**Estimation**

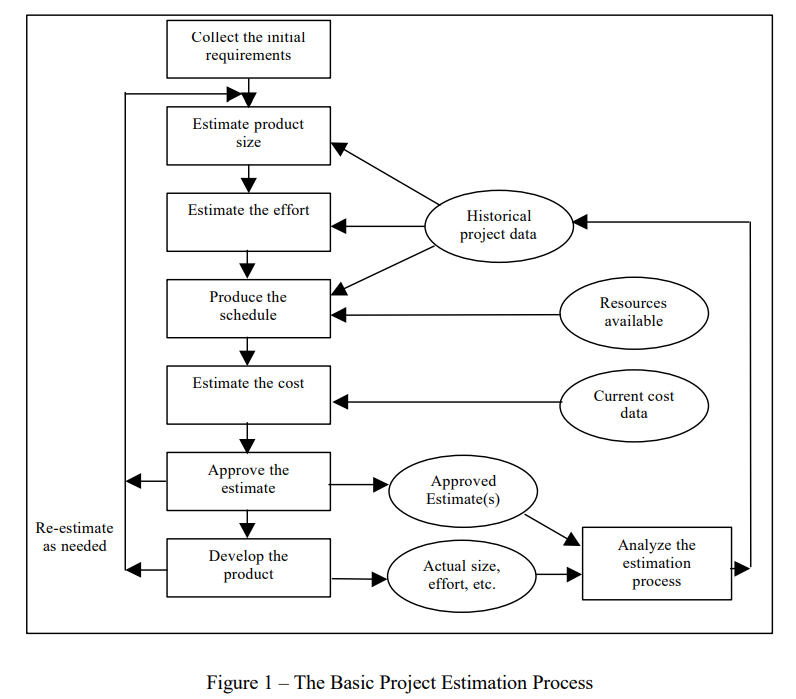
**Estimation** is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.

Estimation determines how much money, effort, resources, and time it will take to build a specific system or product. Estimation is based on −

* Past Data/Past Experience
* Available Documents/Knowledge
* Assumptions
* Identified Risks

The four basic steps in Software Project Estimation are −

* Estimate the size of the development product.
* Estimate the effort in person-months or person-hours.
* Estimate the schedule in calendar months.
* Estimate the project cost in agreed currency.



**When and why is Estimation done?**

Effective software project estimation is one of the most challenging and important activities in software development. Proper project planning and control is not possible without a sound and reliable estimate. As a whole, the software industry doesn’t estimate projects well and doesn’t use estimates appropriately. We suffer far more than we should as a result and we need to focus some effort on improving the situation. Under-estimating a project leads to under-staffing it (resulting in staff burnout), under-scoping the quality assurance effort (running the risk of low quality deliverables), and setting too short a schedule (resulting in loss of credibility as deadlines are missed). For those who figure on avoiding this situation by generously padding the estimate, over-estimating a project can be just about as bad for the organization! If you give a project more resources than it really needs without sufficient scope controls it will use them. The project is then likely to cost more than it should (a negative impact on the bottom line), take longer to deliver than necessary (resulting in lost opportunities), and delay the use of your resources on the next project. Software Project Estimation 101 The four basic steps in software project estimation are: 1) Estimate the size of the development product. This generally ends up in either Lines of Code (LOC) or Function Points (FP), but there are other possible units of measure. A discussion of the pros & cons of each is discussed in some of the material referenced at the end of this report. 2) Estimate the effort in person-months or person-hours. 3) Estimate the schedule in calendar months. 4) Estimate the project cost in dollars (or local currency)

**When is Estimation Done?**

* Project Initiation: During the early stages of a project, to assess feasibility and provide a rough idea of the project's scope and cost.
* Project Planning: In the detailed planning phase, to develop a more accurate project schedule, budget, and resource allocation plan.
* Before Each Iteration/Sprint: In iterative or agile methodologies, estimation is done before each iteration or sprint to plan the tasks to be accomplished in the upcoming cycle.
* Before Major Milestones: Before reaching significant project milestones or decision points to reassess progress and adjust plans accordingly.

