Chapter VII

Developing the Schedule and Cost Plan

You got to be careful if you don't know where you're going, because you might not get there.

(Yogi Berra)

The Project Management Institute (PMI) project management process groups include initiation, planning, execution, control, and closing. In practice, however, the initiation processes of a project are often not part of a project for budgeting and control issues, but rather are charged to management and administration (M&A) or operations and maintenance (O&M) general ledger accounts. In some organizations, these charges are later reversed back to a project after it is decided to move forward with that project. Thus, only the planning, execution, and control processes become part of the project for accounting purposes; sometimes detail planning is part of a project but not overall planning. Similarly, the closing process group may or may not be a formal part of the project, and sometimes those processes are performed by an independent organization. This chapter is concerned with detail project planning, particularly the schedule and cost plan.

Detail Project Planning

After the requirements have been determined, documented, and approved, a detail analysis and breakdown of that project's scope are created. According to PMI.

Scope definition involves subdividing the major project deliverables (as identified in the scope statement and requirements document) into smaller more manageable components in order to:

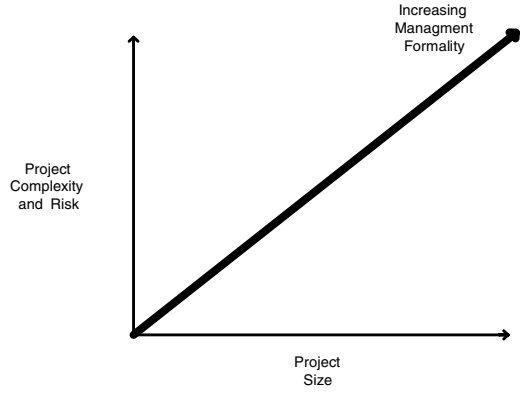
- *Improve the accuracy of estimates*
- *Define a baseline for performance measurements*
- *Facilitate clear responsibility assignments. (PMI, 2000)*

These manageable components are organized into a work breakdown structure (WBS). According to PMI, the major activities (and the order in which they are to be performed) in project planning include the following (PMI, 2000):

- Develop scope statement
- Assemble project team
- Develop work breakdown structure
- Finalize project team
- Build a network diagram (showing activity dependencies)
- Estimate cost and time, and find the "critical path"
- Determine the overall schedule and budget
- Plan procurement
- Plan quality
- Identify risks, quantify them, and develop responses
- Other plans: change control plan, communications plan, and management plan
- Completion of an overall project plan
- Project Plan approval
- "Kickoff Meeting"

For a very simple project, however, one may only need a to-do list and a shopping list. The degree of project management formality has to be adjusted for the size and complexity of a project, as illustrated in Figure 7.1. For a large and/or complex project, all the above activities need to be done; but the order of these activities depends upon the manner in which IT departments estimate project costs and handle resource allocation and costing (particularly human resources).

Figure 7.1. Project planning formality



Developing the Work Breakdown Structure

The concept and name of the WBS goes back to the U.S. Department of Defense (DoD) in the 1950s. The PMI definition of a WBS is "A *deliverable oriented* grouping of project elements that *organizes and defines* the total work scope of the project" (PMI, 2000). PMI is developing a practice standard for the WBS, which was only partially available during the writing of this book.

"The most important part of the planning phase is the development of the work breakdown structure" (Klastorn, 2004). *It is essential to develop a complete and well-organized WBS. Many project managers or coordinators spend too little time in developing the WBS and perhaps too much time in scheduling resources too precisely to WBS tasks. The WBS is the basis for all project planning and later performance monitoring; this is illustrated in Figure 7.2.*

For most projects, the WBS is typically arranged in a multilevel hierarchical manner, a so-called tree structure. The first level (Level zero, or top level) typically corresponds to the project phases. Each level of the WBS is a further breakdown of the previous level. Note that some authors refer to the top level of the WBS as Level 1 and there is only one code (node) at the top level, which represents the project in total. In the IT field, we are accustomed to starting our numbering at zero and consider the top node as our data structure name, not a WBS code (node). Note also that, although the WBS specifies the work to be done, it does not specify the order in which the work is to be done, nor does the WBS numbering indicate the order in which work items are to be done. The WBS numbering per se also does not indicate assignment of work to responsible organizational components, assignment of work to any resources, or the cost of any work item. The WBS can be schematically represented by a tree diagram, either horizontally, as shown in Figure 7.3, or vertically, as shown in Figure 7.4.

Figure 7.2. WBS uses

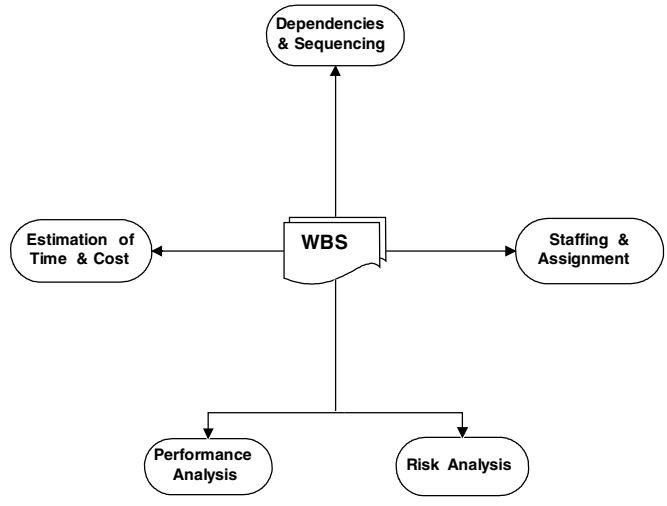


Figure 7.3. Horizontal WBS

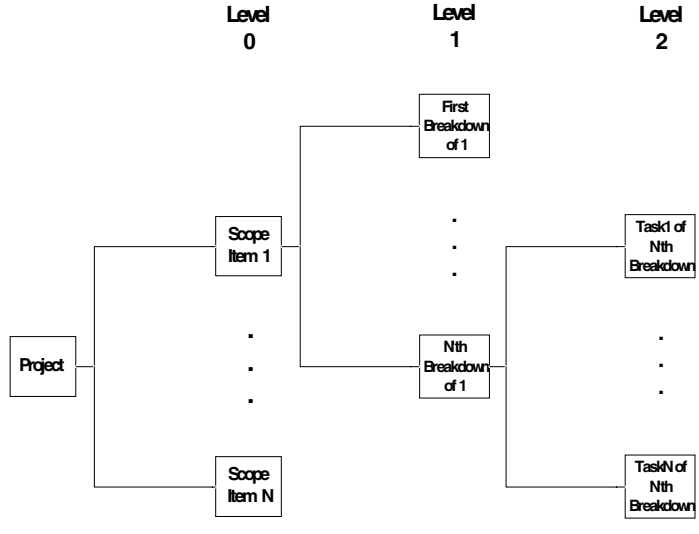
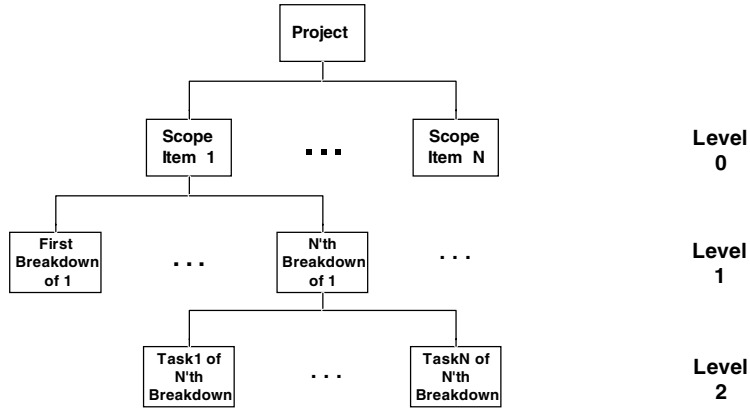


Figure 7.4. Vertical WBS



Text-based schematic representations are typically horizontal with indentation used for each level. Identification numbering of each WBS code typically uses a dot or dash notation. For computer sorting purposes, leading zeros are used so that each level of the WBS identification code uses the same number of digits. For example, with 12 codes at level zero, use "01" through "12", not "1" through "12".

1.0 Scope Item 1

1.1 First Breakdown of Scope Item 1

…

1.N N'th Breakdown of Scope Item 1

1.N.1 First Task of N'th Breakdown of Scope Item 1

…

1.N.N N'th Task of N'th Breakdown of Scope Item 1

2.0 Scope Item 2

…

The lowest level (when viewed as a vertical tree) defines the actual tasks, often called work packets. These work packets are also called activity WBS codes, as opposed to the WBS codes at the higher levels, which are called "control" codes. A control code has one or more lower level subsidiary code, each of which is either another control code or an activity code. Each activity code is associated with one and only one control code. ***Planned costs are posted only to activity accounts, and actual costs may be posted at any level, but the recommended approach is to gather costs at the activity level, also.*** The sum of the costs for a control code's activity codes are rolled up from is subsidiary codes to that control code.

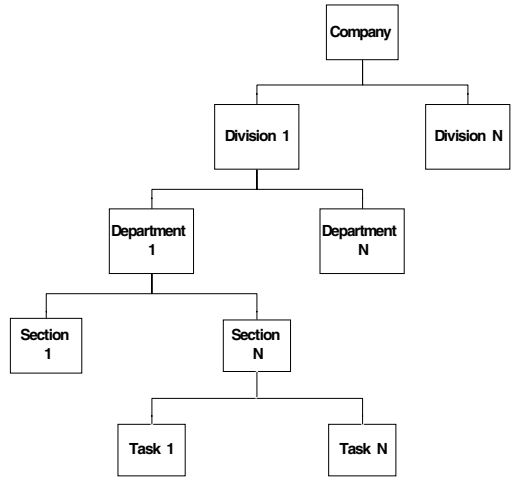
Note that some authors consider the bottom level of the WBS as not a part of the WBS per se, because they consider tasks or activities to be part of the schedule not part of the WBS. In their view, WBS codes are only nouns or adjectives, and the schedule items are verbs. Also, in the early implementation of a WBS for earned value (discussed in a later chapter) by the U.S. DoD, the second lowest level of the WBS was called cost accounts (or control accounts), where each code corresponded to in intersection of the higher WBS level and a performing organization code; below that level of cost accounts were the work packets. In this early method, pricing was done at the cost account level and scheduling (including dependencies between packets) was done at the packet work level. In practice this is not always practical for several reasons including the fact that task dependencies may encompass multiple cost accounts. *In our more realistic and general purpose view, the tasks at the bottom level of the WBS are the activities that later get estimated, priced, assigned resources, and scheduled.*

There are several common methods of logically breaking down the scope of a project.

