CHAPTER

27 PROJECT SCHEDULING

In the late 1960s, a bright-eyed young engineer was chosen to "write" a computer program for an automated manufacturing application. The reason for his selection was simple. He was the only person in his technical group who had attended a computer programming seminar. He knew the ins and outs of assembly language and FORTRAN but nothing about software engineering and even less about project scheduling and tracking.

His boss gave him the appropriate manuals and a verbal description of what had to be done. He was informed that the project must be completed in two months.

He read the manuals, considered his approach, and began writing code. After two weeks, the boss called him into his office and asked how things were going.

"Really great," said the young engineer with youthful enthusiasm. "This was much simpler than I thought. I'm probably close to 75 percent finished."

QUICK LOOK

What is it? You've selected an appropriate process model, you've identified the software engineering tasks that have to be performed, you estimated the amount of work and the number of people, you know the deadline, you've even considered the risks. Now it's time to connect the dots. That is, you have to create a network of software engineering tasks that will enable you to get the job done on time. Once the network is created, you have to assign responsibility for each task, make sure it gets done, and adapt the network as risks become reality. In a nutshell, that's software project scheduling and tracking.

Who does it? At the project level, software project managers using information solicited from software engineers. At an individual level, software engineers themselves.

Why is it important? In order to build a complex system, many software engineering tasks occur in parallel, and the result of work performed during one task may have a profound effect on work to be conducted in another task. These interdependencies are very difficult to understand without a schedule. It's also virtually impossible to assess progress on a moderate or large software project without a detailed schedule.

What are the steps? The software engineering tasks dictated by the software process model are refined for the functionality to be built. Effort and duration are allocated to each task and a task network (also called an "activity network") is created in a manner that enables the software team to meet the delivery deadline established.

What is the work product? The project schedule and related information are produced.

How do I ensure that I've done it right? Proper scheduling requires that: (1) all tasks appear in the network, (2) effort and timing are intelligently allocated to each task, (3) interdependencies between tasks are properly indicated, (4) resources are allocated for the work to be done, and (5) closely spaced milestones are provided so that progress can be tracked.

time-line charts 732
tracking 734
work breakdown 732

The boss smiled and encouraged the young engineer to keep up the good work. They planned to meet again in a week's time.

A week later the boss called the engineer into his office and asked, "Where are we?"

"Everything's going well," said the youngster, "but I've run into a few small snags. I'll get them ironed out and be back on track soon."

"How does the deadline look?" the boss asked.

"No problem," said the engineer. "I'm close to 90 percent complete."

If you've been working in the software world for more than a few years, you can finish the story. It'll come as no surprise that the young engineer[1] stayed 90 percent complete for the entire project duration and finished (with the help of others) only one month late.

This story has been repeated tens of thousands of times by software developers during the past five decades. The big question is why?

27.1 Basic Concepts

Although there are many reasons why software is delivered late, most can be traced to one or more of the following root causes:

- An unrealistic deadline established by someone outside the software team and forced on managers and practitioners.
- Changing customer requirements that are not reflected in schedule changes.
- An honest underestimate of the amount of effort and/or the number of resources that will be required to do the job.
- Predictable and/or unpredictable risks that were not considered when the project commenced.
- Technical difficulties that could not have been foreseen in advance.
- Human difficulties that could not have been foreseen in advance.
- Miscommunication among project staff that results in delays.
- A failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem.

Quote:

"Excessive or irrational schedules are probably the single most destructive influence in all of software."

Capers Jones

Aggressive (read "unrealistic") deadlines are a fact of life in the software business. Sometimes such deadlines are demanded for reasons that are legitimate, from the point of view of the person who sets the deadline. But common sense says that legitimacy must also be perceived by the people doing the work.

Napoleon once said: "Any commander-in-chief who undertakes to carry out a plan which he considers defective is at fault; he must put forth his reasons, insist on the plan being changed, and finally tender his resignation rather than be the instrument of his army's downfall." These are strong words that many software project managers should ponder.

1 In case you were wondering, this story is autobiographical.

The estimation activities discussed in Chapter 26 and the scheduling techniques described in this chapter are often implemented under the constraint of a defined deadline. If best estimates indicate that the deadline is unrealistic, a competent project manager should "protect his or her team from undue [schedule] pressure . . . [and] reflect the pressure back to its originators" [Pag85].

Quote:

"I love deadlines. I like the whooshing sound they make as they fly by."

Douglas Adams

To illustrate, assume that your software team has been asked to build a real-time controller for a medical diagnostic instrument that is to be introduced to the market in nine months. After careful estimation and risk analysis (Chapter 28), you come to the conclusion that the software, as requested, will require 14 calendar months to create with available staff. How should you proceed?

It is unrealistic to march into the customer's office (in this case the likely customer is marketing/sales) and demand that the delivery date be changed. External market pressures have dictated the date, and the product must be released. It is equally foolhardy to refuse to undertake the work (from a career standpoint). So, what to do? I recommend the following steps in this situation:

? What should you do when management demands a deadline that is impossible?

1. Perform a detailed estimate using historical data from past projects. Determine the estimated effort and duration for the project.
2. Using an incremental process model (Chapter 2), develop a software engineering strategy that will deliver critical functionality by the imposed deadline, but delay other functionality until later. Document the plan.
3. Meet with the customer and (using the detailed estimate), explain why the imposed deadline is unrealistic. Be certain to note that all estimates are based on performance on past projects. Also be certain to indicate the percent improvement that would be required to achieve the deadline as it currently exists.[2] The following comment is appropriate:

   I think we may have a problem with the delivery date for the XYZ controller software. I've given each of you an abbreviated breakdown of development rates for past software projects and an estimate that we've done a number of different ways. You'll note that I've assumed a 20 percent improvement in past development rates, but we still get a delivery date that's 14 calendar months rather than 9 months away.

4. Offer the incremental development strategy as an alternative:

   We have a few options, and I'd like you to make a decision based on them. First, we can increase the budget and bring on additional resources so that we'll have a shot at getting this job done in nine months. But understand that this will increase the risk of poor quality due to the tight time line.[3] Second, we can remove a number of the software functions and capabilities that you're requesting. This will make the preliminary

2 If the required improvement is 10 to 25 percent, it may actually be possible to get the job done. But, more likely, the required improvement in team performance will be greater than 50 percent. This is an unrealistic expectation.