**Why is Estimation Done?**

* Resource Allocation: To determine the resources (human, financial, technical) needed to complete the project and allocate them efficiently.
* Budgeting: To develop a project budget and ensure that the project remains financially viable and within the allocated funds.
* Scheduling: To create a realistic project timeline, identifying key milestones and delivery dates.
* Risk Management: To identify potential risks related to time and resource constraints and to develop mitigation strategies.
* Stakeholder Communication: To provide stakeholders (clients, management, team members) with a clear understanding of the project's scope, timeline, and cost, helping manage expectations.
* Performance Tracking: To set benchmarks for measuring project progress and performance, enabling better control and management of the project.

**Project Estimation Techniques**

1. **Top-Down Estimate**

Once more detail is learned on the scope of the project, this technique is usually followed where high-level chunks at the feature or design level are estimated and are decomposed progressively into smaller chunks or work-packets as information is detailed.

2. **Bottom-Up Estimate**

This technique is used when the requirements are known at a discrete level where the smaller workpieces are then aggregated to estimate the entire project. This is usually used when the information is only known in smaller pieces.

3. **Analogous Estimating**

This project estimation technique is used when there is a reference to a similar project executed and it is easy to correlate with other projects. Expert judgment and historical information of similar activities in a referenced project are gathered to arrive at an estimate of the project.

4. **Parametric Estimate**

This technique uses independent measurable variables from the project work. For example, the cost for construction of a building is calculated based on the smallest variable as the cost to build a square feet area, the effort required to build a work packet is calculated from the variable as lines of codes in a software development project. This technique gives more accuracy in project estimation.

5. **Three-point Estimating**

This technique uses a mathematical approach as the weighted average of an optimistic, most likely and pessimistic estimate of the work package. This is often known as the PERT (Program Evaluation and Review Technique).

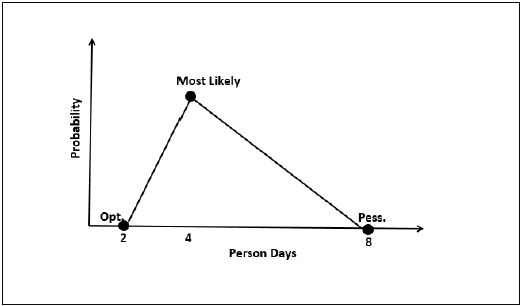
Three-point Estimation looks at three values −

* the most optimistic estimate (O),
* a most likely estimate (M), and
* a pessimistic estimate (least likely estimate (L)).

There has been some confusion regarding Three-point Estimation and PERT in the industry. However, the techniques are different. You will see the differences as you learn the two techniques. Also, at the end of the PERT technique, the differences are collated and presented. If you want to look at them first, you can.

Three-point Estimate (E) is based on the simple average and follows triangular distribution.

**E = (O + M + L) / 3**



Standard Deviation

In Triangular Distribution,

Mean = (O + M + L) / 3

Standard Deviation = √ [((O − E)2 + (M − E)2 + (L − E)2) / 2]

Three-point Estimation Steps

**Step 1** − Arrive at the WBS(Work Breakdown Structure).

**Step 2** − For each task, find three values − most optimistic estimate (O), a most likely estimate (M), and a pessimistic estimate (L).

**Step 3** − Calculate the Mean of the three values.

**Mean = (O + M + L) / 3**

**Step 4** − Calculate the Three-point Estimate of the task. Three-point Estimate is the Mean. Hence,

**E = Mean = (O + M + L) / 3**

**Step 5** − Calculate the Standard Deviation of the task.

**Standard Deviation (SD) = √ [((O − E)2 + (M − E)2 + (L - E)2)/2]**

**Step 6** − Repeat Steps 2, 3, 4 for all the Tasks in the WBS.

**Step 7** − Calculate the Three-point Estimate of the project.

**E (Project) = ∑ E (Task)**

6. **What-If Analysis**

This project estimation technique uses assumptions based on varying factors like scope, time, cost, resources, etc., to evaluate the possible outcomes of the project by doing impact analysis. In a usual scenario, the project estimate is done by conducting estimation workshops with the stakeholders of the project, senior team members who could give valuable inputs to the estimation exercise. The high-level scope is broken down into smaller work packages, components, and activities, each work package is estimated by effort and resources needed to complete the work package. The project may be detailed into the smallest chunk that can be measured.

**Size Oriented Metrics**

* **LOC Metrics**

It is one of the earliest and simpler metrics for calculating the size of the computer program. It is generally used in calculating and comparing the productivity of programmers. These metrics are derived by normalizing the quality and productivity measures by considering the size of the product as a metric.