These methods not only logically divide the overall work, but also are relate to the way costs are accumulated for the project in an organization. Later in this book, we discuss cost gathering methods, constraints, and associated issues.

The organizational method divides the scope based upon the organization doing the work (performing organization), responsible for the work (responsible organization), or benefiting from the work (benefiting organization, usually the customer). This is illustrated in Figure 7.5. For the performing or responsible organization method, costs are gathered by the organization doing or delegating the work through its payroll, job costing, and/or accounts payable system. For the benefiting organization method, costs are usually gathered by a billing and accounts receivable system.

Figure 7.5. Organizational WBS



The product method organizes the scope based upon the logical or physical parts of a product, which is the subject of the project. This is illustrated in Figure 7.6. Here, costs are typically gathered based upon the part being worked upon and may be from a manufacturing/material resources/requirements planning (MRP) system. Because some scope may be related to nonspecific parts (overall management, operation, or services such as engineering design services), a totally product-based WBS is not common. Product-based WBSs are often associated with contracted work, such as with the U.S. government.

The functional (or task-based) WBS is based upon the work being done to carry out the scope of the project, and is illustrated in Figure 7.7. This type of organization is the most effective for measuring project performance as will be shown later in this book. Most organizations have some type of methodology for building the item that is the subject of the project, and the task based WBS usually follows that methodology in its logical layout.

Because none of these pure WBS methods reflect a logical structure by organization, product, or task, a hybrid WBS structure may be used, such as the one shown in Figure 7.8.

To account for the multiple dimensions of organization, product, cost category, and resources, other coding systems may be used for a project in addition to the WBS. These are used for both pure and hybrid types of WBS structures. These coding systems are

- *OBS:* The organizational breakdown structure is used to show organizational units of a performing or responsible organization. There may be an OBS for benefiting organizations also, but there is generally only one benefiting organization for the entire project (and the overall project code may include the benefiting organization code). Typically each task at the lowest level of the WBS (work packet) is associated with one OBS code for the responsible or performing organization.

Figure 7.6. Product WBS

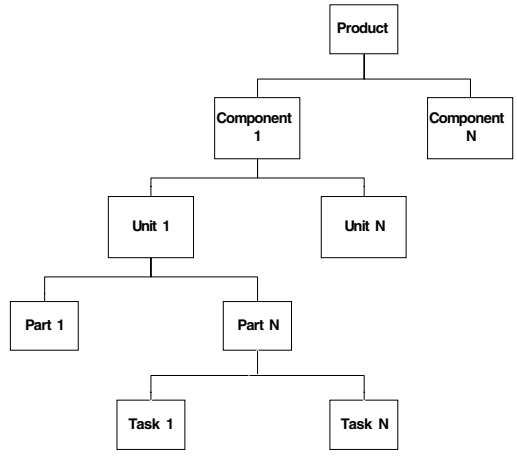


Figure 7.7. Functional WBS

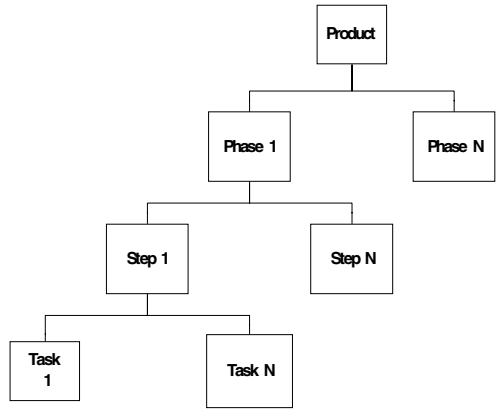
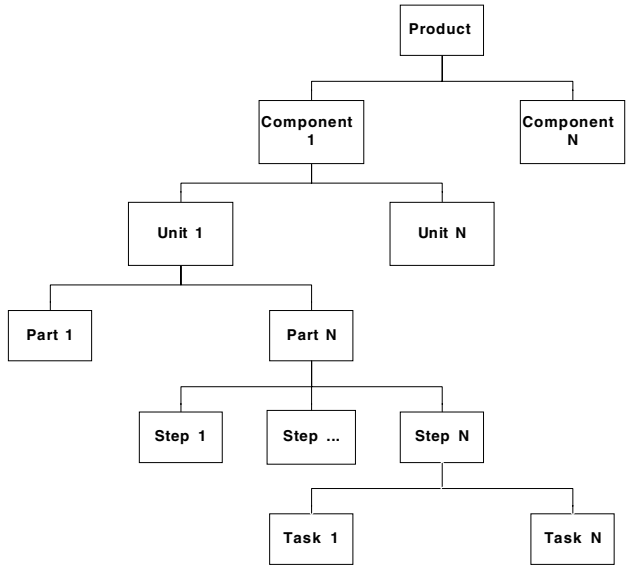


Figure 7.8. Hybrid WBS



- *PBS:* The *part breakdown structure* shows the physical breakdown of parts. A PBS may be the "top" of a hybrid WBS, which has a task-based, lower level WBS. Alternatively, a PBS may be used as a code that is an attribute of some of the WBS codes (those that are specifically associated with one part).
- *RBS:* The *resource breakdown structure* is used to identify resources that are to be used on a project such as individuals (or labor categories). Typically each work packet is associated with one or more RBS codes. This is discussed later in this chapter.

- **CBS:** The *contractor breakdown structure* identifies outside (outside of the company or responsible organization) responsibilities for work or other services and may be used in a similar manner to the OBS (or the OBS may accommodate internal and external organizations). CBS codes are also commonly used for WBS codes at levels above the work packets to show outside delegation.
- **EOC:** The element of cost structure identifies the type of cost incurred or to be incurred such as labor, lease, travel, space allocations and other charges to a project.
- **GL:** The *general ledger* code is an accounting code to which charges are applied in the company's financial system. Typically, the GL code is determined (calculation or table lookup) automatically from other codes (typically the overall project code, the OBS, and the EOC).

In some organizations (i.e., the U.S. government) it is common to develop an organizational assignment matrix, which maps the WBS (down to the lowest level, excluding the work packets/activities) to the OBS. This may be called a responsibility assignment matrix (RAM), but that term is also commonly used for a mapping between the WBS and the RBS. Figure 7.9 shows a conceptual example of this matrix in a spreadsheet. The cells of the spreadsheet are typically the work packets (or a small set of interdependent work packets) and are called *cost accounts*, a term that originally came from the U.S. DoD Cost/Schedule Control Systems Criteria (C/SCSC), discussed in Chapter XIV.

The task or functional WBS is best for project performance measurement and control because the WBS is organized by the actual work to be done. This more clearly illustrated in Chapter XIV. The best approach for IT projects is to base the WBS tasks on the company's adopted methodology for software development, software acquisition, and/or software integration. Thus, an IT WBS is best developed from the project scope using adopted methodology with consideration for how the work will be allocated to organi-

Figure 7.9. Organizational assignment matrix

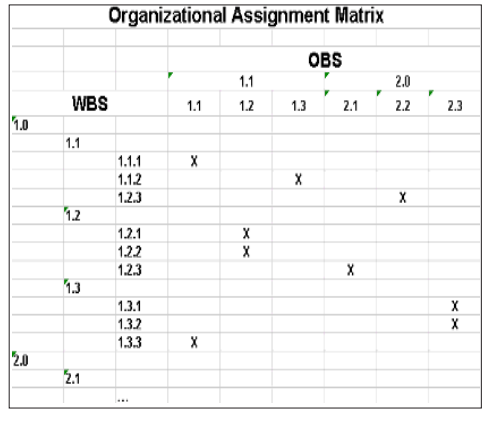
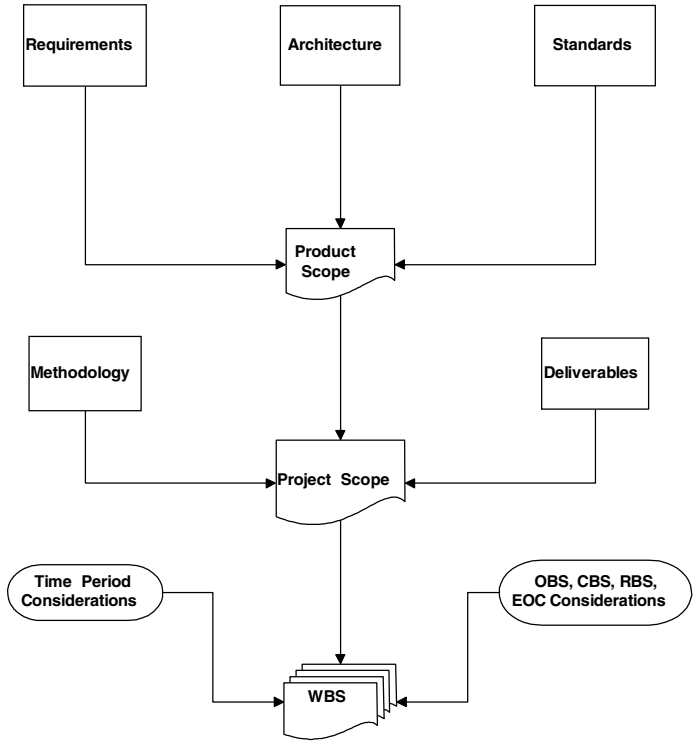


Figure 7.10. WBS development



zational components and further broken down into manageable pieces. This is conceptualized in Figure 7.10.

In using a methodology and task-based structure, it is common to use a PBS to create a hybrid WBS. The PBS for IT projects usually corresponds to software and/or hardware deliverables. There are two ways to structure such a hybrid WBS. The first is the methodology first method, as shown in Figure 7.11, and the second is the deliverable first method, as shown in Figure 7.12. The deliverables shown in these diagrams may consist of several levels, such as a level for major subsystems and a second level for the components of each subsystem (screens, reports, etc.).