3 You might also add that increasing the number of people does not reduce calendar time proportionally.

version of the product somewhat less functional, but we can announce all functionality and then deliver over the 14-month period. Third, we can dispense with reality and wish the project complete in nine months. We'll wind up with nothing that can be delivered to a customer. The third option, I hope you'll agree, is unacceptable. Past history and our best estimates say that it is unrealistic and a recipe for disaster.

There will be some grumbling, but if a solid estimate based on good historical data is presented, it's likely that negotiated versions of option 1 or 2 will be chosen. The unrealistic deadline evaporates.

27.2 Project Scheduling

Fred Brooks was once asked how software projects fall behind schedule. His response was as simple as it was profound: "One day at a time."

The reality of a technical project (whether it involves building a hydroelectric plant or developing an operating system) is that hundreds of small tasks must occur to accomplish a larger goal. Some of these tasks lie outside the mainstream and may be completed without worry about impact on project completion date. Other tasks lie on the "critical path." If these "critical" tasks fall behind schedule, the completion date of the entire project is put into jeopardy.

The tasks required to achieve a project manager's objective should not be performed manually. There are many excellent scheduling tools. Use them.

As a project manager, your objective is to define all project tasks, build a network that depicts their interdependencies, identify the tasks that are critical within the network, and then track their progress to ensure that delay is recognized "one day at a time." To accomplish this, you must have a schedule that has been defined at a degree of resolution that allows progress to be monitored and the project to be controlled.

Software project scheduling is an action that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks. It is important to note, however, that the schedule evolves over time. During early stages of project planning, a macroscopic schedule is developed. This type of schedule identifies all major process framework activities and the product functions to which they are applied. As the project gets under way, each entry on the macroscopic schedule is refined into a detailed schedule. Here, specific software actions and tasks (required to accomplish an activity) are identified and scheduled.

"Overly optimistic scheduling doesn't result in shorter actual schedules, it results in longer ones."

Steve McConnell

Scheduling for software engineering projects can be viewed from two rather different perspectives. In the first, an end date for release of a computer-based system has already (and irrevocably) been established. The software organization is constrained to distribute effort within the prescribed time frame. The second view of software scheduling assumes that rough chronological bounds have been discussed but that the end date is set by the software engineering organization. Effort is distributed to make best use of resources, and an end date is defined after careful analysis of the software. Unfortunately, the first situation is encountered far more frequently than the second.

27.2.1 Basic Principles

Like all other areas of software engineering, a number of basic principles guide software project scheduling:

When you develop a schedule, compartmentalize the work, note task interdependencies, allocate effort and time to each task, and define responsibilities, outcomes, and milestones.

Compartmentalization. The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and the process are refined.

Interdependency. The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence, while others can occur in parallel. Some activities cannot commence until the work product produced by another is available. Other activities can occur independently.

Time allocation. Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort). In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis.

Effort validation. Every project has a defined number of people on the software team. As time allocation occurs, you must ensure that no more than the allocated number of people has been scheduled at any given time. For example, consider a project that has three assigned software engineers (e.g., three person-days are available per day of assigned effort[4]). On a given day, seven concurrent tasks must be accomplished. Each task requires 0.50 person-days of effort. More effort has been allocated than there are people to do the work.

Defined responsibilities. Every task that is scheduled should be assigned to a specific team member.

Defined outcomes. Every task that is scheduled should have a defined outcome. For software projects, the outcome is normally a work product (e.g., the design of a component) or a part of a work product. Work products are often combined in deliverables.

Defined milestones. Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality (Chapter 15) and has been approved.

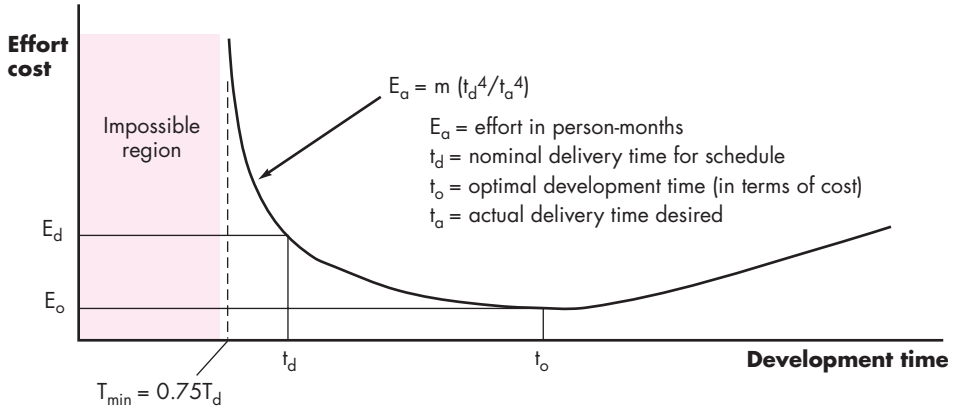
Each of these principles is applied as the project schedule evolves.

27.2.2 The Relationship Between People and Effort

In a small software development project a single person can analyze requirements, perform design, generate code, and conduct tests. As the size of a project increases, more people must become involved. (We can rarely afford the luxury of approaching a 10 person-year effort with one person working for 10 years!)

4 In reality, less than three person-days of effort are available because of unrelated meetings, sickness, vacation, and a variety of other reasons. For our purposes, however, we assume 100 percent availability.

FIGURE 27.1
The relationship between effort and delivery time



ADVICE

If you must add people to a late project, be sure that you've assigned them work that is highly compartmentalized.

There is a common myth that is still believed by many managers who are responsible for software development projects: "If we fall behind schedule, we can always add more programmers and catch up later in the project." Unfortunately, adding people late in a project often has a disruptive effect on the project, causing schedules to slip even further. The people who are added must learn the system, and the people who teach them are the same people who were doing the work. While teaching, no work is done, and the project falls further behind.

In addition to the time it takes to learn the system, more people increase the number of communication paths and the complexity of communication throughout a project. Although communication is absolutely essential to successful software development, every new communication path requires additional effort and therefore additional time.

Over the years, empirical data and theoretical analysis have demonstrated that project schedules are elastic. That is, it is possible to compress a desired project completion date (by adding additional resources) to some extent. It is also possible to extend a completion date (by reducing the number of resources).

The *Putnam-Norden-Rayleigh (PNR) Curve*[5] provides an indication of the relationship between effort applied and delivery time for a software project. A version of the curve, representing project effort as a function of delivery time, is shown in Figure 27.1. The curve indicates a minimum value t_o that indicates the least cost for delivery (i.e., the delivery time that will result in the least effort expended). As we move left of t_o (i.e., as we try to accelerate delivery), the curve rises nonlinearly.

