```
    I.1.3.1   FTR: Review output and input data objects derived in task I.1.2;
    I.1.3.2   Derive a model of functions/behaviors;
              case of: mechanics
              mechanics = quality function deployment
                       meet with customer to review major concept requirements;
                       interview end-users;
                       observe current approach to problem, current process;
                       develop a hierarchical outline of functions/behaviors;
              mechanics = structured analysis
                       derive a context level data flow diagram;
                       refine the data flow diagram to provide more detail;
                       write processing narratives for functions at lowest level of refinement;
              mechanics = object view
                       define operations/methods that are relevant for each class;
              endcase
    I.1.3.3   FTR: Review functions/behaviors with customer and revise as required;
    endtask Task I.1.3
I.1.4       Isolate those elements of the technology to be implemented in software;
I.1.5       Research availability of existing software;
I.1.6       Define technical feasibility;
I.1.7       Make quick estimate of size;
I.1.8       Create a Scope Definition;
endTask definition:   Task I.1
```

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

7.6 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 tasks. Figure 7.3 shows a schematic task network for a concept development project.

The concurrent nature of software engineering activities leads to a number of important scheduling requirements. Because parallel tasks occur asynchronously, the planner must determine intertask dependencies to ensure continuous progress toward completion. In addition, the project manager should be aware of those tasks that lie on the critical path. That is, tasks that must be completed on schedule if the project

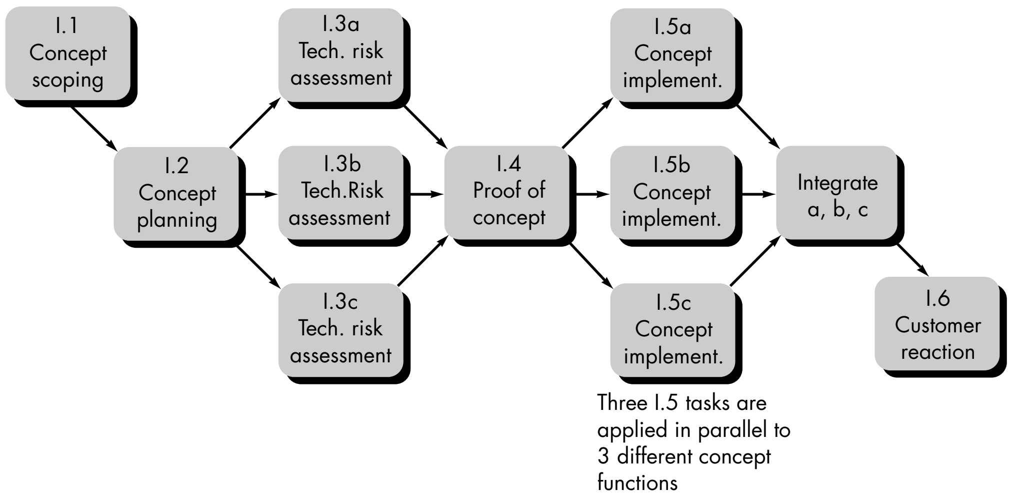

FIGURE 7.3 A task network for concept development

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 7.3 is macroscopic. In a detailed task network (a precursor to a detailed schedule), each activity shown in Figure 7.3 would be expanded. For example, Task I.1 would be expanded to show all tasks detailed in the refinement of Tasks I.1 shown in Section 7.5.

7.7 SCHEDULING

ADVICE

For all but the simplest projects, scheduling should be done with the aid of a project scheduling tool.

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 to software projects.

Program evaluation and review technique (PERT) and *critical path method* (CPM) [MOD83] 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
- Decomposition of tasks

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 the software planner 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.

