UNIT V

PROJECT MANAGEMENT

Software Project Management: Estimation – LOC, FP Based Estimation, Make/Buy Decision COCOMO I & II Model – Project Scheduling – Scheduling, Earned Value Analysis Planning – Project Plan, Planning Process, RFP Risk Management – Identification, Projection - Risk Management-Risk Identification-RMMM Plan-CASE TOOLS

5.1 SOFTWARE PROJECT MANAGEMENT:

Effective software project management focuses on the four P's:

- People
- Product
- Process
- Project

5.1.1 People

- The "**people factor"** is so important that the Software Engineering Institute has developed a People Capability Maturity Model (People-CMM), in recognition of the fact that "every organization needs to continually improve its ability to attract, develop, motivate, organize, and retain the workforce needed to accomplish its strategic business objectives".
- The people capability maturity model defines the following key practice areas for software people:
- staffing
- communication and coordination
- work environment
- performance management
- training
- compensation
- competency analysis and development
- career development
- workgroup development
- team/culture development

5.1.2 Product

Before a project can be planned

- product objectives and scope should be established
- alternative solutions should be considered
- Technical and management constraints should be identified.

- This activity begins as part of the system engineering or business process engineering and continues as the first step in software requirements engineering.
- Objectives identify the overall goals for the product (from the stakeholders' points of view) without considering how these goals will be achieved.
- Scope identifies the primary data, functions, and behaviors that characterize the product, and more important, attempts to bound these characteristics in a quantitative manner.

5.1.3The Process

- A software process provides the framework from which a comprehensive plan for software development can be established.
- A small number of framework activities are applicable to all software projects, regardless of their size or complexity.
- A number of different task sets—tasks, milestones, work products, and quality assurance points—enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team.
- Finally, umbrella activities—such as software quality assurance, software configuration management, and measurement—overlay the process model.
- Umbrella activities are independent of any one framework activity and occur throughout the process.

5.1.4 The Project:

- In order to manage a successful software project, you have to understand what can go wrong so that problems can be avoided.
- To avoid project failure, a software project manager and the software engineers who build the product must avoid a set of common warning signs, understand the critical success factors that lead to good project management, and develop a common sense approach for planning, monitoring, and controlling the project.

5.2 SOFTWARE PROJECT ESTIMATION

- ✓ Software project estimation is a form of problem solving, and in most cases, the problem to be solved (i.e., developing a cost and effort estimate for a software project) is too complex to be considered in one piece.
- ✓ The decomposition approach was discussed from two different points of view:

  i) Decomposition of the problem and
  ii) Decomposition of the process.

- ✓ Estimation uses one or both forms of partitioning.

5.2.1 Software Sizing

The accuracy of a software project estimate is predicated on a number of things:
(1) The degree to which you have properly estimated the size of the product to be built
(2) The ability to translate the size estimate into human effort, calendar time, and dollars (a function of the availability of reliable software metrics from past projects);
(3) The degree to which the project plan reflects the abilities of the software team
(4) The stability of product requirements and the environment that supports the software engineering effort.
The sizing can be estimated using two approaches:

- If a **direct approach** is taken, size can be measured in **lines of code (LOC).**
- If an **indirect approach** is chosen, size is represented as **function points (FP).**

Putnam and Myers [Put92] suggest four different approaches to the sizing problem:

- **Fuzzy logic sizing.**
  - This approach uses the approximate reasoning techniques that are the cornerstone of fuzzy logic.
  - To apply this approach, the planner must identify the type of application, establish its magnitude on a qualitative scale, and then refine the magnitude within the original range.
- **Function point sizing**. The planner develops estimates of the information domain characteristics.
- **Standard component sizing.** Software is composed of a number of different "standard components" that are generic to a particular application area.

For example:

- ✓ The standard components for an information system are subsystems, modules, screens, reports, interactive programs, batch programs, files, LOC, and object-level instructions.
- ✓ The project planner estimates the number of occurrences of each standard component and then uses historical project data to estimate the delivered size per standard component.

Change sizing.

- This approach is used when a project encompasses the use of existing software that must be modified in some way as part of a project.
- The planner estimates the number and type (e.g., reuse, adding code, changing code, and deleting code) of modifications that must be accomplished.

