Module-03

Software Project Scheduling,control & Monitoring

Lecture 1

1. Motivation:

Poorly managed software production leads to wastage of money and time.

2. Learning Objective:

This module explains that people involved in the Software Estimation planning & process will be able to come out with a cost effective solution to develop any kind of a software solution.

1. Students will be able to understand the Software Estimation- empirical estimation models-Cost/Effort estimation model.
2. Students will be able to understand the Planning- work breakdown structure, Gantt chart. They will be able to discuss cost and how to manage the schedule slippage.

3. Objective:

To get knowledge about the activities involved in project management.

4. Prerequisite:

Phases of software life cycle.

5. Learning OutComes:

The student will learn steps that help a software team to understand Software Estimation using empirical estimation models.
The student will learn steps of Planning- work breakdown structure.
The student will understand the Gantt chart, Discuss cost and schedule slippage.

6. Syllabus:

Module	Content	Duration	Self Study Time
2.1	Software Estimation- empirical estimation models Cost/Effort estimation	2 lectures	2 hours
2.2	Planning- work breakdown structure, Gantt chart, Discuss cost and schedule slippage	2 lectures	2 hours

5. Learning

- Product metrics
- Estimation techniques for project
- Project Planning
- Project Scheduling
- Project Tracking

1. **Weightage:**

   04 hours

7. Abbreviations:

(1) PERT - Program Evaluation Review Technique
(2) LOC – Lines Of Code
(3) FP - Function Point
(4) COCOMO - Constructive Cost Model
(5) WBS –Work Breakdown Structure

8. Key Definitions:

1. **Planning:** Project planning is part of project management, which relates to the use of schedules such as Gantt charts to plan and subsequently report progress within the project environment.

2. **Estimation:** The ability to accurately estimate the time and/or cost taken for a project to come in to its successful conclusion is a serious problem for software engineers. The use of a repeatable, clearly defined and well understood software development process has, in recent years, shown itself to be the most effective method of gaining useful historical data that can be used for statistical estimation. In particular, the act of sampling more frequently, coupled with the loosening of constraints between parts of a project, has allowed more accurate estimation and more rapid development times.
3. **Software metric:** Software metric is a measure of some property of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development. The goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments.
4. **LOC (Lines Of Code):** Lines of code is a software metric used to measure the size of a software program by counting the number of lines in the text of the program's source code. SLOC is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or effort once the software is produced.
5. **FP (Function Point):** It is a unit of measurement used in software project estimation. It is used to measure the functional size of an information system. The functional size reflects the amount of functionality that is relevant to and recognized by the user in the business. It is independent of the technology used to implement the system.
6. **COCOMO (Constructive Cost Model):** The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry Boehm. The model uses a basic regression formula, with parameters that are derived from historical project data and current project characteristics.
7. **Gantt chart:** A Gantt chart is a type of bar chart that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Some Gantt charts also show the dependency (i.e., precedence network) relationships between activities.
8. **PERT chart:** A PERT chart is a project management tool used to schedule, organize, and coordinate tasks within a project. PERT stands for Program Evaluation Review Technique, a methodology developed by the U.S. Navy in the 1950s to manage the Polaris submarine missile program.

9. Theory

2.1 Software estimation – Empirical estimation models – Cost/Effort estimation

Planning & Estimation

Software project management begins with a set of activities that are collectively called project planning. Before the project can begin, the manager and the software team must estimate the work to be done, the resources that will be required, and the time that will elapse from start to finish. Whenever estimates are made, future is looked into and some degree of uncertainty is accepted as a matter of course.

3.1.1 Product metrics

Metrics are measurable ways to design and assess the software product. Software metrics domain is partitioned into process, project and product metrics. The product metrics are private to an individual and are often combined to develop project metrics that are public to a software team. Project metrics are then consolidated to create process metrics that are public to the software organization as a whole. An organization combines metrics that come from different individuals or projects by normalizing the measures. By doing so, it is possible to create software metrics that enable comparison to broader organizational averages.

In this section product metric is concentrated. Product metrics help software engineers gain insight into the design and construction of the software they build. Product metrics focus on specific attributes of software engineering work products and are collected as technical tasks (analysis, design, coding, and testing) are being conducted. Software engineers use product metrics is to help them build higher-quality software.

<u>The product Metrics Landscape:</u>

Although a wide variety of metrics taxonomies have been proposed, the following outline addresses the most important metrics area:

i. Metrics for the analysis model. These metrics address various aspects of the analysis model and include:

Functionally delivered-provides an indirect measure of the functionality that is packaged within the software.

System size-measures of the overall size of the system defined in terms of information available as part of the analysis model.

Specification quality-provides an indication of the specificity and completeness of a requirements specification.

ii. Metrics for the design model. These metrics quantify design attributes in a manner that allows a software engineer to assess design quality. Metrics include:

Architectural metrics-provide an indication of the quality of the architectural design.

Component-level metrics-measure the complexity of software components and other characteristics that have a bearing on quality.

Interface design metrics-focus primarily on usability.

Specialized OO design metrics-measure characteristics of classes and their communication and collaboration characteristics.

iii. Metrics for source code. These metrics measure the source code and can be used to assess its complexity, maintainability, and testability, among other characteristics:

Halstead metrics-controversial but nonetheless fascinating, these metrics provide unique measures of a computer program.

Complexity metrics-measure the logical complexity of source code (can also be considered to be component-level design metrics).

Length metrics-provide an indication of the size of the software.

iv. Metrics for testing. These metrics assist in the design of effective test cases and evaluate the efficacy of testing:

Statement and branch coverage metrics-lead to the design of test cases that provide program coverage.

Defect-related metrics-focus on bugs found, rather than on the tests themselves.

Testing effectiveness-provide a real-time indication of the effectiveness of tests that have been conducted.

In-process metrics-process related metrics that can be determined as testing is conducted.

In many cases, metrics for one model may be used in later software engineering activities. For example, design metrics may be used to estimate the effort required to generate source code. In addition, design metrics may be used in test planning and test case design.

3.1.2 Estimation- LOC, FP, and COCOMO models

Estimation of resources, cost, and schedule for a software engineering effort requires experience, access to good historical information, and the courage to commit to quantitative predictions when qualitative information is all that exists. Estimation carries inherent risk1 and this risk leads to uncertainty.

Project complexity has a strong effect on the uncertainty inherent in planning. Complexity, however, is a relative measure that is affected by familiarity with past effort. The first-time developer of a sophisticated e-commerce application might consider it to be exceedingly complex. However, a software team developing its tenth e-commerce Web site would consider such work run of the mill. A number of quantitative software complexity measures have been proposed. Such measures are applied at the design or code level and are therefore difficult to use during software planning (before a design and code exist). However, other, more subjective assessments of complexity (e.g., the function point complexity adjustment factors which will be discussed below) can be established early in the planning process.

Project size is another important factor that can affect the accuracy and efficacy of estimates. As size increases, the interdependency among various elements of the software grows rapidly. Problem decomposition, an important approach to estimating, becomes more difficult because decomposed elements may still be formidable. To paraphrase Murphy's law: "What can go wrong will go wrong”—and if there are more things that can fail, more things will fail.