KEY POINT

If delivery can be delayed, the PNR curve indicates that project costs can be reduced substantially.

As an example, we assume that a project team has estimated a level of effort E_d will be required to achieve a nominal delivery time t_d that is optimal in terms of

5 Original research can be found in [Nor70] and [Put78].

schedule and available resources. Although it is possible to accelerate delivery, the curve rises very sharply to the left of t_d. In fact, the PNR curve indicates that the project delivery time cannot be compressed much beyond $0.75t_d$. If we attempt further compression, the project moves into "the impossible region" and risk of failure becomes very high. The PNR curve also indicates that the lowest cost delivery option, $t_o = 2t_d$. The implication here is that delaying project delivery can reduce costs significantly. Of course, this must be weighed against the business cost associated with the delay.

The software equation [Put92] introduced in Chapter 26 is derived from the PNR curve and demonstrates the highly nonlinear relationship between chronological time to complete a project and human effort applied to the project. The number of delivered lines of code (source statements), *L,* is related to effort and development time by the equation:

$$L = P \times E^{1/3} t^{4/3}$$

where E is development effort in person-months, P is a productivity parameter that reflects a variety of factors that lead to high-quality software engineering work (typical values for P range between 2000 and 12,000), and t is the project duration in calendar months.

Rearranging this software equation, we can arrive at an expression for development effort E:

$$E = \frac{L^3}{P^3 t^4} \tag{27.1}$$

where E is the effort expended (in person-years) over the entire life cycle for software development and maintenance and t is the development time in years. The equation for development effort can be related to development cost by the inclusion of a burdened labor rate factor ($/person-year).

As the project deadline becomes tighter and tighter, you reach a point at which the work cannot be completed on schedule, regardless of the number of people doing the work. Face reality and define a new delivery date.

This leads to some interesting results. Consider a complex, real-time software project estimated at 33,000 LOC, 12 person-years of effort. If eight people are assigned to the project team, the project can be completed in approximately 1.3 years. If, however, we extend the end date to 1.75 years, the highly nonlinear nature of the model described in Equation (27.1) yields:

$$E = \frac{L^3}{P^3 t^4} \sim 3.8 \text{ person-years}$$

This implies that, by extending the end date by six months, we can reduce the number of people from eight to four! The validity of such results is open to debate, but the implication is clear: benefit can be gained by using fewer people over a somewhat longer time span to accomplish the same objective.

27.2.3 Effort Distribution

Each of the software project estimation techniques discussed in Chapter 26 leads to estimates of work units (e.g., person-months) required to complete software

development. A recommended distribution of effort across the software process is often referred to as the *40–20–40 rule.* Forty percent of all effort is allocated to front-end analysis and design. A similar percentage is applied to back-end testing. You can correctly infer that coding (20 percent of effort) is deemphasized.

? How should effort be distributed across the software process workflow?

This effort distribution should be used as a guideline only.[6] The characteristics of each project dictate the distribution of effort. Work expended on project planning rarely accounts for more than 2 to 3 percent of effort, unless the plan commits an organization to large expenditures with high risk. Customer communication and requirements analysis may comprise 10 to 25 percent of project effort. Effort expended on analysis or prototyping should increase in direct proportion with project size and complexity. A range of 20 to 25 percent of effort is normally applied to software design. Time expended for design review and subsequent iteration must also be considered.

Because of the effort applied to software design, code should follow with relatively little difficulty. A range of 15 to 20 percent of overall effort can be achieved. Testing and subsequent debugging can account for 30 to 40 percent of software development effort. The criticality of the software often dictates the amount of testing that is required. If software is human rated (i.e., software failure can result in loss of life), even higher percentages are typical.

27.3 Defining a Task Set for the Software Project

Regardless of the process model that is chosen, the work that a software team performs is achieved through a set of tasks that enable you to define, develop, and ultimately support computer software. No single task set is appropriate for all projects. The set of tasks that would be appropriate for a large, complex system would likely be perceived as overkill for a small, relatively simple software product. Therefore, an effective software process should define a collection of task sets, each designed to meet the needs of different types of projects.

As I noted in Chapter 2, a task set is a collection of software engineering work tasks, milestones, work products, and quality assurance filters that must be accomplished to complete a particular project. The task set must provide enough discipline to achieve high software quality. But, at the same time, it must not burden the project team with unnecessary work.

In order to develop a project schedule, a task set must be distributed on the project time line. The task set will vary depending upon the project type and the degree of rigor with which the software team decides to do its work. Although it is difficult

6 Today, the 40-20-40 rule is under attack. Some believe that more than 40 percent of overall effort should be expended during analysis and design. On the other hand, some proponents of agile development (Chapter 3) argue that less time should be expended "up front" and that a team should move quickly to construction.

to develop a comprehensive taxonomy of software project types, most software organizations encounter the following projects:

WebRef

An adaptable process model (APM) has been developed to assist in defining task sets for various software projects. A complete description of the APM can be found at **www.rspa.com/apm**.

1. *Concept development projects* that are initiated to explore some new business concept or application of some new technology.
2. *New application development* projects that are undertaken as a consequence of a specific customer request.
3. *Application enhancement* projects that occur when existing software undergoes major modifications to function, performance, or interfaces that are observable by the end user.
4. *Application maintenance projects* that correct, adapt, or extend existing software in ways that may not be immediately obvious to the end user.
5. *Reengineering projects* that are undertaken with the intent of rebuilding an existing (legacy) system in whole or in part.

Even within a single project type, many factors influence the task set to be chosen. These include [Pre05]: size of the project, number of potential users, mission criticality, application longevity, stability of requirements, ease of customer/developer communication, maturity of applicable technology, performance constraints, embedded and nonembedded characteristics, project staff, and reengineering factors. When taken in combination, these factors provide an indication of the *degree of rigor* with which the software process should be applied.

27.3.1 A Task Set Example

Concept development projects are initiated when the potential for some new technology must be explored. There is no certainty that the technology will be applicable, but a customer (e.g., marketing) believes that potential benefit exists. Concept development projects are approached by applying the following actions:

1.1 **Concept scoping** determines the overall scope of the project.

1.2 **Preliminary concept planning** establishes the organization's ability to undertake the work implied by the project scope.

1.3 **Technology risk assessment** evaluates the risk associated with the technology to be implemented as part of the project scope.