5.2.2 Problem-Based Estimation:

LOC and FP data are used in two ways during software project estimation:
(1) As estimation variables to "size" each element of the software and
(2) As baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

LOC or FP is then estimated for each function.

- **Baseline productivity metrics** 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.
- Using historical data the project planner expected value by considering following variables.

1. Optimistic
2. Most likely
3. Pessimistic

- 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(sopt), most likely (sm), and pessimistic (spess) estimates. For example,

$$S = \frac{S_{opt} + 4S_m + S_{pess}}{6}$$

Where,

- (size) S
- optimistic(sopt),
- most likely (sm), and
- pessimistic (spess)

LOC based Estimation:

- Size oriented measure is derived by considering the size of software that has been produced.
- The organization builds a simple record of size measure for the software projects. It is built on past experiences of organizations.
- It is a direct measure of software

Project	LOC	Effort	Cost($)	Doc.(pgs.)	Errors	Defects	People
ABC	10,000	20	170	400	100	12	4
PQR	20,000	60	300	1000	129	32	6
XYZ	35,000	65	522	1290	280	87	7
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 5.5.1 Size measure

- A simple set of size measure that can be developed is as given below :
  - Size = Kilo Lines of Code (KLOC)
  - Effort = Person/month
  - Productivity = KLOC/person-month
  - Quality = Number of faults/KLOC
  - Cost = $/KLOC
  - Documentation = Pages of documentation/KLOC
- The size measure is based on the lines of code computation. The lines of code is defined as one line of text in a source file.
- While counting the lines of code the simplest standard is :
  - Don't count blank lines.
  - Don't count comments.
  - Count everything else.
- The size oriented measure is not universally accepted method.

Advantages

1. Artifact of software development which is easily counted.
2. Many existing methods use LOC as a key input.
3. A large body of literature and data based on LOC already exists.

Disadvantages

1. This measure is dependent upon the programming language.
2. This method is well designed but shorter program may get suffered.
3. It does not accommodate non procedural languages.
4. In early stage of development it is difficult to estimate LOC.

Example of LOC based Estimation:

Consider an ABC project with some important modules such as

1. User interface and control facilities
2. 2D graphics analysis
3. 3D graphics analysis
4. Database management
5. Computer graphics display facility
6. Peripheral control function
7. Design analysis models

Estimate the project in based on LOC

Solution:

For estimating the given application we consider each module as separate function and corresponding lines of code can be estimated in the following table as

Function	Estimated LOC
User interface and control facilities (UICF)	2500
Two-dimensional geometric analysis (2DGA)	5600
Three-dimensional geometric analysis (3DGA)	6450
Database management (DBM)	3100
Computer graphics display facilities (CGDF)	4740
Peripheral control function (PCF)	2250
Design analysis modules (DAM)	7980
Estimated lines of code	**32620**

- Expected LOC for 3D Geometric analysis function based on three point estimation is -
  - Optimistic estimation 4700
  - Most likely estimation 6000
  - Pessimistic estimation 10000

$$S = [S_{opt} + (4 * S_m) + S_{pess}]/6$$

Expected value = [4700 + (4 * 6000) + 10000] /6 → 6450

- A review of historical data indicates -
  1. Average productivity is 500 LOC per month
  2. Average labor cost is $6000 per month

Then cost for lines of code can be estimated as

cost/LOC = (6000/500) = $12

By considering total estimated LOC as 32620

- Total estimated project cost = (32620*12) = $391440
- Total estimated project effort = (32620/500) = 65 Person-months

5.3 AN EXAMPLE OF FP-BASED ESTIMATION

- Decomposition for FP-based estimation focuses on information domain valuesrather than software functions.

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

- Each of the complexity weighting factors is estimated, and the value adjustmentfactor is computed.

Factor	Value
Backup and recovery	4
Data communications	2
Distributed processing	0
Performance critical	4
Existing operating environment	3
Online data entry	4
Input transaction over multiple screens	5
Master files updated online	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
Value adjustment factor	**1.17**

Finally, the estimated number of FP is derived:

$$FP_{estimated} = \text{count total} \times [0.65 + 0.01 \times \Sigma(F_i)] = 375$$

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

5.4 THE MAKE/BUY DECISION

Software engineering managers are faced with a make/buy decision that can be further complicated by a number of acquisition options:

(1) Software may be purchased (or licensed) off-the-shelf,

(2) "full-experience" or"partial-experience" software components may be acquired andthen modified and integrated to meet specific needs, or

(3) Software may be custombuilt by an outside contractor to meet the purchaser's specifications.