**Following are the points regarding LOC measures:**

1. In size-oriented metrics, LOC is considered to be the normalization value.
2. It is an older method that was developed when FORTRAN and COBOL programming were very popular.
3. Productivity is defined as KLOC / EFFORT, where effort is measured in person-months.
4. Size-oriented metrics depend on the programming language used.
5. As productivity depends on KLOC, so assembly language code will have more productivity.
6. LOC measure requires a level of detail which may not be practically achievable.
7. The more expressive is the programming language, the lower is the productivity.
8. LOC method of measurement does not apply to projects that deal with visual (GUI-based) programming. As already explained, Graphical User Interfaces (GUIs) use forms basically. LOC metric is not applicable here.
9. It requires that all organizations must use the same method for counting LOC. This is so because some organizations use only executable statements, some useful comments, and some do not. Thus, the standard needs to be established.
10. These metrics are not universally accepted.

Advantages of LOC

1. Simple to measure

Disadvantage of LOC

1. It is defined on the code. For example, it cannot measure the size of the specification.
2. It characterizes only one specific view of size, namely length, it takes no account of functionality or complexity
3. Bad software design may cause an excessive line of code
4. It is language dependent
5. Users cannot easily understand it

* **Halstead's Software Metrics**

According to Halstead's "A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operand."

Token Count

In these metrics, a computer program is considered to be a collection of tokens, which may be classified as either operators or operands. All software science metrics can be defined in terms of these basic symbols. These symbols are called as a token.

The basic measures are

n1 = count of unique operators.  
n2 = count of unique operands.  
N1 = count of total occurrences of operators.  
N2 = count of total occurrence of operands.

In terms of the total tokens used, the size of the program can be expressed as N = N1 + N2.

Halstead metrics are:

**Program Volume (V)**

The unit of measurement of volume is the standard unit for size "bits." It is the actual size of a program if a uniform binary encoding for the vocabulary is used.

        V=N\*log2n

**Program Level (L)**

The value of L ranges between zero and one, with L=1 representing a program written at the highest possible level (i.e., with minimum size).

        L=V\*/V

**Program Difficulty**

The difficulty level or error-proneness (D) of the program is proportional to the number of the unique operator in the program.

        D= (n1/2) \* (N2/n2)

**Programming Effort (E)**

The unit of measurement of E is elementary mental discriminations.

        E=V/L=D\*V

**Estimated Program Length**

According to Halstead, The first Hypothesis of software science is that the length of a well-structured program is a function only of the number of unique operators and operands.

        N=N1+N2

* **Functional Point (FP) Analysis**

Allan J. Albrecht initially developed function Point Analysis in 1979 at IBM and it has been further modified by the International Function Point Users Group (IFPUG). FPA is used to make estimate of the software project, including its testing in terms of functionality or function size of the software product. However, functional point analysis may be used for the test estimation of the product. The functional size of the product is measured in terms of the function point, which is a standard of measurement to measure the software application.

Objectives of FPA

The basic and primary purpose of the functional point analysis is to measure and provide the software application functional size to the client, customer, and the stakeholder on their request. Further, it is used to measure the software project development along with its maintenance, consistently throughout the project irrespective of the tools and the technologies.

**Following are the points regarding FPs**

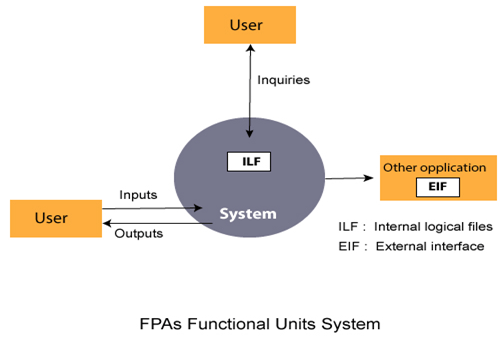
1. FPs of an application is found out by counting the number and types of functions used in the applications. Various functions used in an application can be put under five types, as shown in Table:

**Types of FP Attributes**

|  |  |
| --- | --- |
| **Measurements Parameters** | **Examples** |
| 1.Number of External Inputs(EI) | Input screen and tables |
| 2. Number of External Output (EO) | Output screens and reports |
| 3. Number of external inquiries (EQ) | Prompts and interrupts. |
| 4. Number of internal files (ILF) | Databases and directories |
| 5. Number of external interfaces (EIF) | Shared databases and shared routines. |

All these parameters are then individually assessed for complexity.