The *degree of structural uncertainty* also has an effect on estimation risk. In this context, structure refers to the degree to which requirements have been solidified, the ease with which functions can be compartmentalized, and the hierarchical nature of the information that must be processed.

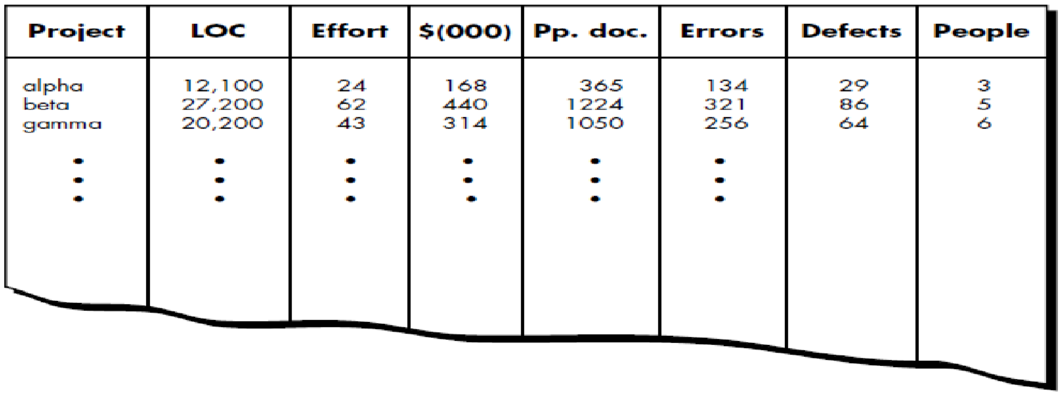
The availability of historical information has a strong influence on estimation risk. By looking back, we can emulate things that worked and improve areas where problems arose. When comprehensive software metrics are available for past projects, estimates can be made with greater assurance, schedules can be established to avoid past difficulties, and overall risk is reduced.

<u>LOC (Lines Of Code):</u>

Measurements in the physical world can be categorized in two ways: direct measures (e.g., the length of a bolt) and indirect measures (e.g., the "quality" of bolts produced, measured by counting rejects). Software metrics can be categorized similarly (direct and indirect measures).

Direct measures of the software engineering process include cost and effort applied. Direct measures of the product include lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time. The cost and effort required to build software, the number of lines of code produced, and other direct measures are relatively easy to collect, as long as specific conventions for measurement are established in advance.

LOC comes under the category of size-oriented metrics. Size-oriented software metrics are derived by normalizing quality and/or productivity measures by considering the *size* of the software that has been produced. If a software organization maintains simple records, a table of size-oriented measures, such as the one shown in the Figure below, can be created. The table lists each software development project that has been completed over the past few years and corresponding measures for that project.



Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
⋮	⋮	⋮	⋮	⋮	⋮		

Figure 2.1: Size-oriented metrics

In order to develop metrics that can be assimilated with similar metrics from other projects, we choose lines of code as our normalization value. From the rudimentary data contained in the table, a set of simple size-oriented metrics can be developed for each project:

- Errors per KLOC (thousand lines of code).
- Defects per KLOC.
- $ per LOC.
- Page of documentation per KLOC.

In addition, other interesting metrics can be computed:

- Errors per person-month.
- LOC per person-month.
- $ per page of documentation.

Size-oriented metrics are not universally accepted as the best way to measure the process of software development. Proponents of the LOC measure claim that LOC is an "artifact" of all software development projects that can be easily counted. On the other hand, opponents argue that LOC measures are programming language dependent, that they penalize well-designed but shorter programs, that they cannot easily accommodate nonprocedural languages.

LOC and FP (Function Point) estimation are distinct estimation techniques. Yet both have a number of characteristics in common. The project planner begins with a bounded statement of software scope and from this statement attempts to decompose software into problem functions that can each be estimated individually. LOC or FP (the estimation variable) is then estimated for each function. Alternatively, the planner may choose another component for sizing such as classes or objects, changes, or business processes affected.

Baseline productivity metrics (e.g., LOC/person-month or FP/person-month) are then applied to the appropriate estimation variable, and cost or effort for the function is derived. Function estimates are combined to produce an overall estimate for the entire project.

The LOC and FP estimation techniques differ in the level of detail required for decomposition and the target of the partitioning. When LOC is used as the estimation variable, decomposition10 is absolutely essential and is often taken to considerable levels of detail. For FP estimates, decomposition works differently. Rather than focusing on function, each of the information domain characteristics—inputs, outputs, data files, inquiries, and external interfaces—as well as the 14 complexity adjustment values (which is discussed below). The resultant estimates can then be used to derive a FP value that can be tied to past data and used to generate an estimate.

Regardless of the estimation variable that is used, the project planner begins by estimating a range of values for each function or information domain value. Using historical data or (when all else fails) intuition, the planner estimates an optimistic, most likely, and pessimistic size value for each function or count for each information domain value. An implicit indication of the degree of uncertainty is provided when a range of values is specified.

A three-point or expected value can then be computed. The *expected value* for the estimation variable (size), *S*, can be computed as a weighted average of the optimistic (*s*opt), most likely (*s*m), and pessimistic (*s*pess) estimates. For example,

$$S = (s_{opt} + 4s_m + s_{pess})/6 \text{ -------------------------------- } (1)$$

gives heaviest credence to the "most likely" estimate and follows a beta probability distribution. We assume that there is a very small probability the actual size result will fall outside the optimistic or pessimistic values.

Once the expected value for the estimation variable has been determined, historical LOC or FP productivity data are applied. One can't be sure that the estimates are correct. Any estimation technique, no matter how sophisticated, must be cross-checked with another approach. Even then, common sense and experience must prevail.

<u>An example of LOC-Based estimation:</u>

As an example of LOC and FP problem-based estimation techniques, a software package to be developed for a computer-aided design application for mechanical components is considered. A review of the *System Specification* indicates that the software is to execute on an engineering workstation and must interface with various

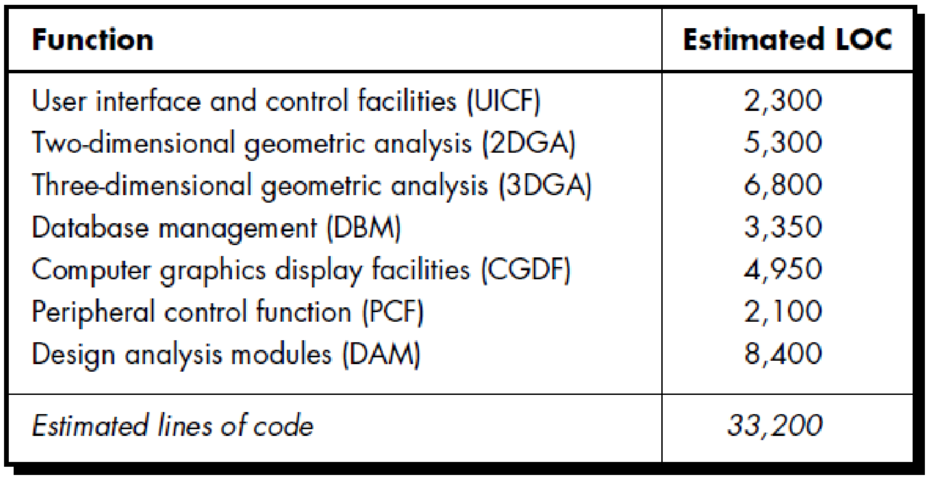
computer graphics peripherals including a mouse, digitizer, high resolution color display and laser printer.
Using the *System Specification* as a guide, a preliminary statement of software scope can be developed:
"The CAD software will accept two- and three-dimensional geometric data from an engineer. The engineer will interact and control the CAD system through a user interface that will exhibit characteristics of good human/machine interface design. All geometric data and other supporting information will be maintained in a CAD database. Design analysis modules will be developed to produce the required output, which will be displayed on a variety of graphics devices. The software will be designed to control and interact with peripheral devices that include a mouse, digitizer, laser printer, and plotter."