5.4.1 Creating A Decision Tree

For example, figure 5.1depicts a decision tree for a softwarebasedsystem x.

In this case, the software engineering organization can

(1) buildsystem x from scratch,

(2) reuse existing partial-experience components to constructthe system,

(3) buy an available software product and modify it to meet local needs,or

(4) contract the software development to an outside vendor.

- The project planner estimates that a difficult development effort will cost $450,000.
- A "simple" development effort is estimated to cost $380,000. The expected value for cost, computed along any branch of the decision tree, is

$$\text{Expected cost} = \Sigma \text{ (path probability)}_i \times \text{(estimated path cost)}_i$$

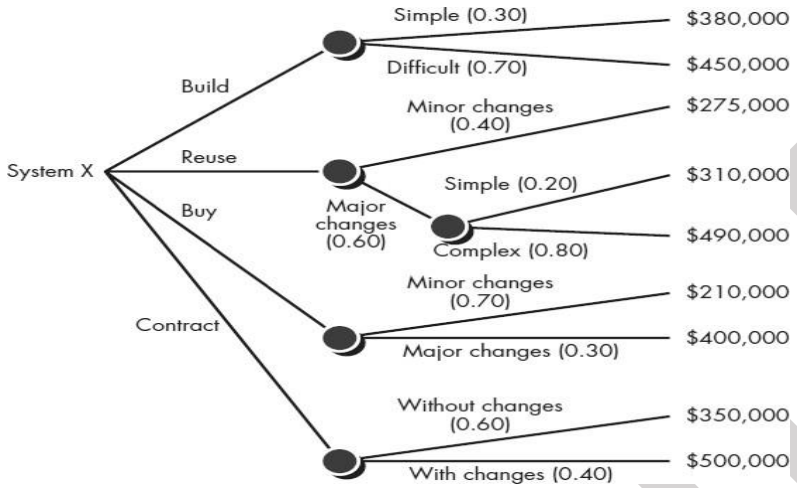


Figure 5.1 A Decision Tree To Support The Make/Buy Decision

- Following other paths of the decision tree, the projected costs for reuse, purchase, and contract, under a variety of circumstances, are also shown. The expected costs for these paths are

  $\text{Expected cost}_{reuse} = 0.40 (\$275K) + 0.60 [0.20 (\$310K) + 0.80 (\$490K)] = \$382K$

  $\text{Expected cost}_{buy} = 0.70 (\$210K) + 0.30 (\$400K) = \$267K$

  $\text{Expected cost}_{contract} = 0.60 (\$350K) + 0.40 (\$500K) = \$410K$

- Based on the probability and projected costs that in Figure , the lowest expected cost is the **"buy" option.**

5.4.2 Outsourcing

- Outsourcing is extremely simple. Software engineering activities are contracted to a third party who does the work at lower cost and, hopefully, higher quality.
- The decision to outsource can be either **strategic or tactical.**
- At the **strategic level,** business managers consider whether a significant portion of all software work can be contracted to others.
- At the **tactical level,** a project manager determines whether part or all of a project can be best accomplished by subcontracting the software work.

Benefits of outsourcing:

- If the software is outsourced then people and resource utilization can be reduced.

Drawbacks of outsourcing:

- The trend of outsourcing will be continued in software industry in order to survive in competitive world.

5.5 THE COCOMO I MODEL

- COCOMO is one of the most widely used software estimation models in the world. This model is developed in 1981 by Barry Boehm to give an estimate of the number of man-months it will take

to develop a software product. COCOMO predicts the efforts and schedule of a software product based on size of the software. COCOMO stands for "ConstructiveCostMOdel".

- COCOMO has three different models that reflect the complexity –
- ✓ Basic model
- ✓ Intermediate model
- ✓ Detailed model.
- Similarly there are three classes of software projects.

1) Organic mode: In this mode, relatively small, simple software projects with a small team are handled. Such a team should have good application experience to less rigid requirements.