The lowest level of the WBS represents the tasks to be done (work packages or activity codes). "The work package usually has a short-time span schedule (40 to 100 hours), is limited to one performing department, and has defined completion criteria" (Kiewel, 1998). *The size and nature of these work packets is very important in regard to project cost planning and control*. It is very important that they

- Are realistically and confidently estimable
- Cannot be (or should not be) logically subdivided

Figure 7.11. Methodology first WBS

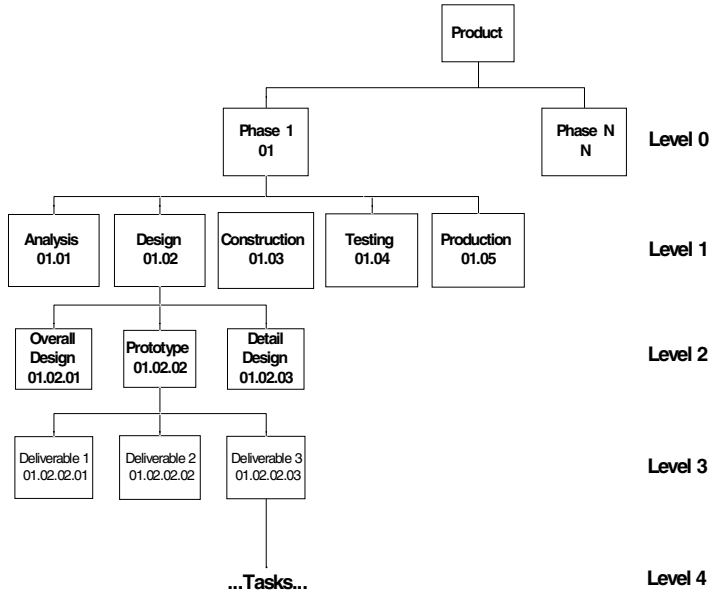
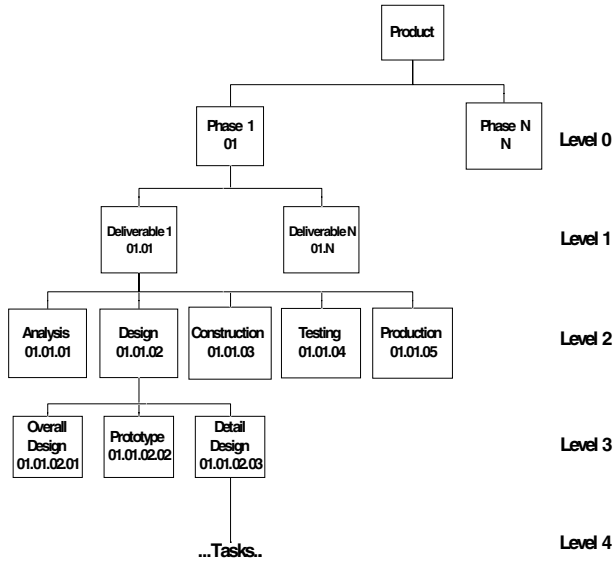


Figure 7.12. Deliverable first WBS

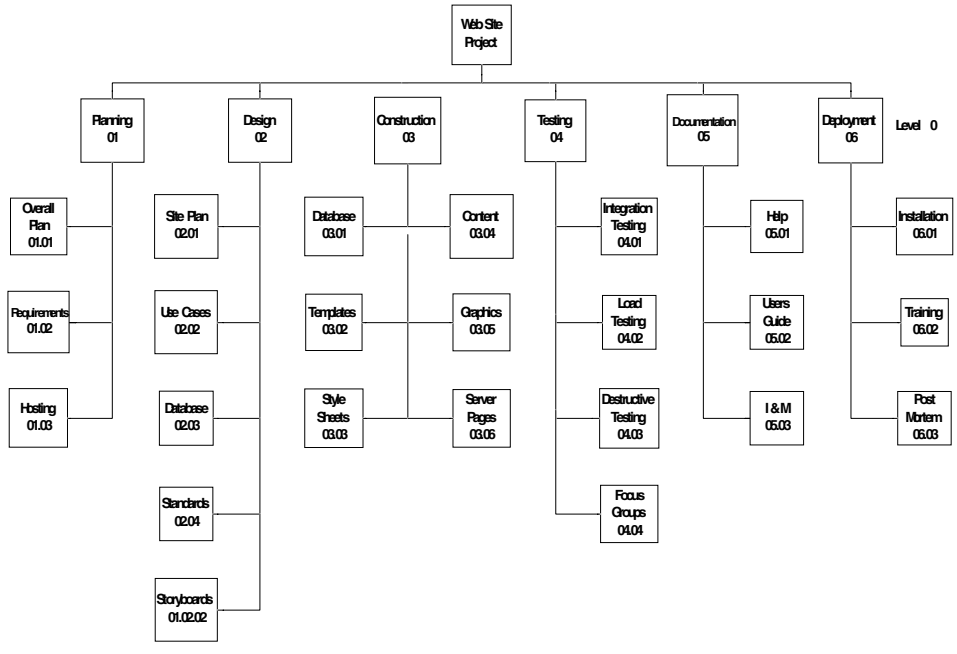


- Are to be assigned to one OBS/CBS code
- Can be completed relatively quickly (80 hours max rule of thumb)
- Have a meaningful and measurable conclusion and deliverable
- Have enough "visibility" so that progress and cost can be determined
- Can be completed without interruption (i.e., need for more info)

Each work packet should be assigned to one organization that is responsible for the work to be completed, so the packet would be associated with a single OBS or CBS code. If possible, it is also best to have each work packet assigned to a single resource (RBS code), although this may not be possible in some IT organizations where techniques such as pair programming are used. Different types of work should not be lumped into a work package; for example, design and coding should be in separate packages, particularly for estimation purposes, discussed later. A time period in a project calendar corresponds to the interval that is to be used for gathering project performance data, including hours expended and percent complete estimates. It is important that a work packet not encompass many such periods; the optimal situation would be for a work packet to correspond exactly to one time period. For performance analysis, progress estimates (% complete) and cost data should be available at each time period for each work packet (incurred effort and cost during that period). Work packets that are highly interdependent should be combined (if such is possible without extending the time of that packet beyond a few time periods) to reduce overall work packed interdependencies for scheduling. A *Work Packet Decomposition Checklist* was presented by Raz and Globerson (1998):

- Is there a need to improve the accuracy of the cost and duration estimates?
- Is more than one individual responsible for the work?
- Does the work content include more than one type of activity?
- Is there a need to know precisely the timing of activities internal to the work package?
- Is there a need to cost out activities internal to the work package?
- Are there any dependencies between the internal activities of the work package and other work packages?
- Are there any significant time breaks in the execution of the internal activities?
- Do resource requirements within the work package change over time?
- Do the prerequisites differ among the internal activities?
- Are there acceptance criteria applicable before the completion of the entire package?
- Are there any intermediate deliverables that can be used to generate a positive cash flow?
- Are there specific risks that require focused attention?

Figure 7.13. Web site development WBS



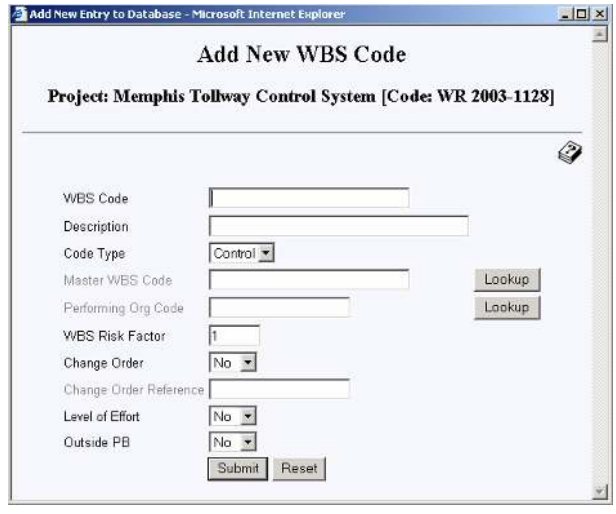
In addition to the formulation of the WBS itself, a WBS dictionary is often created by the project team and is typically used to control what work is done and when. It puts a clear boundary around each work packet:

- *Work Packet ID Info:* Code, name, element of cost
- *Assignments:* Performing organization, resource assigned to, customer contact
- *Time Information:* Start, length, end
- *Dependencies* (dependent WBS codes)
- *Detail Description* (both product/service and work)
- *Acceptance:* Quality criteria, testing required, approval by

As an example of IT WBS formulation, consider the case of a project for the development of a corporate Web site. A WBS based upon a typical software development methodology may appear as in Figure 7.13, which shows the first two levels (0 and 1) of the WBS developed for this project. Each Level 1 item expands into work packets at Level 2.

Software packages used in project management are discussed in Chapter XV, but a typical form to add a WBS code to a project appears in Figure 7.14 (from the Web-based

Figure 7.14. Form to add WBS code



FiveAndDime system). Figure 7.15 shows a screen capture from a display of the Level zero WBS codes. Each Level 1 code expands into the Level 2 codes, which are the work packages for this project. For example, the Level 1 WBS code for Standards is shown expanded to the bottom level in Figure 7.16. Figure 7.17 is an example of another form; this one is used for quicker entry of subsidiary WBS codes, which by default inherit the OBS code of their master WBS code.