This statement of scope is preliminary—it is *not* bounded. Every sentence would have to be expanded to provide concrete detail and quantitative bounding. For example, before estimation can begin the planner must determine what "characteristics of good human/machine interface design" means or what the size and sophistication of the "CAD database" are to be.

For our purposes, it is assumed that further refinement has occurred and that the following major software functions are identified:

- User interface and control facilities (UICF)
- Two-dimensional geometric analysis (2DGA)
- Three-dimensional geometric analysis (3DGA)
- Database management (DBM)
- Computer graphics display facilities (CGDF)
- Peripheral control function (PCF)
- Design analysis modules (DAM)

Following the decomposition technique for LOC, an estimation table, shown in the Figure below, is developed. A range of LOC estimates is developed for each function. For example, the range of LOC estimates for the 3D geometric analysis function is optimistic—4600 LOC, most likely—6900 LOC, and pessimistic—8600 LOC.



Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

Figure 2.2: Estimation table for the LOC method

Applying equation (1) mentioned above, the expected value for the 3D geometric analysis function is 6800 LOC. Other estimates are derived in a similar fashion. By summing vertically in the estimated LOC column, an estimate of 33,200 lines of code is established for the CAD system.

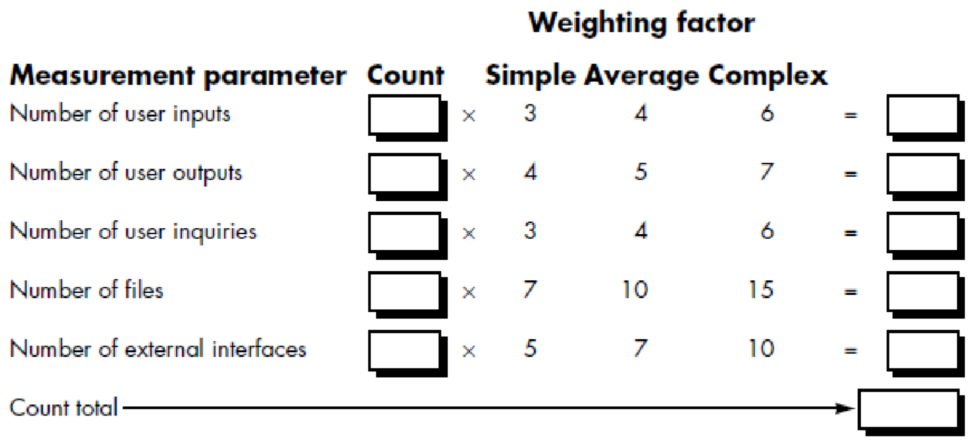
A review of historical data indicates that the organizational average productivity for systems of this type is 620 LOC/pm. Based on a burdened labor rate of $8000 per month, the cost per line of code is approximately $13. Based on the LOC estimate and the historical productivity data, the total estimated project cost is $431,000 and the estimated effort is 54 person-months (the estimates are rounded-off here).

Function Point (FP):

As mentioned earlier, software metrics can be categorized as direct measures and indirect measures. As discussed before direct measures of the product include LOC. Indirect measures of the product include functionality, quality, complexity, efficiency, reliability, maintainability, and many other "–abilities". These indirect measures come under the category of 'function-oriented metrics'.

Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value. Since 'functionality' cannot be measured directly, it must be derived indirectly using other direct measures. Function-oriented metrics were first proposed by Albrecht, who suggested a measure called the *function point.* Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.

Function points are computed by completing the table shown below. Five information domain characteristics are determined and counts are provided in the appropriate table location. Information domain values are defined in the following manner:



		Weighting factor				
Measurement parameter	Count	Simple	Average	Complex		
Number of user inputs		× 3	4	6	=	
Number of user outputs		× 4	5	7	=	
Number of user inquiries		× 3	4	6	=	
Number of files		× 7	10	15	=	
Number of external interfaces		× 5	7	10	=	
Count total						

Figure 2.3: Computing function points

Number of user inputs. Each user input that provides distinct application oriented data to the software is counted. Inputs should be distinguished from inquiries, which are counted separately.
Number of user outputs. Each user output that provides application oriented information to the user is counted. In this context output refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.
Number of user inquiries. An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.
Number of files. Each logical master file (i.e., a logical grouping of data that may be one part of a large database or a separate file) is counted.
Number of external interfaces. All machine readable interfaces (e.g., data files on storage media) that are used to transmit information to another system are counted.

Once these data have been collected, a complexity value is associated with each count. Organizations that use function point methods develop criteria for determining whether a particular entry is simple, average, or complex. Nonetheless, the determination of complexity is somewhat subjective.

To compute function points (FP), the following relationship is used:

$$FP = \text{count total} \times [0.65 + 0.01 \times \Sigma(F_i)] \text{ ------------------------------ } (2)$$

where count total is the sum of all FP entries obtained from the figure for computing function points shown above.

The F_i (i = 1 to 14) are "complexity adjustment values" based on responses to the following questions:

1. Does the system require reliable backup and recovery?
2. Are data communications required?

3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

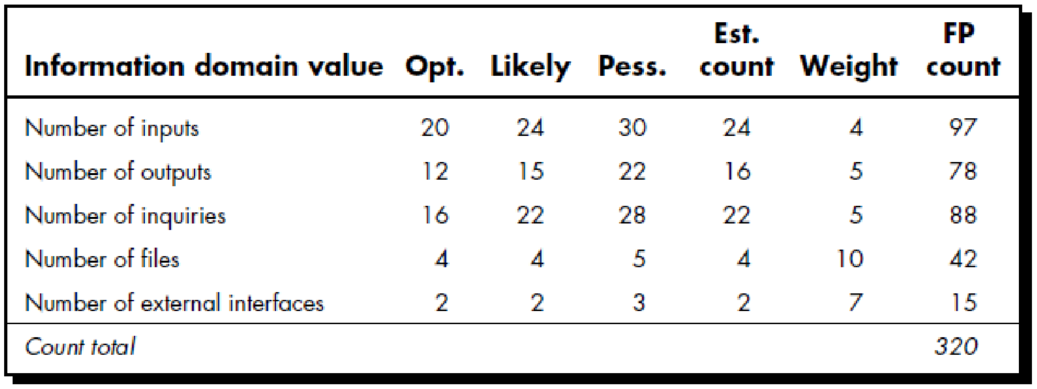
Each of these questions is answered using a scale that ranges from 0 (not important or applicable) to 5 (absolutely essential). The constant values in equation (2) and the weighting factors that are applied to information domain counts are determined empirically.