2) Semi-detached projects: In this class an intermediate projects in which teams with mixed experience level are handled. Such projects may have mix of rigid and less than rigid requirements.

3) Embedded projects: In this class, projects with tight hardware, software and operational constraints are handled.

Let us understand each model in detail.

1) **Basic model:** The basic COCOMO model estimates the software development effort using only Lines of Code. Various equations in this model are –

$$E = a_b (KLOC)^{b_b}$$

$$D = C_b (E)^{d_b}$$

$$P = E/D$$

Where E is the effort applied in person-months.

D is the development time in chronological months.

KLOC means kilo line of code for the project.

P is total number of persons required to accomplish the project.

The coefficients a_b, b_b, c_b, $d_b$for three models are as given below.

Software projects	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Figure 5.2 Coefficients for three models

Merits of basic COCOMO model

Basic COCOMO model is good for quick, early, rough order of magnitude estimates of software project.

Limitations of basic model

1. The accuracy of this model is limited because it does not consider certain factors for cost estimation of software. These factors are hardware constraints, personal quality, and experience, modern techniques and tools.

2. The estimates of COCOMO model are within a factor of 1.3 only 29 % of the time and within the factor of 2 only 60 % of time.

Example

Consider a software project using semi-detached mode with 30,000 lines of code. We will obtain estimation for this project as follows

i) Effort estimation

$$E = a_b (KLOC)^{bb}$$

$$E = 3.0\ (30)^{1.12} \quad \text{where lines of code} = 30000 = 30\ KLOC$$

$$E = 135\ \text{person-month}$$

ii) Duration estimation

$$D = C_b (E)^{d_b}$$

$$= 2.5\ (135)^{0.35}$$

$$D = 14\ \text{months}$$

iii) Persons estimation

$$P = E/D$$

$$= 135/14$$

$$P = 10\ \text{persons approximately}$$

2) Intermediate model

This is an extension of Basic COCOMO model. This estimation model makes use set of **"Cost driver attributes"** to compute the cost of software.

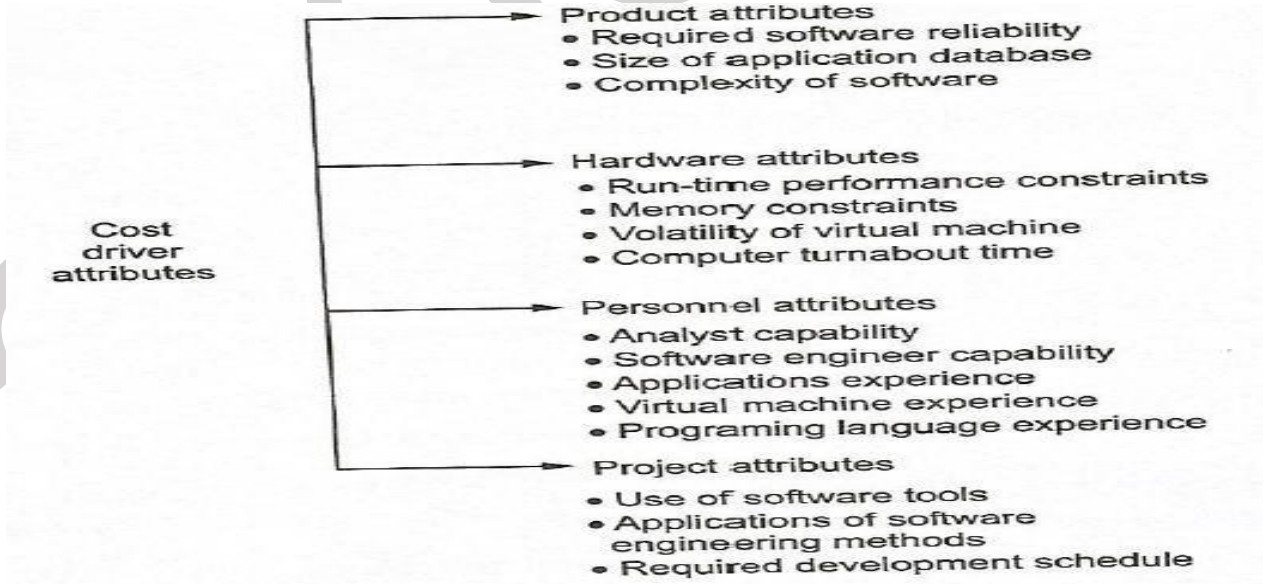
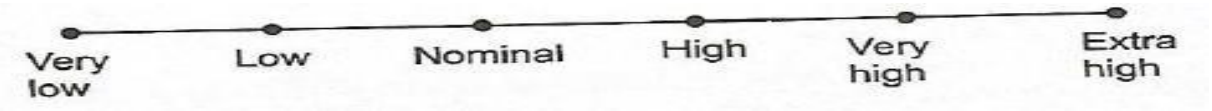