Task Estimation

After a WBS is formed, the next step is to estimate the amount of time it will take to perform each of the tasks at the lowest level of the WBS (work packets). Task estimation is still a problem area for IT projects. In a 2004 *Computerworld* survey, the question was asked in about 150 companies: "Do your project managers have the appropriate skills to develop accurate estimates?" The results were

- Sometimes: 47%
- Rarely: 27%
- Often: 19%
- Most often: 6%
- Routinely: 1% (Brandel, 2004)

Figure 7.15. Table of Level 0 WBS codes

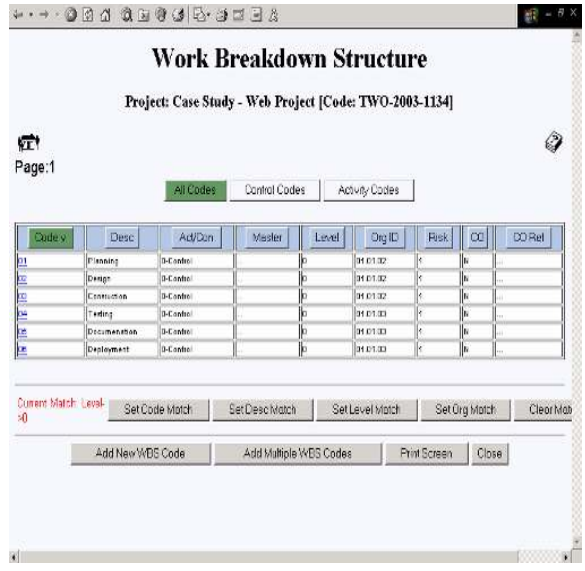


Figure 7.16. Table of Level 1 WBS codes

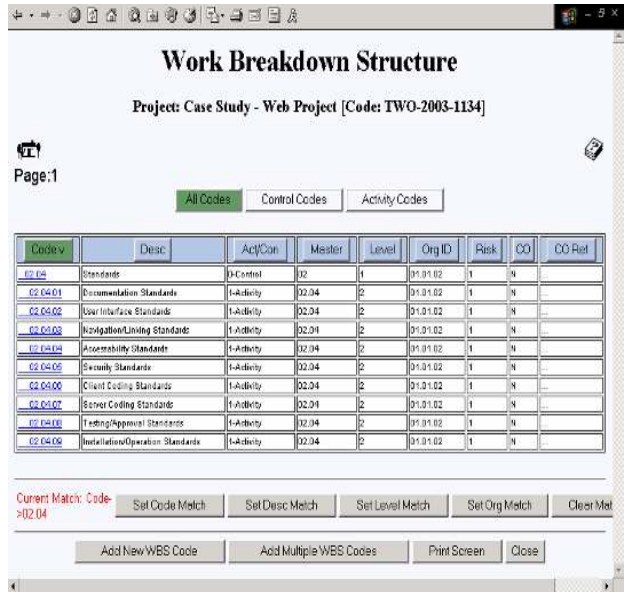
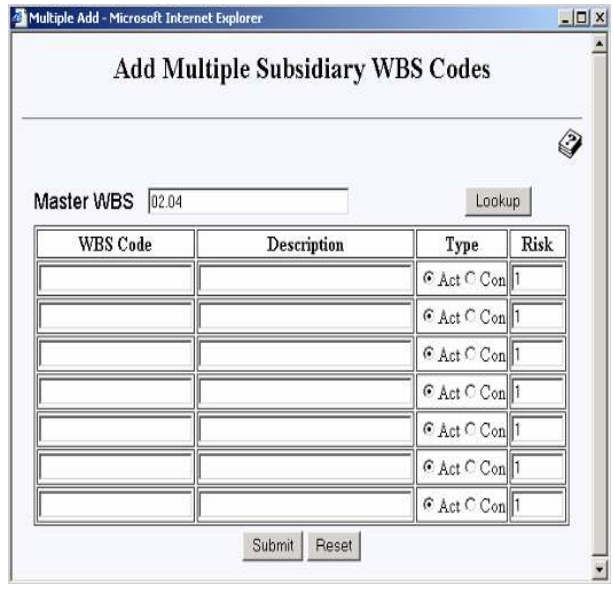


Figure 7.17. Form to add subsidiary WBS codes



In many projects, particularly IT projects, the type of resource(s) needed for each of these tasks must be determined to estimate time and cost. Resource needs are usually specified generically at this point in the project, such as programmer/analyst Level 3 (PA3) or technical writer Level 2 (TW2). If the organization's resource breakdown structure is generically oriented, then a RBS(s) code may be assigned to the WBS codes at this point. Figure 7.18 is an example of a generic resources table that shows the "burdened" rate for each type of resource. The amount of time needed to complete a work packet should be a function of the type of resource and number of resources assigned; for example, we would expect that a PA3 would typically complete a programming task quicker that an PA2. Also, certain types of tasks may have "levels of difficulty" that could only be assigned to a resource above a certain level.

In many IT organizations, however, promotions from one grade level to another are based more upon "time in grade" than upon competency level, so there may not be much actual average productivity increase in moving up the grade scale. For that reason and others, in IT, the productivity rate varies widely between individuals, even in the same grade. This is a phenomenon that is perhaps peculiar to IT and a few other professional areas, such as engineering. As a contrasting example, in the construction area, most bricklayers have a similar productivity rate in terms of bricks per hour. Some IT organizations may specify the actual resources (i.e., by person) at this point in the project (or they may specify a "preferred individual" for a task), but typically, specific resource assignments come later in the project.

For IT projects, most packets involve only labor and the amount of work is usually estimated by using historical data, benchmarks, and/or some specific parametric tech-

Figure 7.18. Generic project resources

Generic Resources		
Code	Description	Burdened Rate
P1	Programmer 1	30
P2	Programmer 2	35
P3	Programmer 3	40
PA1	Programmer Analyst 1	45
PA2	Programmer Analyst 1	50
PA3	Programmer Analyst 1	55
SA1	Systems Architech 1	60
SA2	Systems Architech 2	65
SA3	Systems Architech 3	70
DB1	Database Analsys 1	50
DB2	Database Analsys 2	55
DB3	Database Analsys 3	60
TW1	Technical Writer 1	30
TW2	Technical Writer 2	40
SEC1	Security Specialist 1	40
SEC2	Security Specialist 2	50
TEST1	Test Specialist 1	30
TEST2	Test Specialist 2	40
PM1	Project Manager 1	60
PM2	Project Manager 2	70
PM3	Project Manager 3	80

nique such as lines of code, function point analysis, or number and type of requirements involved. Even if a parametric estimation technique or benchmarking is used, historical data (either internal or external to the organization) should be used for a reality check. Historical estimates for the effort required in the work packets usually come from informal estimates by individuals experienced with the specific type of work. If possible, estimates from the actual people who would be doing the work may prove the best, and it also provides a type of buy in from those individuals regarding the validity of the project estimates. A formal process such as the Delphi technique can also be used to gain better understanding and consensus among multiple experts on the effort required in IT tasks (Roetzheim & Beasley, 1998). This is an iterative technique and can take longer than informal methods, but usually produces better results.

Another form of estimation is analogous estimating or top down estimating. This technique uses prior similar projects, and then estimates the overall cost of the entire current project. Eestimates are then formed from that overall estimate for the first level of the WBS (Level zero) using a percentage breakdown from prior project experience. Then costs are broken down to the next lower level of the WBS, and so on. This technique is often used during project conception and selection but fortunately not typically used for IT projects for detail cost and time estimating. When this technique is used by upper management or PMs to set overall project cost, it can often result in unrealistic budgets and schedules for a project that will ultimately doom the project and/or demoralize the IT staff such that future projects will be at risk. PMs do not have a reputation for making accurate estimates of IT work by themselves. This may be true even if a PM came up through the technical ranks, since IT technology changes so quickly.

One or more benchmarks of task level work may be available from previous, similar projects, or actual benchmarks may be used. Often, prototyping and benchmarking efforts can be combined. These benchmarks may be used in a linear manner to estimate

time for the tasks at hand. For example, we may know that a Web screen form with four fields took 2 hours to build and test, and that another Web form with eight fields took 3 hours to build and test. A linear relationship (two equations in two unknowns) would then predict the time to build and unit test such Web forms would be 1 hour plus the number of fields divided by 2. If there are many such Web forms to build, a "learning curve" relationship could be added, which is usually in a form as:

$$T(i) = T(1)*(a + (1-a)i^{-b}),$$

where $T(i)$ is the time to build the i'th form, $T(1)$ is the time to build the first form, b is the learning rate which is less than 1 (typically in the range of .01 to .1), and a is the incompressibility factor also less than 1 (typically about .5). Actual values of a and b can be calculated by how long it would take a programmer to build the first such form and how long it takes him to build a second and third such form. These values are then substituted in the formula (which is made linear via log functions) to determine the learning factor and the compressibility factor.

In practice, however, this scenario is oversimplified and there is usually more that one factor that influences the time to build an IT component. In the Web form example, each form field may have a different degree of complexity. For example, some fields may require validation, and they may require a new type of validation that is not already in our JavaScript validation library. Some fields may require simple lookups (i.e., drop downs) or more complex database lookup and/or validation, such as foreign key fields. In general, most parametric effort estimation models are a function of problem size and other adjustment factors (sometimes called "cost drivers").

In the early days of software application development (i.e., the 1960s and 1970s), the standard metric for the size and complexity of a software development task was measured in lines of code (LOC). Back then, business applications were written mostly in COBOL, and engineering/scientific applications were written mostly in FORTRAN. Although, it was always augmentable which lines counted and which lines were not counted (such as declarations and comments), and many such metric-based estimation techniques and databases were developed and found utility. The utility of such methods were the greatest where comparisons were made using the same language in the same platform and programming environment; also, the tasks being measured should be totally programming tasks (not involve testing and documentation or other related tasks).

The constructive cost model (COCOMO) was introduced in 1981 (Boehm, 1981). Based on a number of real software development projects, it presented parametric models correlating LOC to task effort. Using COCOMO, one first classifies the type of project into one of three categories: organic, semidetached, and embedded. Organic projects are relatively routine, embedded projects represents new endeavors for the organization, and semidetached are not new endeavors but are more complex than organic projects. There are also three points of application in a project: basic (applied early in a project), intermediate (after requirements are finalized), and advanced (applied after detail design). Basic COCOMO formulas were

Organic Projects: EFFORT = 2.4 * KLOC $^{1.05}$

Semidetached Projects: EFFORT = 3.0 * KLOC $^{1.12}$

Embedded Projects: EFFORT = 3.6 * KLOC $^{1.2}$

where KLOC is 1 thousand lines of code and EFFORT is expressed in person months. The intermediate model uses 15 cost drivers, including reliability needed, database size, complexity, experience and capability factors, schedule factors, and practices and tools maturity. The advanced model gets more detailed and sophisticated including phasing factors.

As languages multiplied and diversity and as fourth generation languages (4GLs), computer aided software engineering (CASE), and what you see is what you get (WYSIWYG) code generators became more common, these basic correlation metrics became less predictive. Other IT changes (such as a move away from mainframe batch processing to client-server real-time turnaround; more emphasis on software reuse, components, and OO software; and spending more time in design and less time in coding) also required new correlations. Other estimation techniques were developed, and COCOMO was expanded to COCOMO II (or COCOMO 2) for newer IT environments (Boehm, 2000). The original COCOMO model was also redesignated COCOMO 81. COCOMO II consists of three submodels, each one offering increased accuracy based upon the depth of planning/design. In order of increasing accuracy, these submodels are the applications composition, early design, and postarchitecture models.