Once function points have been calculated, they are used in a manner analogous to LOC as a way to normalize measures for software productivity, quality, and other attributes:

- Errors per FP.
- Defects per FP.
- $ per FP.
- Pages of documentation per FP.
- FP per person-month.

<u>An example of FP-based estimation:</u>

Decomposition for FP-based estimation focuses on information domain values rather than software functions. Referring to the function point calculation table presented in the Figure shown below, the project planner estimates inputs, outputs, inquiries, files, and external interfaces for the CAD software (this was the one used for LOC-based estimation also). For the purposes of this estimate, the complexity weighting factor is assumed to be average. The figure below, presents the results of this estimate.



Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of inputs	20	24	30	24	4	97
Number of outputs	12	15	22	16	5	78
Number of inquiries	16	22	28	22	5	88
Number of files	4	4	5	4	10	42
Number of external interfaces	2	2	3	2	7	15
Count total						*320*

Figure 2.4: Estimating information domain values

Each of the complexity weighting factors is estimated and the complexity adjustment factor is computed as described below:

Factor	**Value**
Backup and recovery	4
Data communications	2
Distributed processing	0
Performance critical	4
Existing operating environment	3
On-line data entry	4
Input transaction over multiple screens	5
Master files updated on-line	3
Information domain values complex	5
Internal processing complex	5
Code designed for reuse	4
Conversion/installation in design	3
Multiple installations	5
Application designed for change	5
Complexity adjustment factor	1.17

Finally, the estimated number of FP is derived:

$$FP_{\text{estimated}} = \text{count-total} \times [0.65 + 0.01 \times \sum (Fi)]$$

$$FP_{\text{estimated}} = 375$$

The organizational average productivity for systems of this type is 6.5 FP/pm. Based on a burdened labor rate of $8000 per month, the cost per FP is approximately $1230. Based on the LOC estimate and the historical productivity data, the total estimated project cost is $461,000 and the estimated effort is 58 person-months.

COCOMO Model:

This model is based on empirical estimation. This estimation for computer software uses empirically derived formulas to predict effort as a function of LOC or FP. Values for LOC or FP are estimated using the approach described in the previous paragraphs. But instead of using the tables described in those sections, the resultant values for LOC or FP are plugged into the estimation model.

In his classic book on "software engineering economics," Barry Boehm introduced a hierarchy of software estimation models bearing the name COCOMO, for COnstructive COst MOdel. The original COCOMO model became one of the most widely used and discussed software cost estimation models in the industry. It has evolved into a more comprehensive estimation model, called *COCOMO II*. Like its predecessor, COCOMO II is actually a hierarchy of estimation models that address the following areas:

Application composition model. Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.

Early design stage model. Used once requirements have been stabilized and basic software architecture has been established.

Post-architecture-stage model. Used during the construction of the software.

Like all estimation models for software, the COCOMO II models require sizing information. Three different sizing options are available as part of the model hierarchy: object points, function points, and lines of source code.

The COCOMO II application composition model uses object points and is illustrated in the following paragraphs. It should be noted that other, more sophisticated estimation models (using FP and KLOC) are also available as part of COCOMO II.

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

Table 2.1: Complexity weighting for object types

Like function points (described earlier), the *object point* is an indirect software measure that is computed using counts of the number of (1) screens (at the user interface), (2) reports, and (3) components likely to be required to build the application. Each object instance (e.g., a screen or report) is classified into one of three complexity levels (i.e., simple, medium, or difficult) using criteria suggested by Boehm. In essence, complexity is a function of the number and source of the client and server data tables that are required to generate the screen or report and the number of views or sections presented as part of the screen or report.

Once complexity is determined, the number of screens, reports, and components are weighted according to the Table shown above. The object point count is then determined by multiplying the original number of object instances by the weighting factor in the table and summing to obtain a total object point count. When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted:

NOP = (object points) x [(100 _ %reuse)/100]

where NOP is defined as new object points.

To derive an estimate of effort based on the computed NOP value, a "productivity rate" must be derived. The table below presents the productivity rate 'PROD = NOP/person-month' for different levels of developer experience and development environment maturity.

Developer's experience/capability	Very low	Low	Nominal	High	Very high
Environment maturity/capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

Table 2.2: Productivity rates for object points

Once the productivity rate has been determined, an estimate of project effort can be derived as:

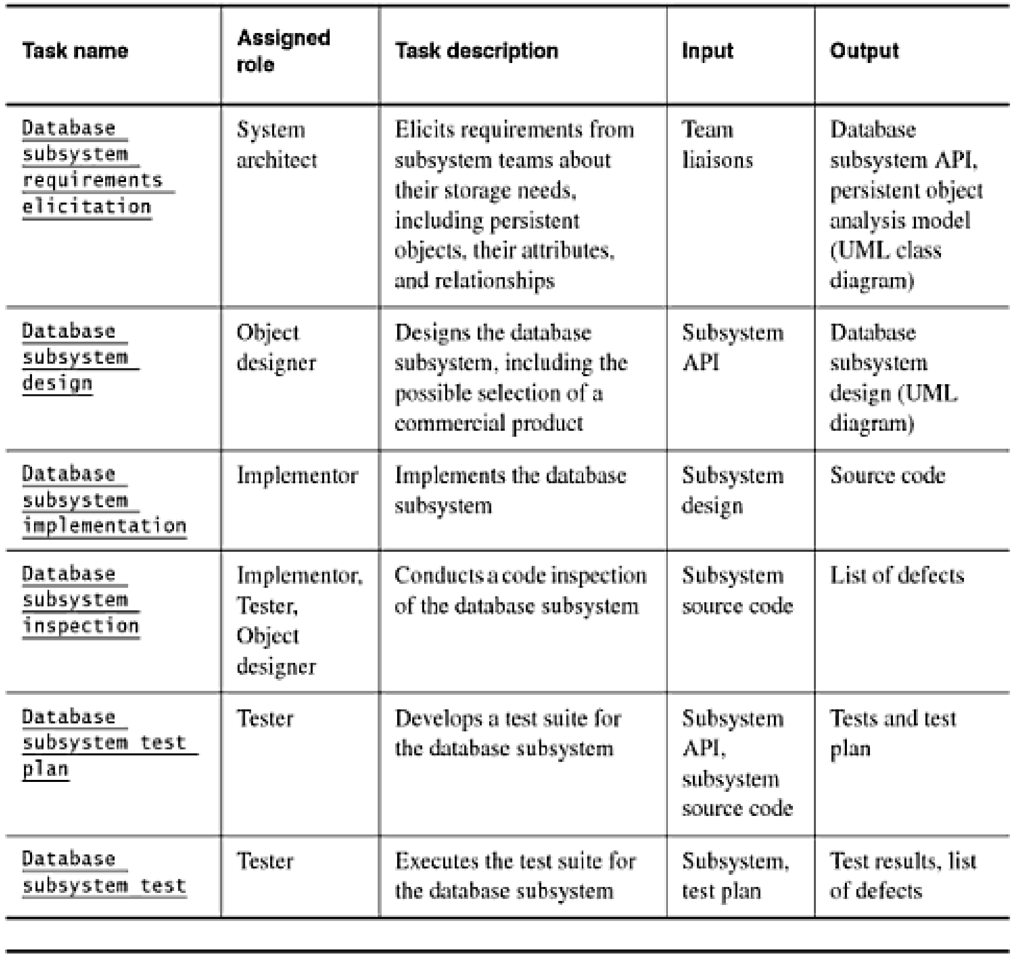
estimated effort = NOP/PROD

In more advanced COCOMO II models, a variety of scale factors, cost drivers, and adjustment procedures are required.

