
4. Software requirements analysis
5. Software architectural design
6. Software detailed design
7. Software coding and testing
8. Software integration
9. Software qualification testing
10. System integration
11. System qualification testing
12. Software installation
13. Software acceptance support

The eight supporting processes in ISO/IEC 12207 are:

- documentation,
- configuration management,
- quality assurance,
- verification,
- validation,
- joint review,
- audit, and
- problem resolution.

The four organizational life cycle processes are:

- management,
- infrastructure,
- improvement, and
- training.

The management process in ISO/IEC 12207 includes five activities for managing software projects:

- initiation and scope definition,
- planning,

- execution and control,
- review and evaluation, and
- closure.

ISO/IEC 12207 is packaged in three volumes:

- 12207.0, software life cycle processes;
- 12207.1, life cycle data; and
- 12207.2, implementation considerations.

ISO/IEC 12207.0 is the primary document; in addition to specifying primary life cycle processes, supporting processes and organizational life cycle processes, it includes appendixes that provide guidance for tailoring the various processes to fit particular situations.

ISO/IEC 12207.1 (life cycle data) includes generic guidelines for 7 types of documents (e.g., plans, descriptions, records) and specific guidelines for 30 kinds of documents (e.g., project management plans, software design descriptions, software quality assurance records).

ISO/IEC 12207.2 (implementation considerations) provides guidance, based on industry experiences, for implementing the life cycle processes in 12207.0.

The IEEE/EIA version of ISO/IEC Standard 12207 was developed by the Software and Systems Engineering Standards Committee of the IEEE Computer Society [IEEE12207]. Simply stated, IEEE/EIA 12207 is ISO/IEC 12207 with modifications and clarifications of wording and the addition of some appendixes. It is the umbrella standard for the IEEE's suite of approximately 40 standards for software engineering documents and processes [standards.ieee.org/software]; each of those standards is (is intended to be) harmonious with IEEE/EIA 12207.

According to the abstract in IEEE/EIA Standard 12207.0–1996, the standard includes clarifications, additions, and changes accepted by the Institute of Electrical and Electronics Engineers (IEEE) and the Electronic Industries Association (EIA). The goal of the standard is to provide better understanding of and a basis for software practices in both national and international business. According to the Foreword to IEEE/EIA 12207.2, it summarizes the best practices of the U.S. software industry in the context of the process structure provided by ISO/IEC 12207. Relevant elements of the ISO and IEEE Standards 12207 are presented in appendixes to the chapters of this text.

1A.3 IEEE/EIA STANDARD 1058

Project management plans based on IEEE Std 1058™–1998 IEEE Standard for Software Project Management Plans will include plans for [IEEE1058]:

- managerial processes,
- technical processes,

- supporting processes, and
- additional processes.

Plans for managerial processes include:

- a startup plan,
- a work plan,
- a control plan,
- a risk management plan, and
- a closeout plan.

Plans for technical processes include plans for a development process model; methods, tools, and techniques; infrastructure; and product acceptance. Supporting process plans include plans for the eight supporting processes in IEEE/EIA Standard 12207; namely configuration management, verification and validation, documentation, quality assurance, reviews and audits, problem resolution, subcontractor management, and process improvement.

Plans for additional processes include plans for other processes such as user training, installation, or ongoing maintenance and support that may not be required on some projects.

An overview of IEEE/EIA Standard 1058 is presented in Chapter 4 of this text. A template for preparing project management plans based on IEEE/EIA Standard 1058 is contained in Appendix 4B to Chapter 4 of this text; an electronic copy of the template can be accessed at the URL for the text, which is listed in the Preface. Relevant elements of IEEE/EIA Standard 1058 are presented in appendixes to the chapters of this text.

1A.4 THE PMI BODY OF KNOWLEDGE

The PMI Body of Knowledge was developed by the Project Management Institute, which is a nonprofit organization that promotes the profession of project management by sponsoring chapters, special interest groups, and affiliations with colleges and universities [www.pmi.org]. PMI has more than 200,000 members worldwide. PMI's activities include education and knowledge acquisition, professional development and networking, career advancement and professional standards, and products and services. PMI offers a certificate examination by which one can become a certified project management professional.

The Guide to the PMI Body of Knowledge (PMBOK®) covers five process groups [PMI04]:

- project initiation,
- project planning,
- executing a project,
- monitoring and controlling a project, and
- closing a project.

TABLE 1A.5 Chapters in the PMBOK® Guide

Chapter 1	Introduction
Chapter 2	Project Life Cycle and Organization
Chapter 3	Project Management Processes for a Project
Chapter 4	Project Integration Management
Chapter 5	Project Scope Management
Chapter 6	Project Time Management
Chapter 7	Project Cost Management
Chapter 8	Project Quality Management
Chapter 9	Project Human Resource Management
Chapter 10	Project Communication Management
Chapter 11	Project Risk Management
Chapter 12	Project Procurement Management

These five process groups include 44 management processes. PMBOK also includes 34 key competencies for project managers. Titles of the chapters in *A Guide to the Project Management Body of Knowledge*, 3rd ed. (PMBOK® Guide) are listed in Table 1A.5; they indicate the scope of topics addressed by PMBOK [PMI04]. Relevant elements of PMBOK are presented in appendixes to the chapters of this text.