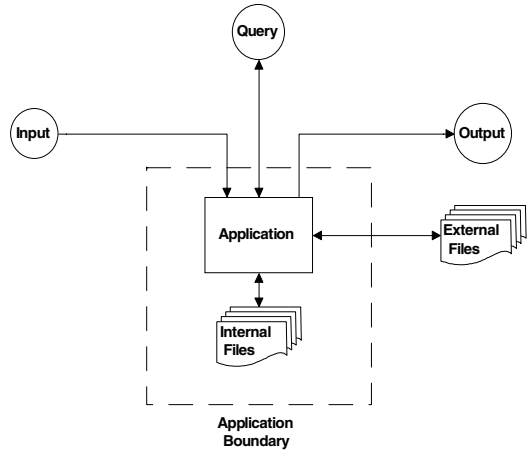
With LOC estimation techniques, one still has to convert an application level component (such as a Web form) into an LOC number. This often can be done by looking at past coding artifacts. The concept of function point analysis (FPA) was proposed by IBM (Albrecht, 1979). This concept starts with application-level components. FPA tends to be relatively independent of language, coding style, and software architecture. A function point is defined as one end-user computational task, such as a query for an input. This notion is important because it allows a function point to be mapped easily into user-oriented requirements and components, but it hides internal functions.

Basic function points are categorized into five groups, or transaction types: files (internal logical file), outputs (external output), inquiries (external inquiry), inputs (external input), and interfaces (external interface). This is illustrated in Figure 7.19. In addition, each type of function point is rated as to it complexity: low, average, and high.

Complexity categories are assigned point scores, and a typical assignment might be *low* = 5, *average* = 10, *high* = 15.

The point score can vary based upon which of the five transaction types is involved. A file is typically a traditional disk file or a relational database table. The file's complexity is low if there are few attributes and it is complex if there are many attributes and groups. An interface is also typically a connection to a file or table, but one that is outside of the domain of this application. An external input provides for data to come into the application. It is typically a user input screen, but it can be any means of gathering input, such as from scanners. The external inputs generally alter the data in the files/tables via add, delete, or update functions; the number of files/tables affected determines the complexity level. An external output provides for data to exit the application boundary

Figure 7.19. Function point types



and is typically a report or an output screen; the number of files/tables from which the data comes determines the complexity. Queries first request and then output data across the application boundary but do not alter the state of the system (data contained in the files/tables); complexity is a function of the number of files/tables involved. Once all the application components are classified and rated for complexity, an unadjusted function point total (UFP) is calculated as shown in the spreadsheet template of Figure 7.20.

The next step in FPA is to calculate a value adjustment factor (VAF) based upon the presence (or lack thereof) of certain general system characteristics. Each of these general system characteristics is assigned a degree of influence from 0 to 5, with zero being *not present* to 5 being a *very strong influence*. These degrees of influence are then summed to get the total influence; this sum is then divided by 100 and added to a base factor (a typical base factor might be 0.65). General characteristics would include distributed data processing, online updates, transaction processing, multiple sites, and so forth. The total function points are then FP = UFP * VAF. For all the details and particulars for different IT environments, please see the standard practices for FPA methodology, which is found in the *IFPUG Counting Practices Manual* (IFPUG, 1996).

FPA can and should be done at several points in an IT project. An initial FPA can be done based on the overall scope statement; a more detailed FPA done later, based on the formal requirements definition; and again later, based on the WBS. FPA can also be done for change orders and to detect scope creep. Without some standardization of how the function points are enumerated and interpreted, consistent results can be difficult to obtain. For example, how are relational database views to be counted—one file/table, or many. Successful application depends on establishing a consistent method of counting function points and keeping records to establish baseline numbers for your specific systems. Training in FPA is available from many sources, and such training is recommended. There are also certification programs.

Figure 7.20. Unadjusted function point total

Unadjusted FPA

	Complexity									
Type	Low			Average			High			Type Total
	# Comp	Scale	Points	# Comp	Scale	Points	# Comp	Scale	Points	
Files		12			16			20		
Inputs		7			12			18		
Outputs		5			10			15		
Queries		5			10			15		
Interfaces		10			15			20		
								Total-->		

Function point methodology has evolved over the years; the International Function Point User Group (IFPUG; www.ifpug.org/), formed in 1986, is the focal point for information on function point analysis (IFPUG, 1996). There is also a FPA user group for the United Kingdom, UFPUG. The original metrics have been considerably updated and refined to cover more IT application areas. Today, FPA has become widely used to estimate a software project's effort and duration, set productivity rates in function points per hour, determine support requirements, and to estimate system change orders. There is a large user community for function points; IFPUG has more than 1,200 member companies, and they cooperate in establishing an effective FPA program.

There are correlation tables to convert between function points and LOC in different IT environments, and this is called "backfiring" (Jones, 1995). Typical approximate values are shown in Figure 7.21.

COCOMO II supports the use of either/both function points or source lines of code. Early in the project, object points are used, which are similar to function points but broader and which basically represent an external object (such as a screen, report, or batch module) and their complexity. Each object generates a number of object points:

Screens – Simple = 1; Medium = 2; Difficult = 3;

Reports – Simple = 2; Medium = 5; Difficult = 8;

Modules – 10

There are guidelines for defining simple, medium, and difficult based upon the number of views used and database files/tables involved. The COCOMO II application composition (rough estimate) formulas using object points (OP) take the form of

EFFORT = NOP / PROD

NOP = OP * (100 - % reuse)/100

Figure 7.21. Function point backfiring

Language	LOC per Function Point
Machine Language	650
Assembly Language	300
Ada	70 - 80
Pascal	80 - 100
C	100 - 150
C++	30 - 50
Visual Basic	30 - 40
Java	25 - 50

PROD is the productivity that varies depending upon the maturity of the organization and the individual(s) involved; it typically ranges from 4 (very low) to 50 (very high). During the early design stage of a project, unadjusted function points are used. Once a specific architecture has been chosen, lines of code are used. The COCOMO II postarchitecture formulas take the form of:

$$\text{EFFORT} = \text{ASLOC}^{(\text{AT}/100)} / \text{ATPROD} + 2.5 * (\text{SIZE} * (1 + \text{BRAK}/100))^{b} \bullet \text{EM}_i$$

$$b = 1.01 + (\text{“SF}_j)/100$$

$$\text{SIZE} = \text{KSLOC} + \text{KASLOC} * \text{AAM} * (100 - \text{AT})/100$$

Where:

AT = % of components adapted

SF_j = scale factor

EM_i = effort multiplier

ASLOC = size of adapted components

BRAK = % code discarded due to requirements volatility

ATPROD = automatic translation productivity

AAM = adaptation adjustment multiplier

KSLOC = thousand lines of source code

KASLOC = thousand lines of adapted source code

There are a number of other estimation methods and formulas including the Software Lifecycle Model (SLIM, or Putnam Model), Price-S (Lockheed Martin Model), Waltson-Felix Model, Bailey-Basil Model, and the Doty Model. For a detailed analysis of these methods, see the DOD report by McGibbon (1997). For example a project of 100,000 lines of source code would be roughly estimated as shown in the following list by some of these models (assuming nominal effort multipliers and scale factors, no discarded code, and no adapted code):

COMOMO II – 500 person months

SLIM – 275 person months

Doty – 650 person months

Walston/Felix – 350 person months

Bailey/Basil – 175 person months (McGibbon, 1997)

There can be wide variation in predicted values for these different parametric models. Kemerer (1997) reported average errors of over 600% for COCOMO 81 models. For many of these estimation models, there are also estimation formulas for the schedule time as a function of the effort in person months, because the relationship between schedule time and person months tends to be nonlinear, particularly for large, complex projects. There are software programs available that implement the COCOMO II methodology, and these are available from both academic (USC) and commercial vendors. As of this writing, the latest version was USC COCOMO II.2001.0. For the latest in COCOMO II methodology, see one of the Web sites that maintain this information, such as http://sunset.usc.edu/research/COCOMOII or www.jsc.nasa.gov/bu2/COCOMO.html. Other estimating and costing software programs are available from a number of vendors, including www.softstarsystems.com/, www.galorath.com/, www.rcinc.com, www.pricesystems.com, www.marotz.com, and www.spr.com.

Whether historical estimates, benchmarks, or parametric models are used, there is a level of uncertainty in the estimates. For parametric models, there will be uncertainty both in the input to the model (size and other factors) as well as in the model itself. People tend to be overly optimistic, particularly in IT, about estimating the time required to do something. On the other hand, it is natural for the people who will actually be doing the work or responsible for the work (i.e., managers) to throw extra "padding" into an estimate. Program evaluation and review technique (PERT) is the most common method of dealing with uncertainties in task duration estimates. It assumes that these durations are random variables drawn from a beta distribution (Malcolm, Rosebooom, & Clark, 1959). Such a distribution is show in Figure 7.22. To calculate the PERT time for a task, three estimates are obtained: optimistic, pessimistic, and most likely. The PERT time (PT) is calculated as:

PT = (Optimistic + Pessimistic + 4 * Most Likely)/6.

The task variance and standard deviation is given by:

Variance = $(\text{Pessimistic} - \text{Optimistic})^2/36$

Standard Deviation = (Pessimistic – Optimistic)/6.

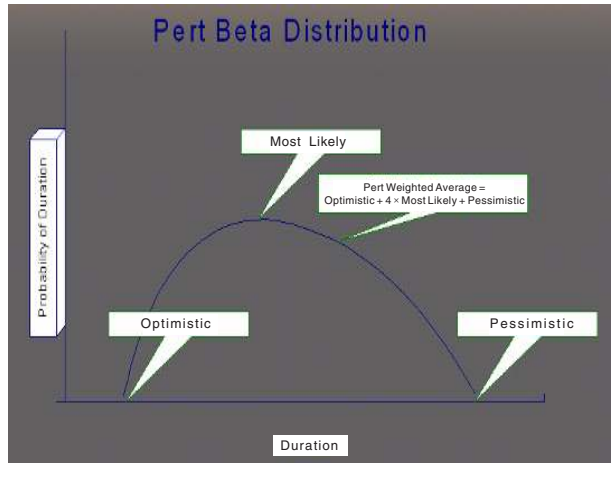
The beta distribution is typically not symmetric but it is unimodal. To find the combined variance from a group of tasks, we do not sum the individual variances, but instead sum

deviation and take its square root. Many authors have investigated the accuracy and utility of PERT estimates and other techniques over the years, and there are other techniques that may be more accurate or useful in certain disciplines. However, for IT projects, the PERT estimate provides a reasonable method of estimating task times in the presence of uncertainty. Using our earlier example with Web forms, an optimistic value could be based upon the situation whereby none of the fields required any special handling (unusual validation or foreign key lookup), and the pessimistic value would be where each field required some type of special handling. It is important not to confuse the notions of pessimistic estimates with risk analysis, discussed later in this book. The pessimistic estimate is not made in regard to some unusual or catastrophic occurrence, such as a prolonged power outage, illness of the assigned programmer, and so on.