1.4 **Proof of concept** demonstrates the viability of a new technology in the software context.

1.5 **Concept implementation** implements the concept representation in a manner that can be reviewed by a customer and is used for "marketing" purposes when a concept must be sold to other customers or management.

1.6 **Customer reaction** to the concept solicits feedback on a new technology concept and targets specific customer applications.

A quick scan of these actions should yield few surprises. In fact, the software engineering flow for concept development projects (and for all other types of projects as well) is little more than common sense.

27.3.2 Refinement of Software Engineering Actions

The software engineering actions described in the preceding section may be used to define a macroscopic schedule for a project. However, the macroscopic schedule must be refined to create a detailed project schedule. Refinement begins by taking each action and decomposing it into a set of tasks (with related work products and milestones).

As an example of task decomposition, consider Action 1.1, Concept Scoping. Task refinement can be accomplished using an outline format, but in this book, a process design language approach is used to illustrate the flow of the concept scoping action:

```
Task definition: Action 1.1 Concept Scoping
1.1.1   Identify need, benefits and potential customers;
1.1.2   Define desired output/control and input events that drive the application;
        Begin Task 1.1.2
        1.1.2.1 TR: Review written description of need[7]
        1.1.2.2 Derive a list of customer visible outputs/inputs
        1.1.2.3 TR: Review outputs/inputs with customer and revise as required; endtask
        Task 1.1.2
1.1.3   Define the functionality/behavior for each major function;
        Begin Task 1.1.3
        1.1.3.1 TR: Review output and input data objects derived in task 1.1.2;
        1.1.3.2 Derive a model of functions/behaviors;
        1.1.3.3 TR: Review functions/behaviors with customer and revise as required;
        endtask Task 1.1.3
1.1.4   Isolate those elements of the technology to be implemented in software;
1.1.5   Research availability of existing software;
1.1.6   Define technical feasibility;
1.1.7   Make quick estimate of size;
1.1.8   Create a scope definition;
endtask definition: Action 1.1
```

The tasks and subtasks noted in the process design language refinement form the basis for a detailed schedule for the concept scoping action.

7 TR indicates that a technical review (Chapter 15) is to be conducted.

27.4 Defining a Task Network

KEY POINT

The task network is a useful mechanism for depicting intertask dependencies and determining the critical path.

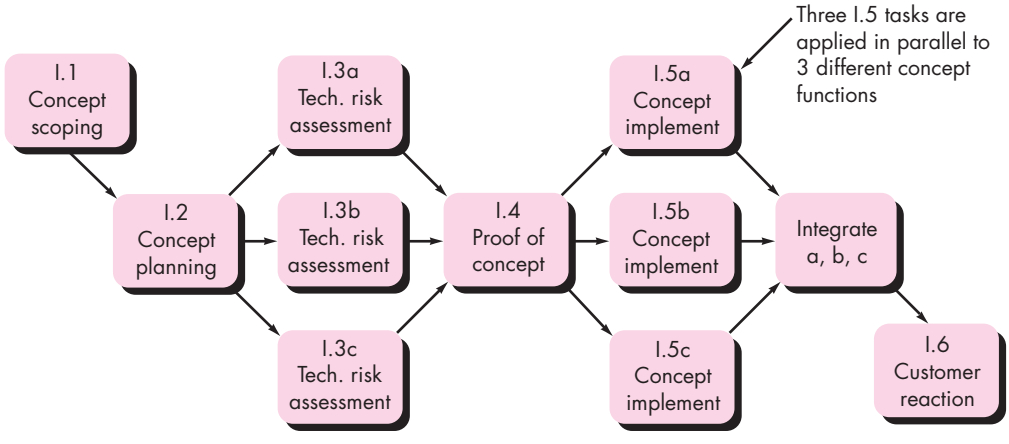
Individual tasks and subtasks have interdependencies based on their sequence. In addition, when more than one person is involved in a software engineering project, it is likely that development activities and tasks will be performed in parallel. When this occurs, concurrent tasks must be coordinated so that they will be complete when later tasks require their work product(s).

A *task network,* also called an *activity network,* is a graphic representation of the task flow for a project. It is sometimes used as the mechanism through which task sequence and dependencies are input to an automated project scheduling tool. In its simplest form (used when creating a macroscopic schedule), the task network depicts major software engineering actions. Figure 27.2 shows a schematic task network for a concept development project.

The concurrent nature of software engineering actions leads to a number of important scheduling requirements. Because parallel tasks occur asynchronously, you should determine intertask dependencies to ensure continuous progress toward completion. In addition, you should be aware of those tasks that lie on the *critical path.* That is, tasks that must be completed on schedule if the project as a whole is to be completed on schedule. These issues are discussed in more detail later in this chapter.

It is important to note that the task network shown in Figure 27.2 is macroscopic. In a detailed task network (a precursor to a detailed schedule), each action shown in the figure would be expanded. For example, Task 1.1 would be expanded to show all tasks detailed in the refinement of Actions 1.1 shown in Section 27.3.2.

FIGURE 27.2 A task network for concept development



27.5 SCHEDULING

Quote:

"*All* we have to decide is what to do with the time that is given to us."

Gandalf in *The Lord of the Rings: Fellowship of the Rings*

Scheduling of a software project does not differ greatly from scheduling of any multitask engineering effort. Therefore, generalized project scheduling tools and techniques can be applied with little modification for software projects.

Program evaluation and review technique (PERT) and the *critical path method* (CPM) are two project scheduling methods that can be applied to software development. Both techniques are driven by information already developed in earlier project planning activities: estimates of effort, a decomposition of the product function, the selection of the appropriate process model and task set, and decomposition of the tasks that are selected.

Interdependencies among tasks may be defined using a task network. Tasks, sometimes called the project *work breakdown structure* (WBS), are defined for the product as a whole or for individual functions.

Both PERT and CPM provide quantitative tools that allow you to (1) determine the critical path—the chain of tasks that determines the duration of the project, (2) establish "most likely" time estimates for individual tasks by applying statistical models, and (3) calculate "boundary times" that define a time "window" for a particular task.

27.5.1 Time-Line Charts

When creating a software project schedule, you begin with a set of tasks (the work breakdown structure). If automated tools are used, the work breakdown is input as

SOFTWARE TOOLS

Project Scheduling

Objective: The objective of project scheduling tools is to enable a project manager to define work tasks; establish their dependencies; assign human resources to tasks; and develop a variety of graphs, charts, and tables that aid in tracking and control of the software project.