Boundary time calculations can be very useful in software project scheduling. Slippage in the design of one function, for example, can retard further development of other functions. Riggs [RIG81] describes important boundary times that may be discerned from a PERT or CPM network: (1) the earliest time that a task can begin when all preceding tasks are completed in the shortest possible time, (2) the latest time for task initiation before the minimum project completion time is delayed, (3) the earliest finish—the sum of the earliest start and the task duration, (4) the latest finish—the latest start time added to task duration, and (5) the *total float*—the amount of surplus time or leeway allowed in scheduling tasks so that the network critical path is maintained on schedule. Boundary time calculations lead to a determination of critical path and provide the manager with a quantitative method for evaluating progress as tasks are completed.

CASE tools project/scheduling and planning

Both PERT and CPM have been implemented in a wide variety of automated tools that are available for the personal computer [THE93]. Such tools are easy to use and make the scheduling methods described previously available to every software project manager.

7.7.1 Timeline Charts

When creating a software project schedule, the planner begins with a set of tasks (the work breakdown structure). If automated tools are used, the work breakdown is input as 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 timeline chart enables you to determine what tasks will be conducted at a given point in time.

As a consequence of this input, a *timeline chart,* also called a *Gantt chart,* is generated. A timeline 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 7.4 illustrates the format of a timeline chart. It depicts a part of a software project schedule that emphasizes the concept scoping task (Section 7.5) for a new 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 timeline 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 7.5). Used in conjunction with the timeline chart, project tables enable the project manager to track progress.