Task Sequencing and the Critical Path

Once a WBS is established and the effort involved in each work packet is determined, the next step is to sequence the tasks in the order they need to be performed. Task sequencing typically uses a network diagram, sometimes called a logic diagram, that shows how the project tasks (work packets) will flow in time from beginning to end of project. This sequencing can actually be done without knowledge of the effort involved in each task, and sometimes the sequencing is done before estimation. However, for the determination of a critical path, we must know how long it will take to do each task as well as the order in which the tasks can be done. How long it takes to do each task depends not only on the amount of effort of a task (typically measured in person hours) but also on how many people will be assigned to a task.

Figure 7.22. Beta distribution

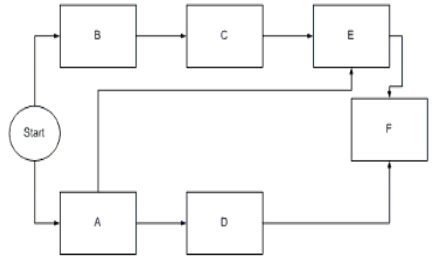


A network diagram is constructed by taking the tasks and putting them in the order they will be done, taking into account any dependencies. This process is called activity sequencing. Note that this process need not be done at the lowest level of the WBS, although it is typically done at that level to maximize the parallelism (the number of task that can be worked on simultaneously). When done at higher levels of the WBS, whole blocks of task may be dependent upon other blocks of tasks. For example, all of the implementation tasks may be dependent on the completion of all of the design tasks as in a strict waterfall methodology of software development. To maximize parallelism, and thus minimize overall development calendar time, it is more common to have the implementation and unit testing of one software module only dependent upon completion of the design for that one module (in reality one module's design may involve a number of design tasks such as its user interface, involved database tables, etc.). Eventually, all the software modules come together for integration testing, which would be dependent upon completion of the implementation and unit testing of every module.

The two general ways to do draw these network diagrams are activity-on-arrow and activity-on-node. The original critical path method (CPM) developed by Dupont and Remington Rand in 1957 used the activity-on-arrow method and used one time estimate only per task (as opposed to several estimates as in the PERT method). The more common method today is to show the activity on a node and the precedence diagramming method (PDM) is based on this representation. In the PDM method, nodes or boxes represent tasks, and arrows between the nodes show task dependencies. This is illustrated in Figure 7.23 with activities labeled A, B, and so on. There are four types of dependencies that may apply between two tasks:

1. *Finish-to-start:* A task must finish before another one can start.
2. *Finish-to-finish:* A task must finish before another can finish.
3. *Start-to-start:* A task must start before another one can start.
4. *Start-to-finish:* A task must start before another one can finish.

Figure 7.23. PDM method



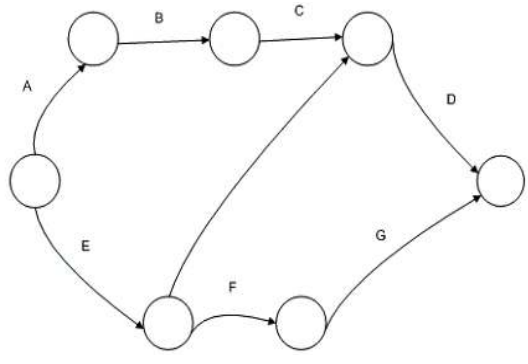
In the illustration, task E is dependent on tasks A and C. All, except the finish-to-start, are not commonly used and can be simulated by setting an earliest start date for a task, if necessary. There are no dummy activities in this method, and this method is easier to visualize than the activity-on-arrow method. Most modern project management software packages use this method.

In the activity-on-arrow (AOA) method, arrows are used to represent tasks, and ovals are used to relate tasks. Generally, only finish-to-start relationships can be used. This method was the one originally used with PERT. Dummy activities may be needed to show dependencies between tasks (often shown with a dotted line). This method does have a visual advantage in that the length of arrow can optionally be used for duration of task. Figure 7.24 illustrates this, and in that figure task D is dependent on E and C, and a dummy activity (no task identification letter) is used to show the dependence of D on E.

The critical path is the longest path(s) through a network diagram from the project start to project end. It can be found using either of the network diagramming techniques after the tasks are estimated; for the activity-on-arrow method, the critical path may run through a dummy activity. The critical path determines the earliest completion date for the project. It may change during a project, if task estimates are revised. There can be more than one critical path, and this is not desirable because it increases overall risk. Critical paths are calculated by most project management scheduling software systems and are done using dynamic or linear programming techniques (an example of using linear programming is found in Chapter XVI).

The amount of time a particular task can be delayed without delaying the entire project completion is called the total slack or float for a task. The float for the entire project is a fixed completion time minus the scheduled time (it may be negative). Tasks on a critical path typically have no slack (unless the project completion date is longer than the critical path activities). In the network diagram of Figure 7.25 (activity-on-node method), durations are shown in the nodes, and the critical path is shown using the bigger arrows. The length of this project is 35 time units, which is the sum of the tasks on the critical path.

Figure 7.24. AOA method



A forward pass through the network is used to calculate early-start and early-finish dates, and a backward pass (from the overall finish date) is used to determine late-start and late-finish dates. The float (total slack) for a task is the difference between the early-start and late-start dates for a task (or between the early-finish and late-finish dates, or the late-finish minus early-start dates plus task duration). Free slack is the amount of time a task can be delayed without delaying the early start of its successor task (minimum early-start time of all successor tasks minus early start of this task minus task duration). Most computer systems show total slack (float) time per task, and project managers use this number to know which tasks need to be most closely monitored and also in allocating scarce resources amongst multiple projects. Free slack can be used to determine how much a critical path task can be *crashed* before the critical path changes. If the fixed end date changes for a project, the critical path does not change; however, the overall project float may become negative. If the overall project float becomes negative (you are behind schedule), then the project manager would consider crashing or fast tracking (covered later in this book). In Figure 7.26, the floats are shown for the tasks that are not on the critical path. Note that the float for task A is 1, not 2, because delaying A by 2 units would not delay C, but the path through A and D to E would become longer than the path through B and C to E.

Lag is a necessary wait time between tasks; one adds lag time to a predecessor task to delay the start of a dependent task. For example, you must wait 2 days after pouring concrete before you can start to build the house frame. Another example, you must order certain materials 10 days before you need to use them. Lag is typically shown on the arrows on an activity-on-node diagram. Lead time is the amount of time you can overlap dependent tasks.

For time-sensitive IT projects, schedule compression can best be achieved by removing unnecessary coupling (dependencies) between tasks, thus allowing for more work on more tasks simultaneously. As discussed earlier in this chapter, this can be done with the proper choice of architecture, standards, and methodology. For example if requirement specifications, standards compliance, and end user documentation is done early in a project, then design and implementation can be done in parallel with integration test planning and end user training. Methodologies were discussed earlier in this book, and

Figure 7.25. PDM network diagram

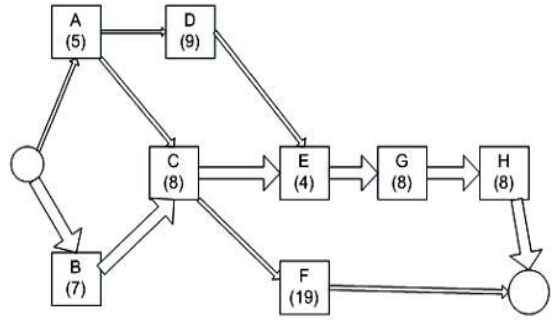
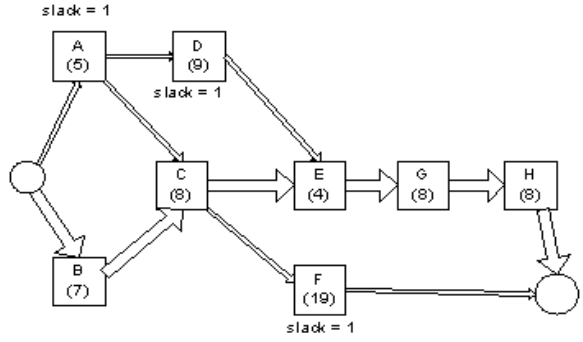


Figure 7.26. Network slack



if modern free-flow incremental techniques are used instead of classical waterfall methods, then more work (WBS packets) can be done in parallel.

Scheduling

After all the tasks are sequenced, a schedule can be developed. The difference between a network diagram and a schedule is that a schedule is *calendar based* and takes into account the length of work weeks and holidays. The basic differences in these project management concepts are

- WBS defines "what"
- Network Diagram defines "how"
- Schedule defines "when"
- RAM defines "who"

If specific resources are assigned at this time, the schedule may also take into consideration resource availability (i.e., percent of their time devoted to this project) and resource absences for vacation and other time off. Many project management software packages can also level resources so that a resource is used an equal amount of time in each workweek. Adding specific resource issues to scheduling will generally lengthen a project's duration. The critical path on the schedule may also be different than the critical path in the network diagram due to resource availability and resource leveling. Graphic representations of schedules can be produced in expanded network diagrams and the nodes in such a schedule drawing may look like those shown in Figure 7.27; an example of such a diagram is shown in Figure 7.28.