Mechanics: In general, project scheduling tools require the specification of a work breakdown structure of tasks or the generation of a task network. Once the task breakdown (an outline) or network is defined, start and end dates, human resources, hard deadlines, and other data are attached to each. The tool then generates a variety of time-line charts and other tables that enable a manager to assess the task flow of a project. These data can be updated continually as the project is under way.

Representative Tools:[8]

AMS Realtime, developed by Advanced Management Systems (**www.amsusa.com**), provides scheduling capabilities for projects of all sizes and types.

Microsoft Project, developed by Microsoft (**www.microsoft.com**), is the most widely used PC-based project scheduling tool.

4C, developed by *4C Systems* (**www.4csys.com**), supports all aspects of project planning including scheduling.

A comprehensive list of project management software vendors and products can be found at **www.infogoal.com/pmc/pmcswr.htm**.

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

a task network or task outline. Effort, duration, and start date are then input for each task. In addition, tasks may be assigned to specific individuals.

KEY POINT

A time-line chart enables you to determine what tasks will be conducted at a given point in time.

As a consequence of this input, a *time-line chart,* also called a *Gantt chart,* is generated. A time-line chart can be developed for the entire project. Alternatively, separate charts can be developed for each project function or for each individual working on the project.

Figure 27.3 illustrates the format of a time-line chart. It depicts a part of a software project schedule that emphasizes the concept scoping task for a word-processing (WP) software product. All project tasks (for concept scoping) are listed in the left-hand column. The horizontal bars indicate the duration of each task. When multiple bars occur at the same time on the calendar, task concurrency is implied. The diamonds indicate milestones.

Once the information necessary for the generation of a time-line chart has been input, the majority of software project scheduling tools produce *project tables*—a tabular listing of all project tasks, their planned and actual start and end dates, and a variety of related information (Figure 27.4). Used in conjunction with the time-line chart, project tables enable you to track progress.

FIGURE 27.3 An example time-line chart

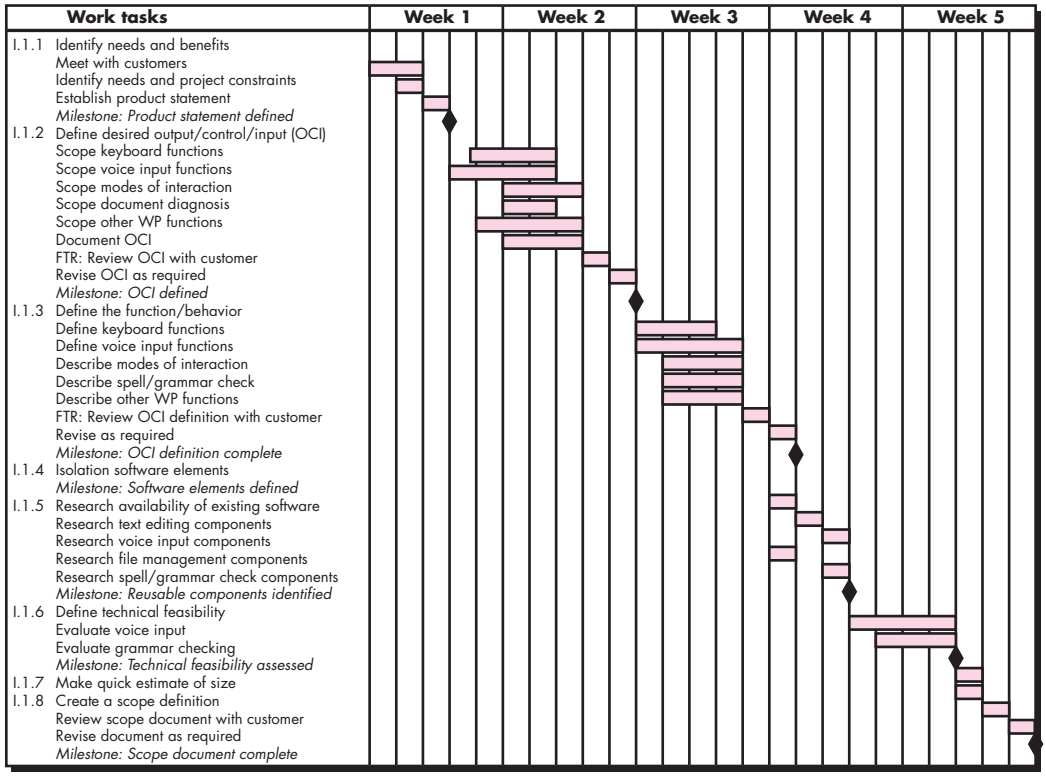


FIGURE 27.4 An example project table

Work tasks	Planned start	Actual start	Planned complete	Actual complete	Assigned person	Effort allocated	Notes
I.1.1 Identify needs and benefits							Scoping will require more effort/time
Meet with customers	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 p-d	
Identify needs and project constraints	wk1, d2	wk1, d2	wk1, d2	wk1, d2	JPP	1 p-d	
Establish product statement	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 p-d	
Milestone: Product statement defined	wk1, d3	wk1, d3	wk1, d3	wk1, d3			
I.1.2 Define desired output/control/input (OCI)							
Scope keyboard functions	wk1, d4	wk1, d4	wk2, d2		BLS	1.5 p-d	
Scope voice input functions	wk1, d3	wk1, d3	wk2, d2		JPP	2 p-d	
Scope modes of interaction	wk2, d1		wk2, d3		MLL	1 p-d	
Scope document diagnostics	wk2, d1		wk2, d2		BLS	1.5 p-d	
Scope other WP functions	wk1, d4	wk1, d4	wk2, d3		JPP	2 p-d	
Document OCI	wk2, d1		wk2, d3		MLL	3 p-d	
FTR: Review OCI with customer	wk2, d3		wk2, d3		all	3 p-d	
Revise OCI as required	wk2, d4		wk2, d4		all	3 p-d	
Milestone: OCI defined	wk2, d5		wk2, d5				
I.1.3 Define the function/behavior							

27.5.2 Tracking the Schedule

Quote:

"The basic rule of software status reporting can be summarized in a single phrase: 'No surprises'."