Figure 5.3 Cost driver Attributes

- Now these 15 attributes get a 6-point scale ranging from "very low" to "extra high'.

- These ratings can be viewed as



- The effort multipliers for each cost driver attribute are as given in following table. The product of all effort multipliers result in "**Effort Adjustment Factor" (EAF).**
- The formula for effort calculation can be-

$E=a_i(KLOC) * EAF$ person months

- The values for a_i and b_i for various class of software projects are-

Software project	a_i	b_i
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

Figure 5.4 Various Classes of software projects

- The duration and person estimate is same as in basic COCOMO model. i.e.

$D = c_b (E)^{d_b}$ months i.e. use values of c_b and d_b

$P = E/D$ persons

Merits of intermediate model

1. This model can be applied to almost entire software product for easy and rough cost estimation during early stage.
2. It can also be applied at the software product component level for obtaining more accurate cost estimation.

Limitations of intermediate model

1. The estimation is within 20 % of actual 68 % of the time.
2. The effort multipliers are not dependent on phases.
3. A product with many components is difficult to estimate.

Example

Considera project having 30,000 lines of code which is embedded software with critical area hence reliability is high. The estimation can be

$$E = a_i (KLOC)^{b_i} \cdot EAF$$

As reliability is high, EAF = 1.15 (product attribute)

$a_i = 2.8$, $b_i = 1.20$ } for embedded software

$$E = 2.8 (30)^{1.20} * 1.15$$

$$= 191 \text{ person-month}$$

$$D = c_b (E)^{d_b} = 2.5 (191)^{0.32}$$

$$= 13 \text{ months approximately}$$

P = E/ D

= 191/13

P = 15 persons approximately

3) Detailed COCOMO model

- The detailed model uses the same equations for estimation as the Intermediate Model. But detailed model can estimate the effort (E), duration (D) and persons (P) of each of development phases, subsystems, modules.
- The experimentation with different development strategies is allowed in this model.
- Four phases used in detailed COCOMO model are
    1. Requirements Planning and Product Design (RPD)
    2. Detailed Design (DD)
    3. Code and Unit Test (CUT)
    4. Integrate and Test (IT)
- The effort multipliers for detailed COCOMO are

Phases	Very low	Low	Nominal	High	Very high
RPD	1.80	0.85	1.00	0.75	0.55
DD	1.35	0.85	1.00	0.90	0.75
CUT	1.35	0.85	1.00	0.90	0.75
IT	1.50	1.20	1.00	0.85	0.70

Figure 5.5 Effort Multipliers for detailed COCOMO Model

- Using these detailed cost drivers, an estimate is determined for each phase of the life cycle.

5.6 THE COCOMO II MODEL

- COCOMO, for Constructive Cost Model
- 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 andsystem interaction, assessment of performance, and evaluation of technologymaturity are paramount.
- **Early design stage model**. Used once requirements have been stabilized andbasic software architecture has been established.
- **Post-architecture-stage model**. Used during the construction of the 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

- 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.
- The object point count is then determined by multiplying the original number of object instances by the weighting factor in the figure 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:

$$\text{NOP} = (\text{object points}) \times [(100 - \%\text{reuse})/100]$$

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.

$$\text{PROD} = \frac{\text{NOP}}{\text{person-month}}$$

- Figure 5.6 presents the productivity rate 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

Figure5.6 Productivity rate for developer experience and environment maturity

PRODUCTIVITY RATE FOR OBJECT POINTS.