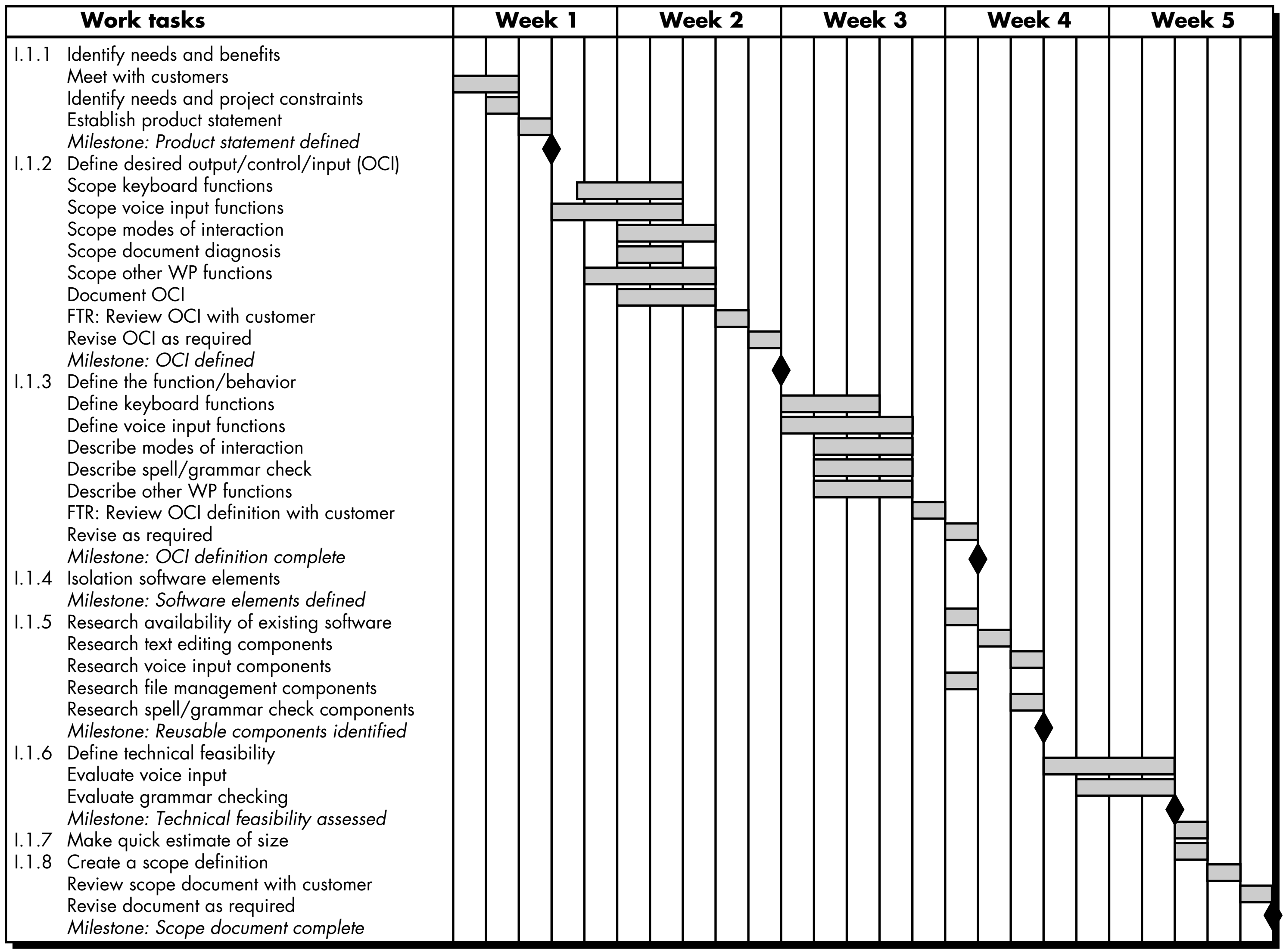

FIGURE 7.4 An example timeline chart

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							

FIGURE 7.5 An example project table

7.7.2 Tracking the Schedule

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

Quote:

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

Capers Jones

- 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 7.4) have been accomplished by the scheduled date.
- Comparing actual start-date to planned start-date for each project task listed in the resource table (Figure 7.5).
- Meeting informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon.
- Using earned value analysis (Section 7.8) to assess progress quantitatively.

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, the project manager must exercise control to reconcile them as quickly as possible. After a problem has been diagnosed,[10] additional resources may be focused on the problem area: staff may be redeployed or the project schedule can be redefined.

ADVICE

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

When faced with severe deadline pressure, experienced project managers sometimes use a project scheduling and control technique called *time-boxing* [ZAH95]. 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.

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

10 It is important to note that schedule slippage is a symptom of some underlying problem. The role of the project manager is to diagnose the underlying problem and act to correct it.

task has been completed.[11] 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.

7.8 EARNED VALUE ANALYSIS

KEY POINT

Earned value provides a quantitative indication of progress.

In Section 7.7.2, we 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 us 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 to assess progress?

1. The *budgeted cost of work scheduled* (BCWS) is determined for each work task represented in the schedule. During the estimation activity (Chapter 5), 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,

   $$\text{BAC} = \Sigma\ (\text{BCWS}_k) \text{ for all tasks } k$$

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.

11 A cynic might recall the saying: "The first 90 percent of a system takes 90 percent of the time. The last 10 percent of the system takes 90 percent of the time."

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:

Schedule performance index, SPI = BCWP/BCWS

Schedule variance, SV = BCWP – BCWS

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.

WebRef

A wide array of earned value analysis resources (comprehensive bibliography, papers, hotlinks) can be found at **www.acq.osd.mil/pm/**

Percent scheduled for completion = BCWS/BAC

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

Percent complete = BCWP/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

Cost performance index, CPI = BCWP/ACWP

Cost variance, CV = BCWP – 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 the software project manager to take corrective action before a project crisis develops.

7.9 ERROR TRACKING

KEY POINT

Error tracking allows you to compare current work with past efforts and provides a quantitative indication of the quality of the work being conducted.

Throughout the software process, a project team creates work products (e.g., requirements specifications or prototype, design documents, source code). But the team also creates (and hopefully corrects) errors associated with each work product. If error-related measures and resultant metrics are collected over many software projects, a project manager can use these data as a baseline for comparison against error data collected in real time. Error tracking can be used as one means for assessing the status of a current project.

In Chapter 4, the concept of defect removal efficiency was discussed. To review briefly, the software team performs formal technical reviews (and, later, testing) to find and correct errors, *E,* in work products produced during software engineering