Capers Jones

If it has been properly developed, the project schedule becomes a road map that defines the tasks and milestones to be tracked and controlled as the project proceeds. Tracking can be accomplished in a number of different ways:

- Conducting periodic project status meetings in which each team member reports progress and problems
- Evaluating the results of all reviews conducted throughout the software engineering process
- Determining whether formal project milestones (the diamonds shown in Figure 27.3) have been accomplished by the scheduled date
- Comparing the actual start date to the planned start date for each project task listed in the resource table (Figure 27.4)
- Meeting informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon
- Using earned value analysis (Section 27.6) to assess progress quantitatively

ADVICE

The best indication of progress is the completion and successful review of a defined software work product.

In reality, all of these tracking techniques are used by experienced project managers.

Control is employed by a software project manager to administer project resources, cope with problems, and direct project staff. If things are going well (i.e., the project is on schedule and within budget, reviews indicate that real progress is being made and milestones are being reached), control is light. But when problems

occur, you must exercise control to reconcile them as quickly as possible. After a problem has been diagnosed, additional resources may be focused on the problem area: staff may be redeployed or the project schedule can be redefined.

When faced with severe deadline pressure, experienced project managers sometimes use a project scheduling and control technique called *time-boxing* [Jal04]. The time-boxing strategy recognizes that the complete product may not be deliverable by the predefined deadline. Therefore, an incremental software paradigm (Chapter 2) is chosen, and a schedule is derived for each incremental delivery.

KEY POINT

When the defined completion date of a time-boxed task is reached, work ceases for that task and the next task begins.

The tasks associated with each increment are then time-boxed. This means that the schedule for each task is adjusted by working backward from the delivery date for the increment. A "box" is put around each task. When a task hits the boundary of its time box (plus or minus 10 percent), work stops and the next task begins.

The initial reaction to the time-boxing approach is often negative: "If the work isn't finished, how can we proceed?" The answer lies in the way work is accomplished. By the time the time-box boundary is encountered, it is likely that 90 percent of the task has been completed.[9] The remaining 10 percent, although important, can (1) be delayed until the next increment or (2) be completed later if required. Rather than becoming "stuck" on a task, the project proceeds toward the delivery date.

27.5.3 Tracking Progress for an OO Project

Although an iterative model is the best framework for an OO project, task parallelism makes project tracking difficult. You may have difficulty establishing meaningful milestones for an OO project because a number of different things are happening at once. In general, the following major milestones can be considered "completed" when the criteria noted have been met.

Technical milestone: OO analysis completed

- All classes and the class hierarchy have been defined and reviewed.
- Class attributes and operations associated with a class have been defined and reviewed.
- Class relationships (Chapter 6) have been established and reviewed.
- A behavioral model (Chapter 7) has been created and reviewed.
- Reusable classes have been noted.

Technical milestone: OO design completed

- The set of subsystems has been defined and reviewed.
- Classes are allocated to subsystems and reviewed.
- Task allocation has been established and reviewed.

9 A cynic might recall the saying: "The first 90 percent of the system takes 90 percent of the time; the remaining 10 percent of the system takes 90 percent of the time."

- Responsibilities and collaborations have been identified.
- Attributes and operations have been designed and reviewed.
- The communication model has been created and reviewed.

Technical milestone: OO programming completed

- Each new class has been implemented in code from the design model.
- Extracted classes (from a reuse library) have been implemented.
- Prototype or increment has been built.

Technical milestone: OO testing

Debugging and testing occur in concert with one another. The status of debugging is often assessed by considering the type and number of "open" errors (bugs).

- The correctness and completeness of OO analysis and design models has been reviewed.
- A class-responsibility-collaboration network (Chapter 6) has been developed and reviewed.
- Test cases are designed, and class-level tests (Chapter 19) have been conducted for each class.
- Test cases are designed, and cluster testing (Chapter 19) is completed and the classes are integrated.
- System-level tests have been completed.

Recalling that the OO process model is iterative, each of these milestones may be revisited as different increments are delivered to the customer.

27.5.4 Scheduling for WebApp Projects

WebApp project scheduling distributes estimated effort across the planned time line (duration) for building each WebApp increment. This is accomplished by allocating the effort to specific tasks. It is important to note, however, that the overall WebApp schedule evolves over time. During the first iteration, a macroscopic schedule is developed. This type of schedule identifies all WebApp increments and projects the dates on which each will be deployed. As the development of an increment gets under way, the entry for the increment on the macroscopic schedule is refined into a detailed schedule. Here, specific development tasks (required to accomplish an activity) are identified and scheduled.

As an example of macroscopic scheduling, consider the **SafeHomeAssured.com** WebApp. Recalling earlier discussions of **SafeHomeAssured.com**, seven increments can be identified for the Web-based component of the project:

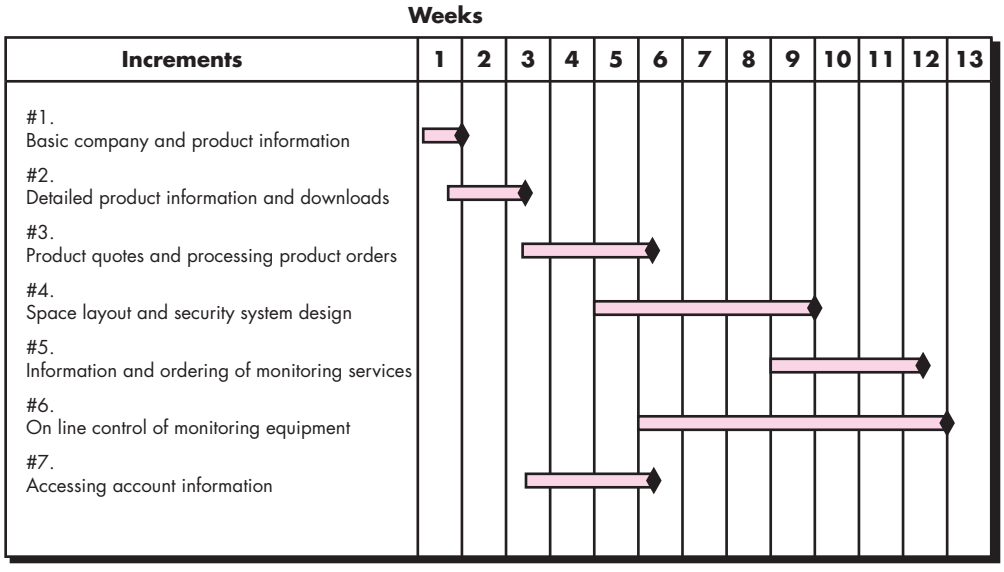
Increment 1: Basic company and product information

Increment 2: Detailed product information and downloads

Increment 3: Product quotes and processing product orders

Increment 4: Space layout and security system design

FIGURE 27.5 Time line for macroscopic project schedule



Increment 5: Information and ordering of monitoring services

Increment 6: Online control of monitoring equipment

Increment 7: Accessing account information

The team consults and negotiates with stakeholders and develops a *preliminary* deployment schedule for all seven increments. A time-line chart for this schedule is illustrated in Figure 27.5.