3.2.2 Scheduling:

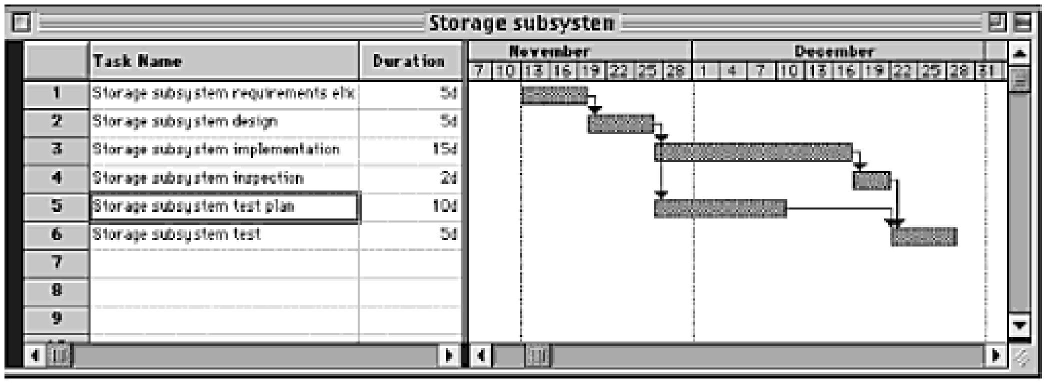
A schedule is the mapping of tasks onto time: each task is assigned start and end times. This allows us to plan the deadlines for individual deliverables. The two most often used diagrammatic notations for schedules are PERT (Program Evaluation Review

Technique) and Gantt charts. A **Gantt chart** is a compact way to present the schedule of a software project along the time axis. A Gantt chart is a bar graph on which the horizontal axis represents time and the vertical axis lists the different tasks to be done. Tasks are represented as bars whose length corresponds to the planned duration of the task. A schedule for the database subsystem example is represented as a Gantt chart in the following figure. Before the tasks for the database subsystem is given in a table.



Task name	Assigned role	Task description	Input	Output
Database subsystem requirements elicitation	System architect	Elicits requirements from subsystem teams about their storage needs, including persistent objects, their attributes, and relationships	Team liaisons	Database subsystem API, persistent object analysis model (UML class diagram)
Database subsystem design	Object designer	Designs the database subsystem, including the possible selection of a commercial product	Subsystem API	Database subsystem design (UML diagram)
Database subsystem implementation	Implementor	Implements the database subsystem	Subsystem design	Source code
Database subsystem inspection	Implementor, Tester, Object designer	Conducts a code inspection of the database subsystem	Subsystem source code	List of defects
Database subsystem test plan	Tester	Develops a test suite for the database subsystem	Subsystem API, subsystem source code	Tests and test plan
Database subsystem test	Tester	Executes the test suite for the database subsystem	Subsystem, test plan	Test results, list of defects

Table 2.4: Examples of tasks for the realization of the database subsystem



	Task Name	Duration
1	Storage subsystem requirements elic	5d
2	Storage subsystem design	5d
3	Storage subsystem implementation	15d
4	Storage subsystem inspection	2d
5	Storage subsystem test plan	10d
6	Storage subsystem test	5d
7		
8		
9		

Figure 2.6: An example of schedule for the database subsystem (Gantt chart)

A **PERT chart** represents a schedule as an acyclic graph of tasks. The following figure is a PERT chart for the database subsystem schedule. The planned start and duration of the tasks are used to compute the critical path, which represents the shortest possible path through the graph. The length of the critical path corresponds to the shortest possible schedule, assuming sufficient resources to accomplish, in parallel, tasks that are independent. Moreover, tasks on the critical path are the most important, as a delay in any of these tasks will result in a delay in the overall project. The tasks and bars represented in thicker lines belong to the critical path.

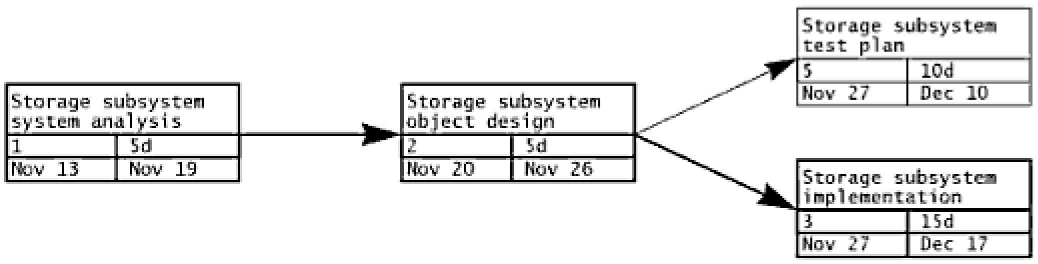

Storage subsystem system analysis
1
5d
Nov 13
Nov 19
Storage subsystem object design
2
5d
Nov 20
Nov 26
Storage subsystem test plan
5
10d
Nov 27
Dec 10
Storage subsystem implementation
3
15d
Nov 27
Dec 17

Figure 2.7: Schedule for the database subsystem (PERT chart)