Figure 7.27. Schedule nodes

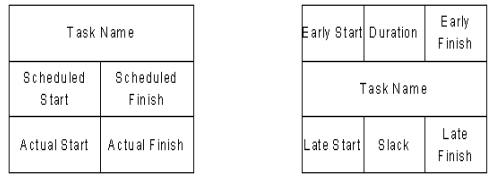
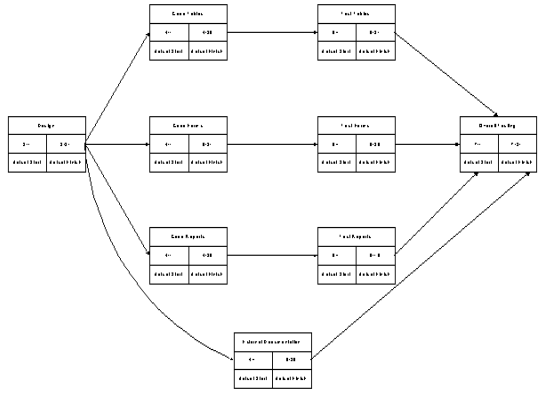
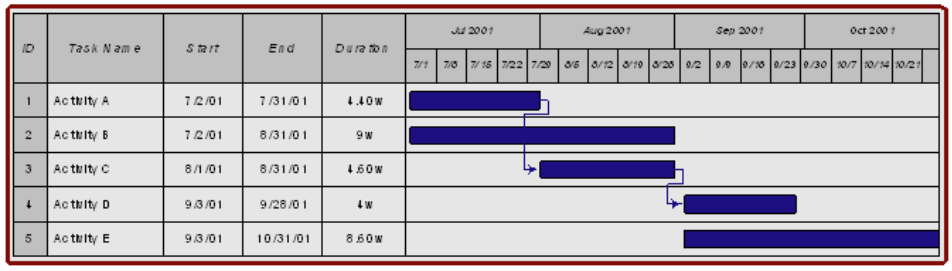


Figure 7.28. Network diagram with expanded nodes



The typical graphical representation of a schedule, however, is the Gantt chart, which is the most commonly used graphic in project management. It was named after Henry Gantt, who first developed it during World War I with the U.S. Army. He was a mechanical engineer and management consultant and his charts were later employed on major U.S. infrastructure projects, including the Hoover Dam and the interstate highway system. An example is shown in Figure 7.29. The Gantt chart represents each task as a horizontal bar and is a time-line view of the tasks. This chart is a bar graph on which the length of each bar designates the start and finish dates for each task. Each task is labeled with its WBS code and description. There are many variations of the basic Gantt chart; more information may be added in extra columns or in the graphic area with the bars. In the graphic area there may be precedence relations, shading in the bars to show percent completions of the tasks, coloring of the bars to show tasks on the critical path, milestones (zero-length event markers), WBS hierarchy and envelopes, and so forth. A problem with the Gantt chart is that it portrays an even distribution of effort from the start of a task to the end of that task (particularly when a percent-complete shading of the bars is used). Another problem with the traditional Gantt chart is that it requires a lot of "real estate" on a computer screen or a printout for large projects. Figure 7.30 shows a Plan Gantt chart from the FiveAndDime system, which also shows planned cost in the bars; this is a Web-hyperlink-based system where one can click on WBS codes for added detail or to "drill down" to the next level of the WBS.

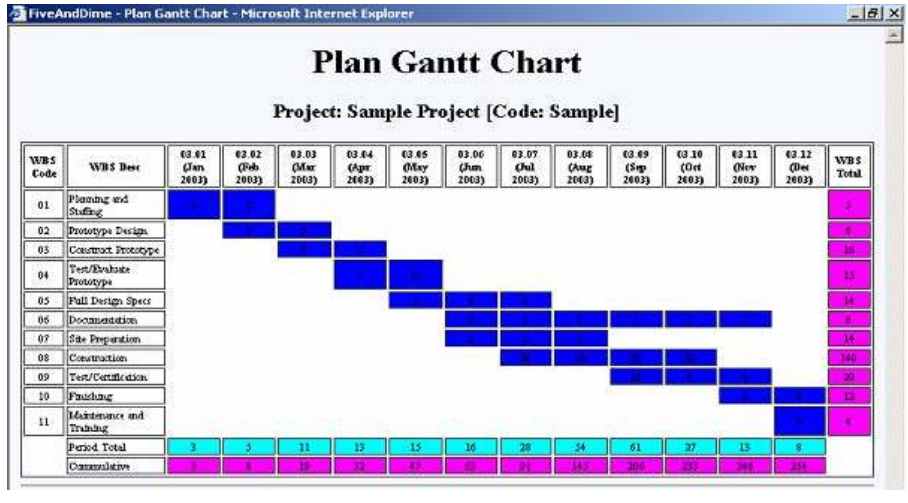
Figure 7.29. Gantt chart



ID	Task Name	Start	End	Duration
1	Activity A	7/2/01	7/31/01	4.40w
2	Activity B	7/2/01	8/31/01	9w
3	Activity C	8/1/01	8/31/01	4.60w
4	Activity D	9/3/01	9/28/01	4w
5	Activity E	9/3/01	10/31/01	8.60w

One must be careful, particularly with IT projects, in not overplanning a schedule. A project manager should focus on managing the project, not managing the schedule. A number of companies plan resource allocations using specific resource assignments and plan down to the day. IT human resources are largely professional types, they may work varying numbers of hours per day, they may be called upon to help another person or another project from time to time, and they may take off a day or two for whatever reason, whenever they so chose. As mentioned previously, estimates involve considerable uncertainty, and productivity rates of individuals, even within the same category, vary greatly in IT. These factors have always played a major role in IT project scheduling; one of the best discussions of this goes back to Frederick Brooks's classic book, *The Mythical Man-Month* (Brooks, 1975). Many companies have wasted valuable time with dedicated project schedulers, who constantly modify the project schedule, trying to keep up with individual resource availability. A rule of thumb for a work packet is to have one

Figure 7.30. Planning Gantt chart



timekeeping reporting period, which is 1 or 2 weeks in most organizations. Making resource assignments at that level (either specific or generic resources) is a more reasonable approach for IT projects. In other words, for each WBS work packet after the effort (in person hours) is determined, simply assign one or more individuals to that packet for the number of periods (weeks) necessary. Unrealistic schedules are one of the primary reasons for project crises and failures. According to Young (2003), seven things to do before one can create a realistic schedule are:

1. Nail down the scope and requirements
2. Prototype the biggest technical risks
3. Create a model for the user interface
4. Pay attention to industry-standard estimates for similar projects
5. Let each person create an estimated task schedule for his own work
6. Accept only observable, measurable status reports
7. Subdivide all the tasks until each task takes 1 or 2 weeks to complete

Resource Assignment and Costing Methods

Resource assignment in IT is usually necessary to estimate the time duration of the work packets. In some industries, for some projects, time estimates may be independent of resource assignments. Resource assignment for preliminary task time and cost estimates is usually done generically, using a table, as was shown in Figure 7.18. The category rates may be averages for employees in that category, midpoints for the corporate salary bracket, contractor rates for contract labor, or some external average rate, such as that of a regulatory body or trade union.

At some time early in the project, the effort to staff the project begins. The initial staffing depends upon the time phased need of certain resources, the lead time to bring said resources onto the project, conflicting needs for said resources, and other related resource issues. The project may be reestimated (repriced) at that time if there is a significant difference in the rates for the actual resources versus the planned rates for those resources; for contracted projects, this depends on the terms of the contract. A responsibility assignment matrix (RAM) may be developed, which is similar to the organizational assignment matrix previously shown but shows the actual resources (mapping between WBS and RBS) as well as the organization; some organizations may maintain a single RAM for all projects. Figure 7.31 shows a window from the FiveAndDime system for adding resource information; FiveAndDime allows the adding of resources to the overall organization (such as for generic rate categories) and the adding of project-specific resources.

Figure 7.31. Form to add resource

Assignment of actual resources to a project brings up the problem of resource leveling, that is making sure that a single resource is not overcommitted, not just to a single project, but between all projects in which that resource is involved. Some project management software programs can do some form of resource leveling within one project and some can do leveling between multiple projects. In either case, these automatic solutions may be far from optimum. The first step in resource leveling should be to obtain a report showing the planned time for each resource across all projects for the time span of interest. When a resource is overcommitted there are several possible solutions, including setting up task priorities, setting up a finish to start precedence between the tasks competing for said resource, getting the resource to agree to be overcommitted for a period of time (preferably in writing), and finding an alternative resource internally or via contracting or outsourcing.

As the project proceeds, actual costs accrue to the project. The costing of resources, particularly human resources, is one where confidentiality concerns may override concerns for accuracy of actual costs. The hours a person charges to a project and particular WBS codes are regularly reported. There are, however, a variety of ways to turn those actual hours into actual costs. Some organizations apply the actual burdened (including benefits, taxes, and possibly other factors) hourly cost of each individual. Although this approach to actual cost acquisition is the most accurate, it can reveal individual salaries. To solve this problem, many organizations use some type of average rates, which may be the same table used for cost estimating purposes. Another way to handle this problem for projects, where almost all the cost is labor, is to report and control actual costs in terms of person-hours instead of dollars; these details are discussed later in this book.

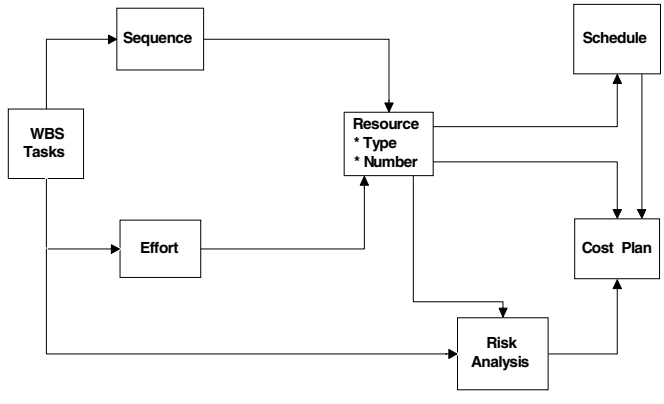
Developing the Cost Plan

After the schedule is developed, the estimated cost of each work packet is rolled up to the top level of the WBS to develop the cost plan. The cost plan, sometimes referred to as a cash flow analysis, shows the amount of money that has been planned to be spent as a function of time. This process is illustrated in Figure 7.32. Sometimes two such curves are developed, one with each work packet starting at the scheduled time, and another on which each noncritical task starts at the latest start time. If cash flow is an issue, one can choose between the two curves or, in a contract situation, one can base a payment schedule on one of the two curves.