It is important to note that the deployment dates (represented by diamonds on the time-line chart) are preliminary and may change as more detailed scheduling of the increments occurs. However, this macroscopic schedule provides management with an indication of when content and functionality will be available and when the entire project will be completed. As a preliminary estimate, the team will work to deploy all increments with a 12-week time line. It's also worth noting that some of the increments will be developed in parallel (e.g., increments 3, 4, 6 and 7). This assumes that the team will have sufficient people to do this parallel work.

Once the macroscopic schedule has been developed, the team is ready to schedule work tasks for a specific increment. To accomplish this, you can use a generic process framework that is applicable for all WebApp increments. A *task list* is created by using the generic tasks derived as part of the framework as a starting point and then adapting these by considering the content and functions to be derived for a specific WebApp increment.

Each framework action (and its related tasks) can be adapted in one of four ways: (1) a task is applied as is, (2) a task is eliminated because it is not necessary for the

increment, (3) a new (custom) task is added, and (4) a task is refined (elaborated) into a number of named subtasks that each becomes part of the schedule.

To illustrate, consider a generic *design modeling* action for WebApps that can be accomplished by applying some or all of the following tasks:

- Design the aesthetic for the WebApp.
- Design the interface.
- Design the navigation scheme.
- Design the WebApp architecture.
- Design the content and the structure that supports it.
- Design functional components.
- Design appropriate security and privacy mechanisms.
- Review the design.

As an example, consider the generic task *Design the Interface* as it is applied to the fourth increment of **SafeHomeAssured.com**. Recall that the fourth increment implements the content and function for describing the living or business space to be secured by the *SafeHome* security system. Referring to Figure 27.5, the fourth increment commences at the beginning of the fifth week and terminates at the end of the ninth week.

There is little question that the *Design the Interface* task must be conducted. The team recognizes that the interface design is pivotal to the success of the increment and decides to refine (elaborate) the task. The following subtasks are derived for the *Design the Interface* task for the fourth increment:

- Develop a sketch of the page layout for the space design page.
- Review layout with stakeholders.
- Design space layout navigation mechanisms.
- Design "drawing board" layout.[10]
- Develop procedural details for the graphical wall layout function.
- Develop procedural details for the wall length computation and display function.
- Develop procedural details for the graphical window layout function.
- Develop procedural details for the graphical door layout function.
- Design mechanisms for selecting security system components (sensors, cameras, microphones, etc.).

10 At this stage, the team envisions creating the space by literally drawing the walls, windows, and doors using graphical functions. Wall lines will "snap" onto grip points. Dimensions of the wall will be displayed automatically. Windows and doors will be positioned graphically. The end user can also select specific sensors, cameras, etc., and position them once the space has been defined.

- Develop procedural details for the graphical layout of security system components.
- Conduct pair walkthroughs as required.

These tasks become part of the increment schedule for the fourth WebApp increment and are allocated over the increment development schedule. They can be input to scheduling software and used for tracking and control.

SAFEHOME

Tracking the Schedule

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

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

The conversation:

Doug (glancing at a PowerPoint slide): The schedule for the first *SafeHome* increment seems reasonable, but we're going to have trouble tracking progress.

Vinod (a concerned look on his face): Why? We have tasks scheduled on a daily basis, plenty of work products, and we've been sure that we're not overallocating resources.

Doug: All good, but how do we know when the requirements model for the first increment is complete?

Jamie: Things are iterative, so that's difficult.

Doug: I understand that, but . . . well, for instance, take "analysis classes defined." You indicated that as a milestone.

Vinod: We have.

Doug: Who makes that determination?

Jamie (aggravated): They're done when they're done.

Doug: That's not good enough, Jamie. We have to schedule TRs [technical reviews, Chapter 15], and you haven't done that. The successful completion of a review on the analysis model, for instance, is a reasonable milestone. Understand?

Jamie (frowning): Okay, back to the drawing board.

Doug: It shouldn't take more than an hour to make the corrections . . . everyone else can get started now.

27.6 EARNED VALUE ANALYSIS

Earned value provides a quantitative indication of progress.

In Section 27.5, I discussed a number of qualitative approaches to project tracking. Each provides the project manager with an indication of progress, but an assessment of the information provided is somewhat subjective. It is reasonable to ask whether there is a quantitative technique for assessing progress as the software team progresses through the work tasks allocated to the project schedule. In fact, a technique for performing quantitative analysis of progress does exist. It is called *earned value analysis* (EVA). Humphrey [Hum95] discusses earned value in the following manner:

> The earned value system provides a common value scale for every [software project] task, regardless of the type of work being performed. The total hours to do the whole project are estimated, and every task is given an earned value based on its estimated percentage of the total.

Stated even more simply, earned value is a measure of progress. It enables you to assess the "percent of completeness" of a project using quantitative analysis rather than rely on a gut feeling. In fact, Fleming and Koppleman [Fle98] argue that earned value analysis "provides accurate and reliable readings of performance from as early as 15 percent into the project." To determine the earned value, the following steps are performed:

? How do I compute earned value and use it to assess progress?

1. The *budgeted cost of work scheduled* (BCWS) is determined for each work task represented in the schedule. During estimation, the work (in person-hours or person-days) of each software engineering task is planned. Hence, $BCWS_i$ is the effort planned for work task *i*. To determine progress at a given point along the project schedule, the value of BCWS is the sum of the $BCWS_i$ values for all work tasks that should have been completed by that point in time on the project schedule.
2. The BCWS values for all work tasks are summed to derive the *budget at completion* (BAC). Hence,

   $BAC = \Sigma (BCWS_k)$ for all tasks k
3. Next, the value for *budgeted cost of work performed* (BCWP) is computed. The value for BCWP is the sum of the BCWS values for all work tasks that have actually been completed by a point in time on the project schedule.