**The FPA functional units are shown in Fig:**



1. FP characterizes the complexity of the software system and hence can be used to depict the project time and the manpower requirement.

2. The effort required to develop the project depends on what the software does.

3. FP is programming language independent.

4. FP method is used for data processing systems, business systems like information systems.

5. The five parameters mentioned above are also known as information domain characteristics.

6. All the parameters mentioned above are assigned some weights that have been experimentally determined and are shown in Table

**Weights of 5-FP Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Measurement Parameter** | **Low** | **Average** | **High** |
| 1. Number of external inputs (EI) | 7 | 10 | 15 |
| 2. Number of external outputs (EO) | 5 | 7 | 10 |
| 3. Number of external inquiries (EQ) | 3 | 4 | 6 |
| 4. Number of internal files (ILF) | 4 | 5 | 7 |
| 5. Number of external interfaces (EIF) | 3 | 4 | 6 |

The functional complexities are multiplied with the corresponding weights against each function, and the values are added up to determine the UFP (Unadjusted Function Point) of the subsystem.

**Translating size Estimate into effort Estimate**

Translating size estimates into effort estimates is a critical step in software project management. Effort estimation involves determining the amount of work (usually measured in person-hours, person-days, or person-months) required to complete a project based on its size. Here are the steps and methods commonly used to translate size estimates into effort estimates:

**Steps to Translate Size Estimates into Effort Estimates**

Determine the Size Estimate: Use one of the formal size estimation methods like Function Points (FP), Use Case Points (UCP), or Lines of Code (LOC).

Select an Effort Estimation Model: Choose an appropriate model based on the project's context, available historical data, and the selected size estimation method. Common models include COCOMO, Expert Judgment, and the use of historical data.

Gather Historical Data: Collect historical data from past projects similar to the current one. This data should include both size and effort metrics to create a basis for estimation.

Calculate Effort Using the Chosen Model: Apply the selected effort estimation model to the size estimate to derive the effort estimate.

Effort Estimation Models

1. COCOMO (Constructive Cost Model)

The COCOMO model translates size estimates into effort estimates using predefined formulas and coefficients. It considers different project types (organic, semi-detached, embedded) and factors in the complexity of the project.

Basic COCOMO Formula:

Effort (Person-Months)=𝑎×(KLOC)𝑏

For example, for an organic project:

a=2.4 and

𝑏=1.05

Intermediate and Detailed COCOMO:

These models add effort multipliers and scale factors to the basic formula to adjust for various project attributes and environmental factors.

2. Function Points to Effort

Function Points (FP) can be converted into effort using productivity rates. The productivity rate (e.g., FP per person-month) is often derived from historical data.

Formula:

Effort (Person-Months)

=

Function Points

Productivity Rate (FP/Person-Month)

Effort (Person-Months)=

Productivity Rate (FP/Person-Month)

Function Points

​

For instance, if the productivity rate is 5 FP per person-month and the project is 200 FP, the effort estimate is

200

5

=

40

5

200

​

=40 person-months.

3. Use Case Points (UCP)

Use Case Points can be converted to effort by multiplying the UCP by a productivity factor (e.g., person-hours per UCP).

Formula:

Effort (Person-Hours)

=

UCP

×

Productivity Factor (Person-Hours/UCP)

Effort (Person-Hours)=UCP×Productivity Factor (Person-Hours/UCP)

For example, if the productivity factor is 20 person-hours per UCP and the project has 100 UCP, the effort estimate is

100

×

20

=

2000

100×20=2000 person-hours.

4. Expert Judgment

Experts use their experience and historical data to translate size estimates into effort estimates. This method is more subjective but can be very accurate if the experts have relevant experience.

Process:

Experts review the size estimate and compare it with similar past projects.

They consider factors such as team experience, project complexity, and technology stack.

They provide an effort estimate based on this analysis.

Adjusting Effort Estimates

Once the initial effort estimate is calculated, it may need to be adjusted for various factors:

Project Complexity:

Adjust for the complexity of the project, considering factors like algorithm complexity, integration requirements, and performance constraints.