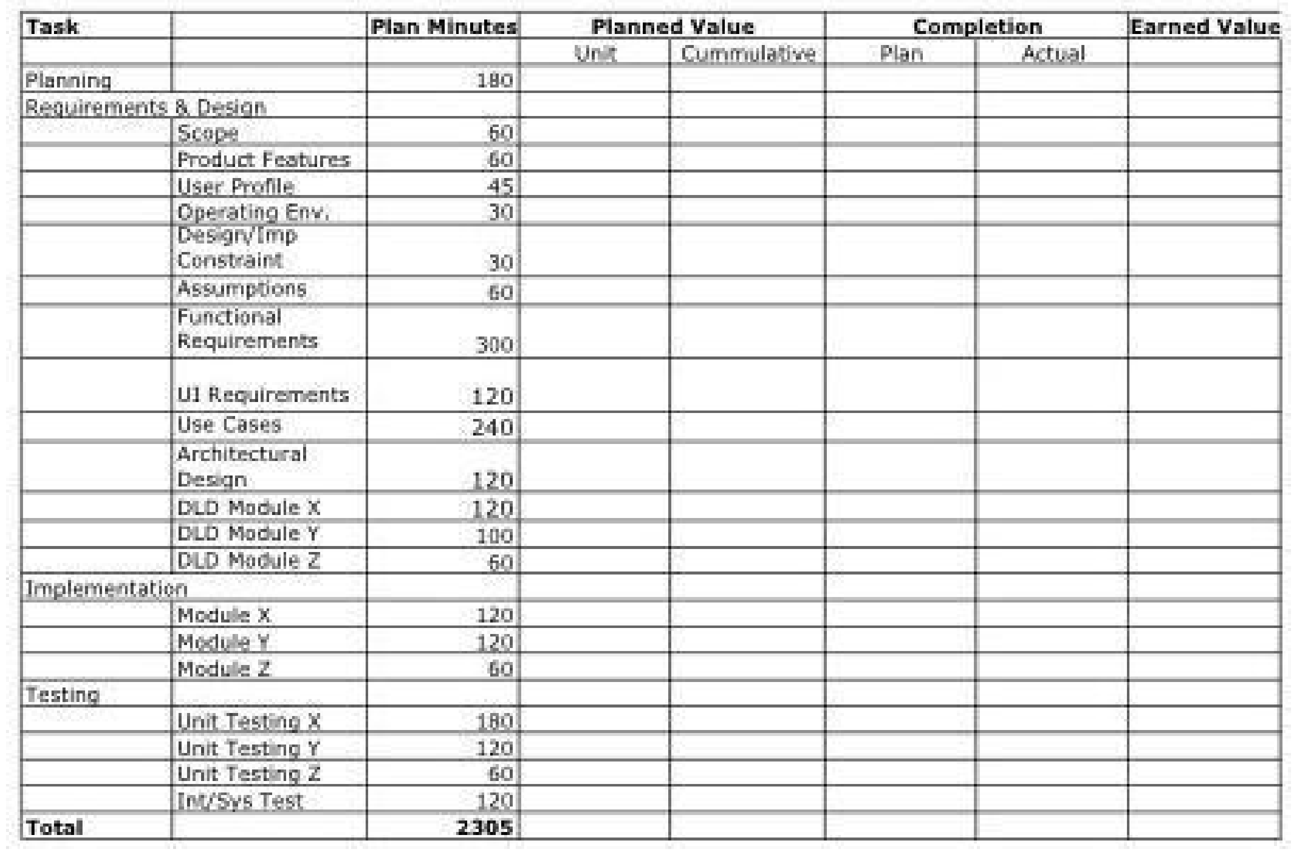
3.2.3 Tracking

As the project progresses, the project manager understands the activities to be completed and milestones to be tracked and controlled with the help of project schedule. Tracking of project schedule is done in several ways.
* Conducting periodic meetings with team members: - By conducting periodic meetings, the project manager is able to distinguish between completed and uncompleted activities or those that are yet to start. In addition, the project manager considers the problems in the project as reported by the team members.
* Assessing the results of reviews: - Software reviews are conducted when one or more activities of the project are complete or when a particular development phase is complete. The purpose of conducting reviews is to check whether the software is developed according to user requirements or not.
* Determining the milestones: - Milestones indicating the expected output par described. These milestones check the status of a project by comparing the progress of activities with the estimated end date of the project.
* Using earned value analysis to determine the progress of the project: - The progress of the project is determined quantitatively by the earned value analysis technique. This technique provides an estimate for every task without considering its type and the total hours required to accomplish the project. Based on this estimation, each activity is given an earned value, which is a measure of progress and describes the percentage of the activities that have been completed.

<u>Earned value analysis:</u>

The earned value system provides a common value scale for every task, regardless of the type of work being performed. The total hours to do the whole project are estimated, and every task is given an earned value based on its estimated percentage of the total. Basically, earned value is useful as it provides a quantitative technique of assessing progress on the project as a whole (even when tasks are completed in an order that differs from the original schedule).

Consider the following example project:



Task		Plan Minutes	Planned Value		Completion		Earned Value
			Unit	Cummulative	Plan	Actual	
Planning		180					
Requirements & Design							
	Scope	60					
	Product Features	60					
	User Profile	45					
	Operating Env.	30					
	Design/Imp Constraint	30					
	Assumptions	60					
	Functional Requirements	300					
	UI Requirements	120					
	Use Cases	240					
	Architectural Design	120					
	DLD Module X	120					
	DLD Module Y	100					
	DLD Module Z	60					
Implementation							
	Module X	120					
	Module Y	120					
	Module Z	60					
Testing							
	Unit Testing X	180					
	Unit Testing Y	120					
	Unit Testing Z	60					
	Int/Sys Test	120					
Total		**2305**					

Figure 2.8: Example project for illustrating earned value analysis

Calculating the planned value:

The planned value for a task is the estimated percentage of the planned duration of that task of the total planned duration of all tasks.