Wilkens [Wil99] notes that "the distinction between the BCWS and the BCWP is that the former represents the budget of the activities that were planned to be completed and the latter represents the budget of the activities that actually were completed." Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:

$$\text{Schedule performance index, SPI} = \frac{\text{BCWP}}{\text{BCWS}}$$

$$\text{Schedule variance, SV} = \text{BCWP} - \text{BCWS}$$

WebRef

A wide array of earned value analysis resources can be found at **www.acq.osd.mil/pm/**.

SPI is an indication of the efficiency with which the project is utilizing scheduled resources. An SPI value close to 1.0 indicates efficient execution of the project schedule. SV is simply an absolute indication of variance from the planned schedule.

$$\text{Percent scheduled for completion} = \frac{\text{BCWS}}{\text{BAC}}$$

provides an indication of the percentage of work that should have been completed by time *t*.

$$\text{Percent complete} = \frac{\text{BCWP}}{\text{BAC}}$$

provides a quantitative indication of the percent of completeness of the project at a given point in time *t*.

It is also possible to compute the *actual cost of work performed* (ACWP). The value for ACWP is the sum of the effort actually expended on work tasks that have

been completed by a point in time on the project schedule. It is then possible to compute

$$\text{Cost performance index, CPI} = \frac{\text{BCWP}}{\text{ACWP}}$$

$$\text{Cost variance, CV} = \text{BCWP} - \text{ACWP}$$

A CPI value close to 1.0 provides a strong indication that the project is within its defined budget. CV is an absolute indication of cost savings (against planned costs) or shortfall at a particular stage of a project.

Like over-the-horizon radar, earned value analysis illuminates scheduling difficulties before they might otherwise be apparent. This enables you to take corrective action before a project crisis develops.

27.7 Summary

Scheduling is the culmination of a planning activity that is a primary component of software project management. When combined with estimation methods and risk analysis, scheduling establishes a road map for the project manager.

Scheduling begins with process decomposition. The characteristics of the project are used to adapt an appropriate task set for the work to be done. A task network depicts each engineering task, its dependency on other tasks, and its projected duration. The task network is used to compute the critical path, a time-line chart, and a variety of project information. Using the schedule as a guide, you can track and control each step in the software process.

Problems and Points to Ponder

27.1. "Unreasonable" deadlines are a fact of life in the software business. How should you proceed if you're faced with one?

27.2. What is the difference between a macroscopic schedule and a detailed schedule? Is it possible to manage a project if only a macroscopic schedule is developed? Why?

27.3. Is there ever a case where a software project milestone is not tied to a review? If so, provide one or more examples.

27.4. "Communication overhead" can occur when multiple people work on a software project. The time spent communicating with others reduces individual productively (LOC/month), and the result can be less productivity for the team. Illustrate (quantitatively) how engineers who are well versed in good software engineering practices and use technical reviews can increase the production rate of a team (when compared to the sum of individual production rates). Hint: You can assume that reviews reduce rework and that rework can account for 20 to 40 percent of a person's time.

27.5. Although adding people to a late software project can make it later, there are circumstances in which this is not true. Describe them.

27.6. The relationship between people and time is highly nonlinear. Using Putnam's software equation (described in Section 27.2.2), develop a table that relates number of people to project

duration for a software project requiring 50,000 LOC and 15 person-years of effort (the productivity parameter is 5000 and $B = 0.37$). Assume that the software must be delivered in 24 months plus or minus 12 months.

27.7. Assume that you have been contracted by a university to develop an online course registration system (OLCRS). First, act as the customer (if you're a student, that should be easy!) and specify the characteristics of a good system. (Alternatively, your instructor will provide you with a set of preliminary requirements for the system.) Using the estimation methods discussed in Chapter 26, develop an effort and duration estimate for OLCRS. Suggest how you would:

a. Define parallel work activities during the OLCRS project.

b. Distribute effort throughout the project.

c. Establish milestones for the project.

27.8. Select an appropriate task set for the OLCRS project.

27.9. Define a task network for OLCRS described in Problem 27.7, or alternatively, for another software project that interests you. Be sure to show tasks and milestones and to attach effort and duration estimates to each task. If possible, use an automated scheduling tool to perform this work.

27.10. If an automated scheduling tool is available, determine the critical path for the network defined in Problem 27.9.

27.11. Using a scheduling tool (if available) or paper and pencil (if necessary), develop a timeline chart for the OLCRS project.

27.12. Assume you are a software project manager and that you've been asked to compute earned value statistics for a small software project. The project has 56 planned work tasks that are estimated to require 582 person-days to complete. At the time that you've been asked to do the earned value analysis, 12 tasks have been completed. However the project schedule indicates that 15 tasks should have been completed. The following scheduling data (in person-days) are available:

Task	Planned Effort	Actual Effort
1	12.0	12.5
2	15.0	11.0
3	13.0	17.0
4	8.0	9.5
5	9.5	9.0
6	18.0	19.0
7	10.0	10.0
8	4.0	4.5
9	12.0	10.0
10	6.0	6.5
11	5.0	4.0
12	14.0	14.5
13	16.0	—
14	6.0	—
15	8.0	—

Compute the SPI, schedule variance, percent scheduled for completion, percent complete, CPI, and cost variance for the project.

Further Readings and Information Sources

Virtually every book written on software project management contains a discussion of scheduling. Wysoki (*Effective Project Management,* Wiley, 2006), Lewis (*Project Planning Scheduling and Control,* 4th ed., McGraw-Hill, 2006), Luckey and Phillips (*Software Project Management for Dummies,* For Dummies, 2006), Kerzner (*Project Management: A Systems Approach to Planning, Scheduling, and Controlling,* 9th ed., Wiley, 2005), Hughes (*Software Project Management,* McGraw-Hill, 2005), The Project Management Institute (*PMBOK Guide,* 3d ed., PMI, 2004), Lewin (*Better Software Project Management,* Wiley, 2001), and Bennatan (*On Time, Within Budget: Software Project Management Practices and Techniques,* 3d ed., Wiley, 2000) contain worthwhile discussions of the subject. Although application specific, Harris (*Planning and Scheduling Using Microsoft Office Project 2007,* Eastwood Harris Pty Ltd., 2007) provides a useful discussion of how scheduling tools can be used to successfully track and control a software project.

Fleming and Koppelman (*Earned Value Project Management,* 3d ed., Project Management Institute Publications, 2006), Budd (*A Practical Guide to Earned Value Project Management,* Management Concepts, 2005), and Webb and Wake (*Using Earned Value: A Project Manager's Guide,* Ashgate Publishing, 2003) discuss the use of earned value techniques for project planning, tracking, and control in considerable detail.

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