- Once the productivity rate has been determined, an estimate of project effort is computed using

$$\text{Estimated effort} = \frac{\text{NOP}}{\text{PROD}}$$

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

5.7 SOFTWARE PROJECT SCHEDULING

Software project scheduling is an action that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks.

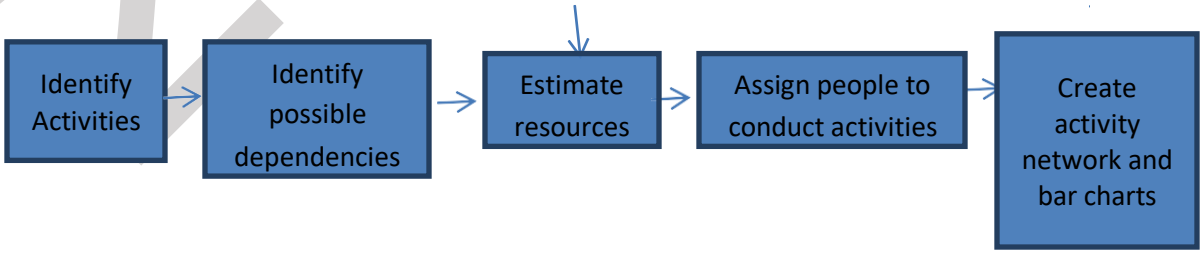


Figure 5.7 Project Scheduling Process

5.7.1 Basic Principles:

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

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

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

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

Effort validation. Every project has a defined number of people on the software team. As time allocation occurs, you must ensure that no more than the allocated number of people has been scheduled at any given time. For example, consider a project that has three assigned software engineers (e.g., three person-days are available per day of assigned effort4). On a given day, seven concurrent tasks must be accomplished. Each task requires 0.50 person-days of effort. More effort has been allocated than there are people to do the work.

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

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

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

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

5.7.2 Scheduling:

- Scheduling of a software project does not differ greatly from scheduling of any multitask engineering effort. Therefore, generalized project scheduling tools and techniques can be applied with little modification for software projects.
- **Program evaluation and review technique (PERT) and the critical path method (CPM)** are two project scheduling methods that can be applied to software development.
- Both techniques are driven by information already developed in earlier project planning activities: estimates of effort, a decomposition of the product function, the selection of the appropriate process model and task set, and decomposition of the tasks that are selected.

5.7.3.1 Time-Line Charts:

- When creating a software project schedule begin with a set of tasks (the work breakdown structure).
- If automated tools are used, the work breakdown is input as a task network or task outline. Effort, duration, and start date are then input for each task. In addition, tasks may be assigned to specific individuals.
- As a consequence of this input, a time-line chart, also called a Gantt chart, is generated. A time-line chart can be developed for the entire project. Alternatively, separate charts can be developed for each project function or for each individual working on the project.
- Figure 5.8illustrates the format of a time-line chart.