Team Experience:

Consider the experience and skill level of the team members who will be working on the project.

Tool and Environment:

Adjust for the tools and development environment, which can impact productivity.

Risk Factors:

Factor in potential risks and uncertainties that could affect the project's effort.

Example Calculation

Let's consider a project with the following parameters:

Size Estimate: 500 Function Points

Productivity Rate: 4 FP per person-month

Effort Estimate Calculation:

Effort (Person-Months)

=

500

FP

4

FP/Person-Month

=

125

Person-Months

Effort (Person-Months)=

4 FP/Person-Month

500 FP

​

=125 Person-Months

Adjusting for a team with moderate experience and some anticipated risks:

Adjustment Factor: 1.2 (20% increase for experience and risk)

Adjusted Effort=125 Person-Months×1.2=150 Person-Months

Translating effort Estimates into schedule Estimate

Translating effort estimates into schedule estimates involves determining the duration required to complete a project based on the estimated effort (typically measured in person-hours, person-days, or person-months) and the availability of resources. This process requires considering factors such as the number of team members, their productivity, and any project constraints or dependencies.

Steps to Translate Effort Estimates into Schedule Estimates

Determine the Effort Estimate:

The effort estimate is usually expressed in person-months, person-days, or person-hours.

Identify Resource Availability:

Determine the number of resources (team members) available and their working hours per day/week/month.

Calculate Initial Schedule Estimate:

Divide the total effort estimate by the number of available resources to get the initial schedule estimate.

Adjust for Project Factors:

Consider factors like team productivity, potential risks, holidays, and non-working days to adjust the initial schedule estimate.

Example Calculation

Let's consider an example where:

Effort Estimate: 150 person-months

Number of Resources (Team Members): 10

Working Days per Month: 20

Working Hours per Day: 8

Step-by-Step Calculation

Calculate Total Work Hours:

Total Work Hours

=

Effort Estimate (in person-months)

×

Working Days per Month

×

Working Hours per Day

Total Work Hours=Effort Estimate (in person-months)×Working Days per Month×Working Hours per Day

Total Work Hours

=

150

person-months

×

20

days/month

×

8

hours/day

=

24

,

000

hours

Total Work Hours=150 person-months×20 days/month×8 hours/day=24,000 hours

Calculate Schedule Estimate in Months:

Schedule Estimate (in months)

=

Total Work Hours

Number of Resources

×

Working Hours per Month

Schedule Estimate (in months)=

Number of Resources×Working Hours per Month

Total Work Hours

​

Working Hours per Month

=

20

days/month

×

8

hours/day

=

160

hours/month

Working Hours per Month=20 days/month×8 hours/day=160 hours/month

Schedule Estimate (in months)

=

24

,

000

hours

10

×

160

hours/month

=

24

,

000

1

,

600

=

15

months

Schedule Estimate (in months)=

10×160 hours/month

24,000 hours

​

=

1,600

24,000

​

=15 months

Adjustments for Project Factors

Productivity Adjustment:

If team productivity is expected to be less than 100%, adjust the schedule accordingly. For example, if productivity is 80%, the schedule needs to be extended.

Adjusted Schedule Estimate

=

Initial Schedule Estimate

Productivity

Adjusted Schedule Estimate=

Productivity

Initial Schedule Estimate

​

Adjusted Schedule Estimate

=

15

months

0.8

=

18.75

months

Adjusted Schedule Estimate=

0.8

15 months

​

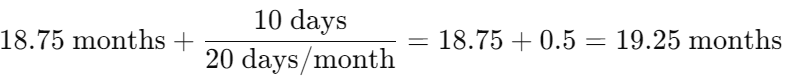
=18.75 months

Non-Working Days:

Account for holidays, weekends, and other non-working days. For example, if there are 2 weeks of holidays:

Non-Working Days Adjustment= Holidays/Working Days per Month

​Adjusted Schedule Estimate=



Risk Factors:

Add contingency time to account for potential risks and uncertainties. For example, adding a 10% buffer:

Final Schedule Estimate=Adjusted Schedule Estimate×(1+Risk Buffer)

Final Schedule Estimate

Final Schedule Estimate=19.25 months×1.10=21.175 months

Effort and Schedule Sizing the project by using function points, SLOC, or other methods is a job only half done. Transforming the size to a deliverable effort within a comfortable schedule makes the project planning a complete success story. Further, the total project effort (for example, in person months) that needs to be consumed in a given schedule provides the guidance to do a proper resource loading