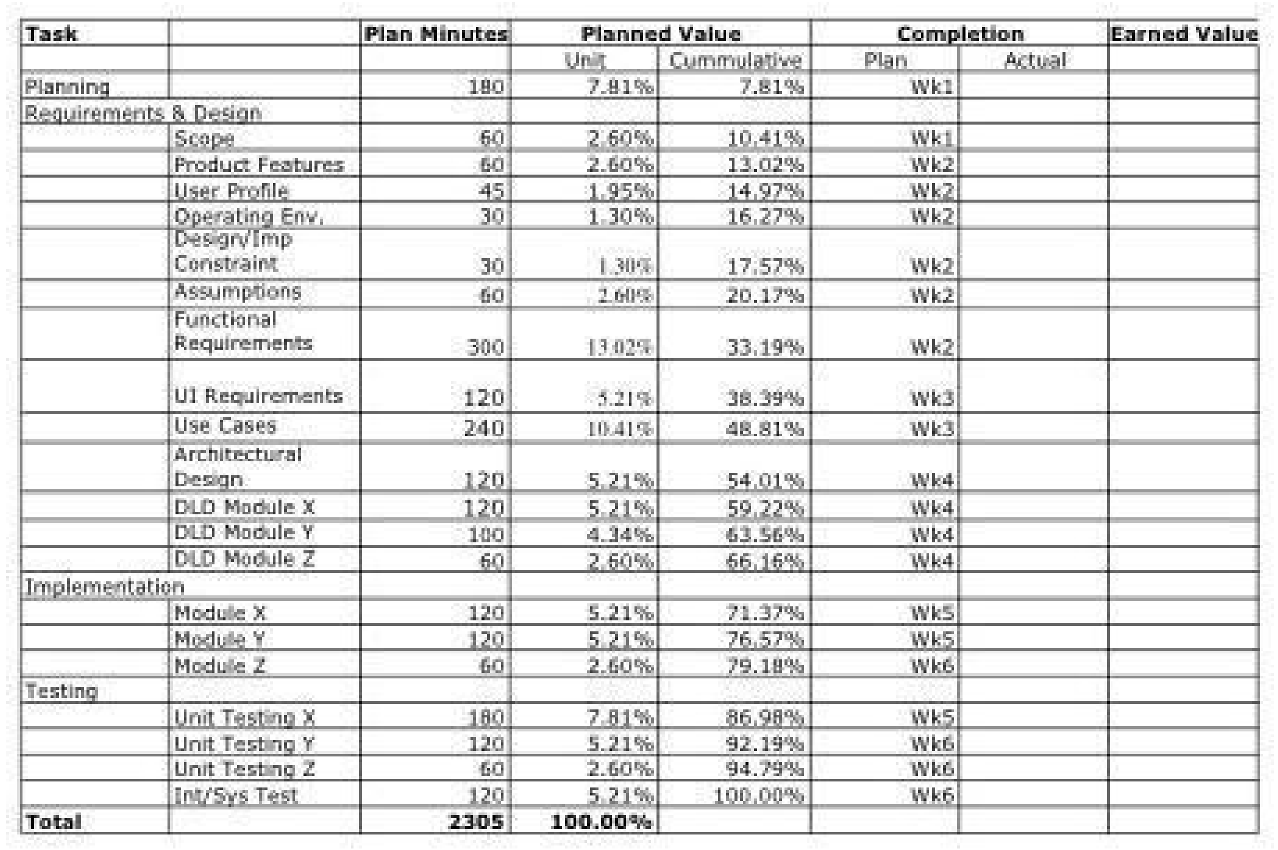
Planned Value T = DT / TD

where T is a task, DT is the duration of T and TD is the total planned duration of all tasks.

Applying this formula the planned value for the planning task is calculated from the above table. It is estimated that planning will take 180 minutes. The total duration of all of our estimates for development is 2305 minutes. The planned value for the planning task by the formula above is, therefore, given by

Planned Value = 180/2305 ~ 7.81%

This value is written in the planned value (unit) column for the planning task. The same is done for all of the tasks in the plan. The total of all of the unit planned values will be 100% if this is done correctly.



Task		Plan Minutes	Planned Value		Completion		Earned Value
			Unit	Cummulative	Plan	Actual	
Planning		180	7.81%	7.81%	Wk1		
Requirements & Design							
	Scope	60	2.60%	10.41%	Wk1		
	Product Features	60	2.60%	13.02%	Wk2		
	User Profile	45	1.95%	14.97%	Wk2		
	Operating Env.	30	1.30%	16.27%	Wk2		
	Design/Imp Constraint	30	1.30%	17.57%	Wk2		
	Assumptions	60	2.60%	20.17%	Wk2		
	Functional Requirements	300	13.02%	33.19%	Wk2		
	UI Requirements	120	5.21%	38.39%	Wk3		
	Use Cases	240	10.41%	48.81%	Wk3		
	Architectural Design	120	5.21%	54.01%	Wk4		
	DLD Module X	120	5.21%	59.22%	Wk4		
	DLD Module Y	100	4.34%	63.56%	Wk4		
	DLD Module Z	60	2.60%	66.16%	Wk4		
Implementation							
	Module X	120	5.21%	71.37%	Wk5		
	Module Y	120	5.21%	76.57%	Wk5		
	Module Z	60	2.60%	79.18%	Wk6		
Testing							
	Unit Testing X	180	7.81%	86.98%	Wk5		
	Unit Testing Y	120	5.21%	92.19%	Wk6		
	Unit Testing Z	60	2.60%	94.79%	Wk6		
	Int/Sys Test	120	5.21%	100.00%	Wk6		
Total		**2305**	**100.00%**				

Figure 2.9: Calculating the planned value

Consider that one wishes to document when he/she expects each task to be completed. In this example, this has been done by filling in the week number in the planned completion column. In order to track progress against this plan, one should have an idea of what percentage complete he/she expects to be at the end of each week. A cumulative total of the planned value is kept in the cumulative planned value column. This column can be used to determine what percentage of the project will be complete by the end of week 3, for example.

It can be seen that the sum of all of the planned values for tasks planned to be complete before the end of week 3 is approx. 48.81% (because the tasks are in order, the cumulative planned value for the last week 3 entry gives us this number - if the tasks are listed unordered by completion date, the sum must be calculated by adding the planned values of all tasks due to be completed before the date in question).

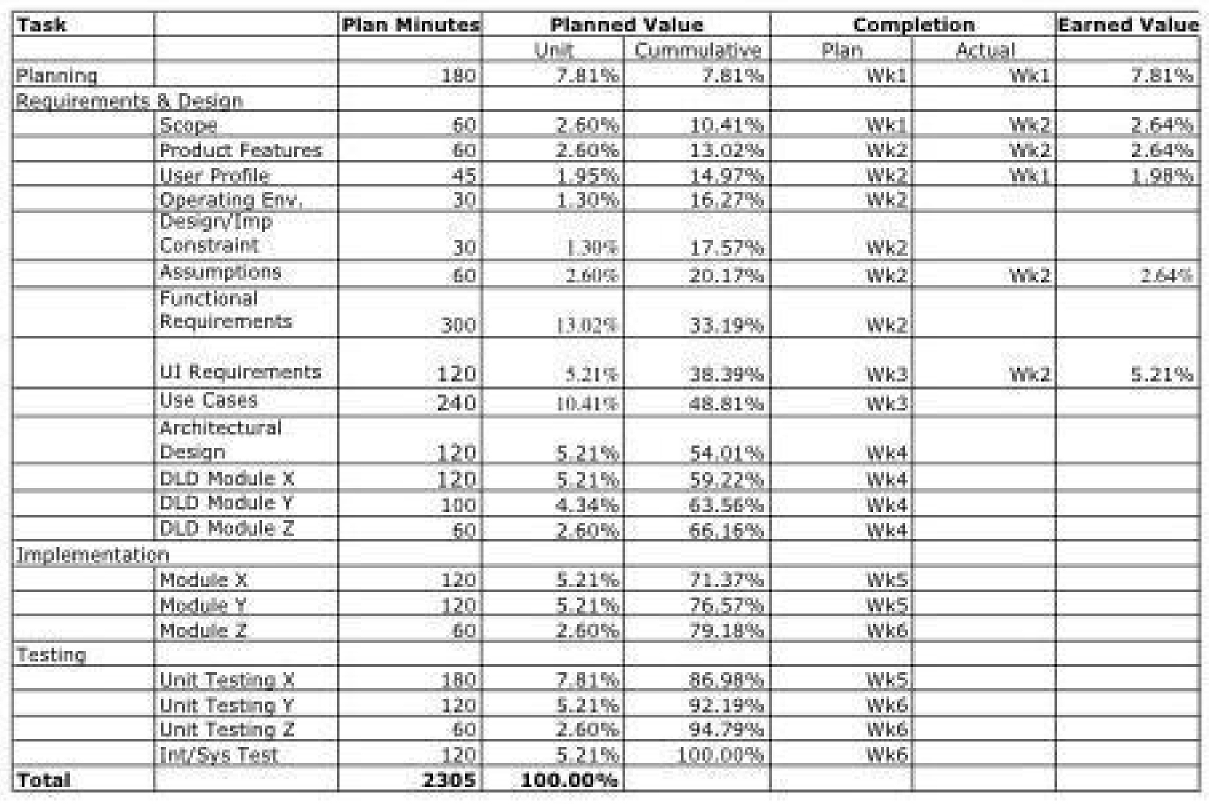
Tracking a project with earned value:

Consider that the end of the second week is being reached on this project and one need to know the progress toward completion on time. Also consider that everything has not been done in the order in which it was planned, so this may not be easy to guess at. In this case the concept of an earned value can be used. Earned value is assigned only to completed tasks. If a task is completed with a planned value of p%, then the earned value for that task is p%. Incomplete tasks have no earned value.