Once the cost plan is accepted, it becomes the baseline for the project; baseline cost plans may be modified later as change orders are added. As an example, consider the following two-level WBS (Level zero is in bold) from an example appearing in the *Project Management Journal* (Brandon, 1998, 1999):

- *Planning & Staffing:* Project Plan, Resource Commitments
- *Prototype Design:* Requirements, Design
- *Construct Prototype:* Detail Design, Construction
- *Test/Evaluate Prototype:* Test Plan Development, Component Testing, Full Testing, Destructive Testing, Test Documentation
- *Full Design Specifications:* External Specifications, Internal Specifications
- *Documentation:* Internal Specifications, Required External Documents
- *Site Preparation:* Site Design, Site Construction
- *Construction:* Component Construction, Component Assembly
- *Test/Certification:* Test Plan Development, Component Testing, Full Testing, Destructive Testing, Test Documentation

Figure 7.32. Schedule and cost plan development



- *Finishing:* Component Finishing, Assembly Finishing
- *Maintenance Training:* Maintenance Personnel, Supervisory Personnel

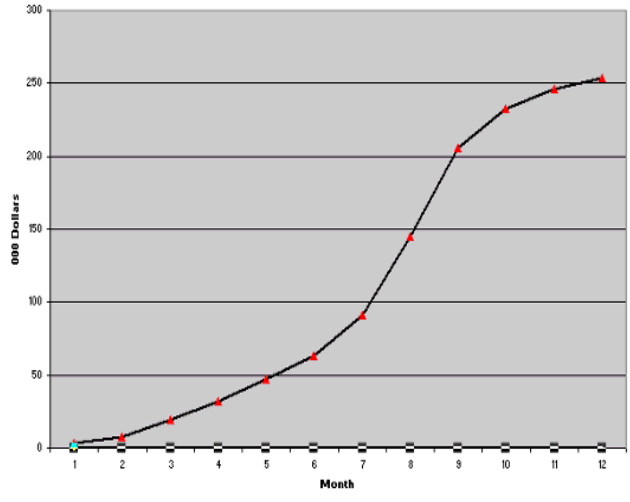
The cost plan for this WBS (at Level zero) is shown in Figure 7.33, and a graph of the cost plan is shown in Figure 7.34. The cost plan becomes the basis for cost performance reporting as discussed later in this book. In developing a cost plan in most software project management programs each WBS work packet is associated with one of more resources and has a start date and a complete date. The cost calculated from the effort of the task (usually in person hours) and the rates of the assigned resource(s) is evenly spread from the start date to the completion date. In other types of project management software, time periods (work periods) are separate entities in the program and cost is associated with a time period and a work packet to form a plan item; in other words, a plan item is how much is to be spent in a particular time period for a particular WBS code.

Figure 7.33. Cost plan in spreadsheet

Project Cost Plan

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	TOTAL
WBS (Level 0)													
Planning & Staffing	3	2											5
Prototype Design		3	3										6
Construct Prototype			8	8									16
Test/Evaluate Prototype				5	10								15
Full Design Specs					5	6	3						14
Documentation						2	2	1	1	1	1		8
Site Preparation						8	3	3					14
Construction							20	60	60	20			140
Test/Certification									10	6	4		20
Finishing											8	4	12
Maintenance Training												4	4
Monthly Plan	3	5	11	13	15	16	28	54	61	27	13	8	254
Cumulative	3	8	19	32	47	63	91	145	206	233	246	254	

Figure 7.34. Cost plan graph



As was discussed in Chapter VI, in many organizations and in the U.S. government, standard calendars are set up based upon a work period of 1 week (for either a fiscal or calendar year). This later method provides for nonlinear distribution of WBS cost over time, easier handling of project overhead (level of effort) costs, and improved performance metric calculation (discussed later in this book). Figure 7.35, from the FiveAndDime systems, shows project work plan items for the current example, and Figure 7.30 showed a Plan Gantt chart which shows the individual plan items in a spreadsheet like chart.

This cost plan is sometimes called the *planned cost based on known work* and may include risk factors at the individual WBS work packet level. The inclusion of added cost based upon risk factors at the WBS work packet level (mainly estimation risk factors) and overall project risk is discussed in Chapter VIII. There may also be some unpriced additional work (real or suspected) added which brings the cost up to the expected work cost or the performance measurement baseline. The difference between the expected work cost and the planned cost based on known work is called the undistributed budget. The undistributed budget is for known work and sometimes used as a holding account until a formal contract change is made.

The project manager generally has responsibility to manage to the performance measurement baseline. In addition a management reserve may be added as a reserve for overall project risks including unforeseen work that is still within the overall scope of the project. The performance measurement baseline plus the management reserve adds up to the total project budget (or for a contract situation the negotiated cost). These provisions are part of the ANSI standard on Earned Value Management System guidelines (ANSI/EIA-748-1998), which is discussed later in this book and is illustrated in Figure 7.36.

For IT projects, costs other than labor may be present and these may or may not be allocated to the project. Contractor and consultant costs are usually part of the WBS and RBS structures. Other direct costs, such as hardware, software, training, entertainment, and travel, may made be part of the WBS or just added to the cost plan developed via

Figure 7.35. Cost plan table

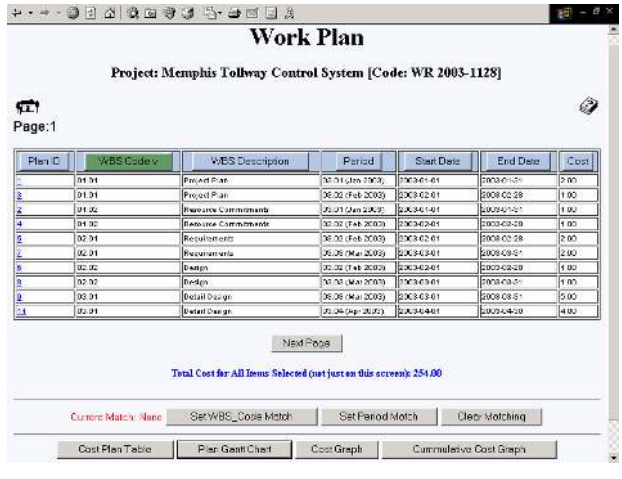
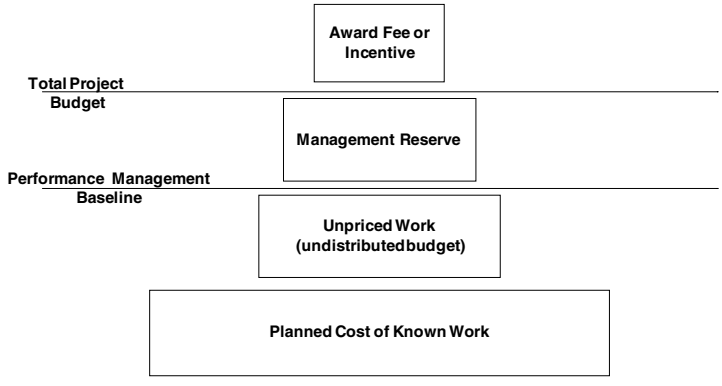


Figure 7.36. Hierarchy of contract cost



the WBS. Indirect, overhead, or level of effort types of cost may also be assigned to a project and become part of the project budget. These may be General and Administrative (G & A), Internal Operations and Maintenance (I & O), allocations (space, utilities, consumables, communications, etc.), nonproject time (general meetings, general training, etc.). In some organizations these costs are included in the burdened rates for the labor categories. In some organizations, these costs are given separate WBS codes. In projectized organizations, and even in traditional line management organizations, as more work becomes team and projected based, it becomes desirable to associate these costs with projects. If given separate WBS codes, their contribution to the cost plan is typically a linear spread over the project life.

Chapter Summary

The formulation of a detail schedule and cost plan has been discussed and illustrated in this chapter. WBS formulation, task sequencing, task estimation, scheduling, and costing methods have all been covered. The detail scope, time, and cost planning of this chapter forms the basis for other detail plans described in subsequent chapters including the risk plan, procurement plan, HR plan, quality plan, control plan, and change plan.

References

Albrecht, A. (1979, October 14-17). Measuring application development productivity. *Proceedings of the SHARE/GUIDE IBM Applications Development Symposium,* Monterey, CA.

Boehm, B. (1981). *Software engineering economics.* Upper Saddle River, NJ: Prentice Hall.

Boehm, B. (2000). *Software estimation with Cocomo II.* Upper Saddle River, NJ: Prentice Hall.

Brandel, M. (2004, August). Budgetary black holes. *Computerworld.*

Brandon, D. (1998). Implementing earned value easily and effectively. *Project Management Journal, 29*(2) 11-18.

Brandon, D. (1999). Implementing earned value easily and effectively. In J. Pinto (Ed.), *Essentials of project control* (pp. 113-134). Newton Square, PA: Project Management Institute Press.

Brooks, F. (1975). *The mythical man-month*. Boston: Addison Wesley.

IFPUG. (1996). *IFPUG counting practices manual.* Retrieved from www.ifpug.org/

Jones, C. (1995). Backfiring: Converting lines of code to function points. *IEEE Computer*, *28*(11), 87-88.

Kemerer, C. (1987). An empirical validation of software cost estimation models. *Communications of the ACM, 30*(5), 416-429.

Kiewel, B. (1998, January). Measuring progress in software development. *PM Network,* 29-32.

Klastorn, T. (2004). *Project management—Tools and trade-offs*. New York: Wiley.

Malcolm, D., Roseboom, J., & Clark, C. (1959). Application of a technique for research and development program evaluation. *Operations Research, 7*(5), 646-669.

McGibbon, T. (1997, August 20). Modern empirical cost and schedule estimation tools. *DOD Data and Analysis Center for Software Report (DACS),* Air Force Research Laboratory.

PMI. (2000). *The project management body of knowledge (PMBOK).* Newton Square, PA. ISBN 1-880410-22-2.

PMI. (2004). *Practice standards for work breakdown structures.*

Raz, T., & Globerson, S. (1998, December). Effective sizing and content definition of work packages. *Project Management Journal,* 17-23.

Roetzheim, W., & Beasley, R. (1998). *Software project cost and schedule estimating: Best practices*. Upper Saddle River, NJ: Prentice Hall.

Young, S. (2003, August). Why IT projects fail. *Computerworld,* p. 44.