A diagram of a business process

Description automatically generated

Once the phase-wise resource loading details are available, you can apply the resource rate to each category of resource—such as project manager, architect, analyst, and developer—for the duration of the assignment. Thus the total base cost for the project is calculated. You can then add project management, configuration management, and other overheads as appropriate to get the gross cost. Figure 7.3 shows the broad parameters that are to be taken into account during different lifecycle stages of the project execution. Deriving Effort The overall project effort (typically measured in person months) is directly dependent on two critical inputs: application size and project team/ programmer productivity. The steps to calculate each of these items are as follows:

• From the given specification for the application, calculate the size of the application. The size can be estimated by using one of the popular estimation methods, such as

• Function points method: Output will be in FP count.

• Object points method: Output will be a list of classes of simple/ medium/complex categories. • SLOC method: Output will be a “gut feel” of lines of code.

• Make sure that you have the productivity (delivery rate) available for the technology platform on which the application is being developed. For every language there are available average productivity figures that should be adjusted by the historic project productivity data for your own IT organization. Productivity of your project team:

• Is based on competency of programmers

• Is specific to a given technology

• Is dependent on the software development environment

• Convert application size to effort (in person months):

• Effort = Application size × productivity

• The effort thus derived is the total project effort that would be spent for all the lifecycle stages of the project, from requirements creation through user acceptance. Add project management and configuration management effort as applicable. The effort is also the aggregate of the individual effort spent by each of the resources assigned to the project.

**Scheduling**

Transforming the overall project effort into a delivery schedule (elapsed time) is somewhat tricky. If the right approach is not applied, the risks of project failure are high. There are three alternatives to calculate the schedule:

• Use popular scheduling methods like COCOMO II.

• “Gut feel” scheduling based on past experience.

• Schedule driven by business user need.

The schedule data that can be obtained by one of these methods is in the form of duration required to deliver the project itself. For example, the schedule could span 10 months from the start date of the project. The schedule thus encompasses all the lifecycle stages of the entire project. From the total duration given to the project team, the project manager must divide the time into lifecycle-based segments. The lifecycle phase percentage is also to be based on historical delivery information of the IT organization. For example, with 10 months of elapsed time, the schedule can be split as follows:

• Requirements: 2 months (20 percent)

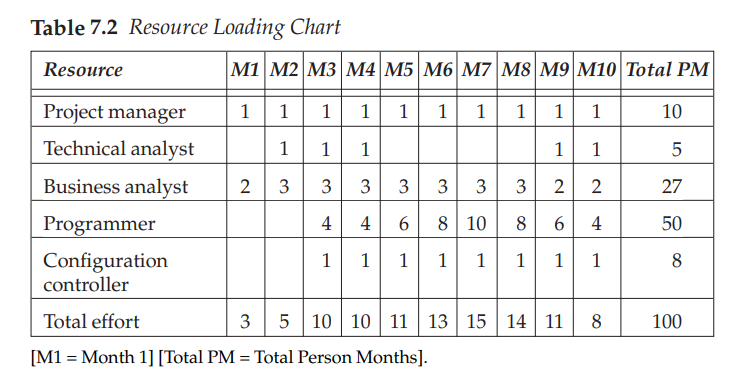
• Detailed design: 1.5 months (15 percent)

• Build and unit test: 4 months (40 percent)

• System and integration test: 2.5 months (25 percent)

**Resource Loading**

Resource loading is a complex activity and has to be worked on with extreme care. Improper assignment of resources will have an impact on



project delivery schedules as well as the quality of outputs. Resource loading requires two critical mapping considerations:

• The right resource role for the appropriate lifecycle stage. For example, you need to know when to assign a project manager, an architect, or a programmer.

• The right duration of assignment. This includes when to assign and when to release. The effort spent by each resource is determined by tactful resource allocation method.

For Figure 7.3 shown earlier in this section, the resource loading patterns are displayed illustratively in Table 7.2. For your project, you can prepare a table showing resource role assignments for the appropriate durations. For example, assume a total project effort of 100 person months. This effort includes project management and configuration management effort. Table 7.2 illustrates the typical resource loading based on the percentage breakup of elapsed time, as given in the example in this chapter.