In the example below (shown by the following figure), compiled at the end of the second week, it is observed that the group has completed the following tasks with their associated earned value:

- Planning – 7.81%
- Scope – 2.64%
- Product Features – 2.64%
- User Profile – 1.98%
- Assumptions – 2.64%
- UI requirements – 5.21%

By summing the earned values for the completed tasks it is determined that at the end of week 2, the cumulative earned value (the percent complete) is 7.81+2.64+2.64+1.98+2.64+5.21=22.91%. Now, knowing the earned value the project group can determine if they are ahead, behind on schedule to completion.



Task		Plan Minutes	Planned Value		Completion		Earned Value
			Unit	Cummulative	Plan	Actual	
Planning		180	7.81%	7.81%	Wk1	Wk1	7.81%
Requirements & Design							
	Scope	60	2.60%	10.41%	Wk1	Wk2	2.64%
	Product Features	60	2.60%	13.02%	Wk2	Wk2	2.64%
	User Profile	45	1.95%	14.97%	Wk2	Wk1	1.98%
	Operating Env.	30	1.30%	16.27%	Wk2		
	Design/Imp Constraint	30	1.30%	17.57%	Wk2		
	Assumptions	60	2.60%	20.17%	Wk2	Wk2	2.64%
	Functional Requirements	300	13.02%	33.19%	Wk2		
	UI Requirements	120	5.21%	38.39%	Wk3	Wk2	5.21%
	Use Cases	240	10.41%	48.81%	Wk3		
	Architectural Design	120	5.21%	54.01%	Wk4		
	DLD Module X	120	5.21%	59.22%	Wk4		
	DLD Module Y	100	4.34%	63.56%	Wk4		
	DLD Module Z	60	2.60%	66.16%	Wk4		
Implementation							
	Module X	120	5.21%	71.37%	Wk5		
	Module Y	120	5.21%	76.57%	Wk5		
	Module Z	60	2.60%	79.18%	Wk6		
Testing							
	Unit Testing X	180	7.81%	86.98%	Wk5		
	Unit Testing Y	120	5.21%	92.19%	Wk6		
	Unit Testing Z	60	2.60%	94.79%	Wk6		
	Int/Sys Test	120	5.21%	100.00%	Wk6		
Total		**2305**	**100.00%**				

Figure 2.10: Project status at the end of second week

Looking at the cumulative planned value in the table, it is observed that at the end of week 2 to be on schedule, the group should have been 33.19% complete. By the above calculation that they are actually only 22.91% complete. Hence it is noted that, the project is falling behind schedule. Knowing this, allows the project team to take action. Such action may involve: adjusting work practices, negotiating with the client for an extension to the deadline etc.

3.2.4 Discuss schedule and cost slippage.

There is a direct re;lation betewwn delay in schedule and cost. As the schedule gets delayed , the cost of the project increases.

4. Objective Questions:

1. Software project estimation techniques can be broadly classified under which of the following:
   a. automated processes
   b. decomposition techniques
   c. empirical models
   d. regression models
   e. both b and c
2. The size estimate for the software product to be built must be based on a direct measure like LOC
   a. True
   b. False
3. Problem-based estimation is based on problem decomposition which focuses on
   a. information domain values
   b. project schedule
   c. software functions
   d. process activities
   e. both a and c
4. LOC-based estimation techniques require problem decomposition based on
   a. information domain values
   b. project schedule
   c. software functions
   d. process activities
5. FP-based estimation techniques require problem decomposition based on

a. information domain values
b. project schedule
c. software functions
d. process activities

6. Process-based estimation techniques require problem decomposition based on
    a. information domain values
    b. project schedule
    c. software functions
    d. process activities
    e. both c and d
7. Unlike a LOC or function point each person's "use-case" is exactly the same size
    a. True
    b. False
8. When agreement between estimates is poor the cause may often be traced to 'inadequately defined project scope' or 'inappropriate productivity data'
    a. True
    b. False
9. Empirical estimation models are typically based on
    a. Expert judgment based on past project experiences
    b. refinement of expected value estimation
    c. regression models derived from historical project data
    d. trial and error determination of the parameters and coefficients
10. COCOMO II is an example of a suite of modern empirical estimation models that require sizing information expressed as:
    a. function points
    b. lines of code
    c. object points
    d. any one of the above
11. Function points are of no use in developing estimates for object-oriented software.
    a. True
    b. False
12. The objective of software project planning is to
    a. convince the customer that a project is feasible.
    b. make use of historical project data.
    c. enable a manager to make reasonable estimates of cost and schedule.
    d. determine the probable profit margin prior to bidding on a project.
13. Since project estimates are not completely reliable, they can be ignored once a software development project begins.
    a. True
    b. False

Answers: 1. e 2. b 3. e 4. c 5. a 6. e 7. b 8. a 9. c 10. d 11. b 12. c 13. b

5. Subjective Questions:

1. What are the various activities during software project planning? Describe them in detail?
2. Explain in detail about scheduling a project and tracking its progress?
3. Write a note on COCOMO model?
4. Explain the following estimation techniques:
   a) LOC b) FP
2. Write notes on product metrics?

6. Questions:

1. Explain the COCOMO used for software estimation.
Ans:
Simple on-line cost model for estimating the number of person-months required to develop software. The model also estimates the development schedule in months and produces an effort and schedule distribution by major phases. This is based on Barry Boehm's Constructive Cost Model (COCOMO).

This is top level model. Basic COCOMO, is applicable to the large majority of software projects. The model estimates cost using one of three different development modes: organic, semidetached and embedded.

Organic: Small projects, in-house environment

Semidetached: intermediate, mixture of organic and embedded product generally extends 300(KDSI) thousand delivered source instructions.

Embedded: coupled complex hardware, software regulations
Eg: electronic fund transfer, Air traffic control

2. Write two advantages of PERT chart.
Ans:
- Makes visible dependencies (precedence rates) between WBS elements
- Facilitates identification of critical path and makes this visible
- Facilitates early start, late start, slack for each activity

- Provides for potentially reduced project duration due to better understanding of dependencies leading to improved overlapping of activities and tasks where feasible

3. Explain how project scheduling and tracking is done for a software development project.

Ans:
A schedule is the mapping of tasks onto time: each task is assigned start and end times. This allows us to plan the deadlines for individual deliverables. The two most often used diagrammatic notations for schedules are PERT (Program Evaluation Review Technique) and Gantt charts.

As the project progresses, the project manager understands the activities to be completed and milestones to be tracked and controlled with the help of project schedule. Tracking of project schedule is done in several ways.

- Conducting periodic meetings with team members
- Assessing the results of reviews
- Determining the milestones
- Using earned value analysis to determine the progress of the project