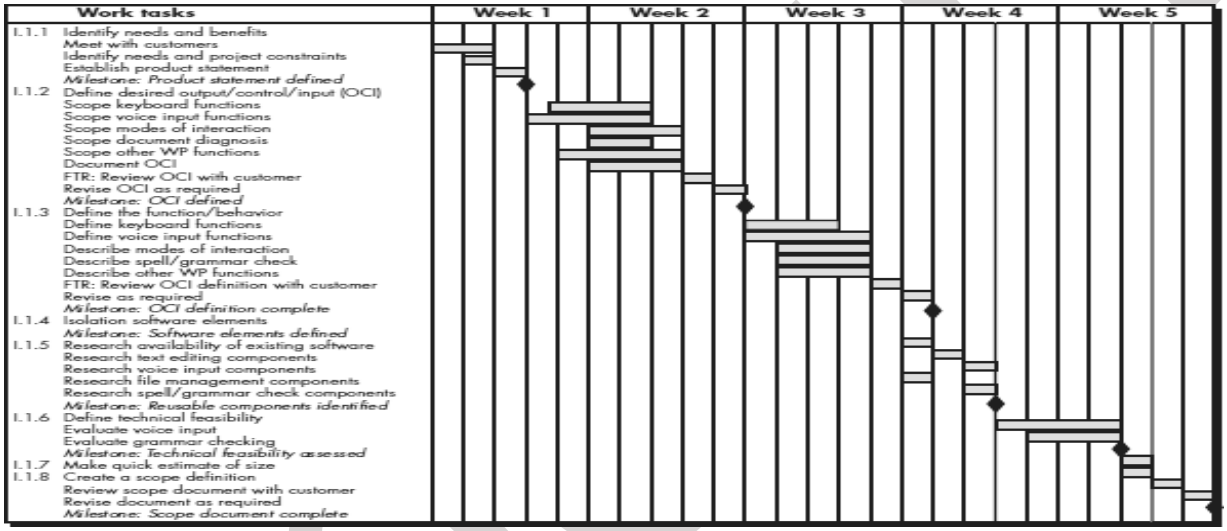


Figure 5.8 Format of Time-Line chart

An example time-line chart

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

Figure 5.9 An example project table

5.7.3.2 Tracking the Schedule:

The project schedule becomes a road map that defines the tasks and milestones to be tracked and controlled as the project proceeds. Tracking can be accomplished in a number of different ways:

- Conducting periodic project status meetings in which each team member reports progress and problems.
- Evaluating the results of all reviews conducted throughout the software engineering process.
- Determining whether formal project milestones (the diamonds shown in Figure) have been accomplished by the scheduled date.
- Comparing the actual start date to the planned start date for each project task listed in the resource table.
- Using earned value analysis to assess progress quantitatively.

- When faced with severe deadline pressure, experienced project managers sometimes use a project scheduling and control technique called **time-boxing.**
- The time-boxing strategy recognizes that the complete product may not be deliverable by the predefined deadline.
- Therefore, an incremental software paradigm is chosen, and a schedule is derived for each incremental delivery.
- The tasks associated with each increment are then time-boxed.
- This means that the schedule for each task is adjusted by working backward from the delivery date for the increment. A "box" is put around each task.
- When a task hits the boundary of its time box (plus or minus 10 percent), work stops and the next task begins.

5.8 EARNED VALUE ANALYSIS

- Earned value analysis (EVA) is technique of performing quantitative analysis of the software project.
- Earned value system provides a common value scale for every task of software project.
- The EVA acts as a measure for software project progress.
- With the help of quantitative analysis made in EVA, we can know how much percentage of the project is completed.

To determine the earned value, the following steps are performed:

1. The budgeted cost of work scheduled (BCWS) is determined for each work task represented in the schedule. During estimation, the work (in person-hours or person-days) of each software engineering task is planned.

   Hence, **BCWSi** is the effort planned for work task i. To determine progress at a given point along the project schedule, the value of BCWS is the sum of the BCWSi values for all work tasks that should have been completed by that point in time on the project schedule.

2. The BCWS values for all work tasks are summed to derive the budget at completion (BAC). Hence,

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

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

Difference between the BCWS and the BCWP:

BCWS - represents the budget of the activities that were planned to be completed

BCWP - represents the budget of the activities that actually were completed.

Given values for BCWS, BAC, and BCWP, important progress indicator scan be computed:

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

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

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

- Provides an indication of the percentage of work that should have been completed by time t.

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

- Provides a quantitative indication of the percent of completeness of the project at a given point in time t.
- It is also possible to compute the actual cost of work performed (ACWP).
- The value for ACWP is the sum of the effort actually expended on work tasks that have been completed by a point in time on the project schedule.
- It is then possible to compute

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

- A CPI value close to 1.0 provides a strong indication that the project is within its defined budget.
- CV is an absolute indication of cost savings (against planned costs) or shortfall at a particular stage of a project.
- Thus EVA ultimately helps the project manager to take the appropriate corrective actions.

5.9 PROJECT PLAN:

- Aplan, drawn up at the start of the project, should be used as the driver for the project
- The initial plan should be the best possible plan given the available information. It must be regularly revised as new information becomes available.

- PP is the application of knowledge, skills, tools, and techniques to project activities to meet project requirements.
- Software engineering is a managed process. The software development takes place within an organization and is subject to a range of schedule, budget, and organizational constraints. **Project planning process:**

Establish the project constraints
Make initial assessments of the project parameters
Define project milestones and deliverables
while project has not been completed or cancelled **loop**
    Draw up project schedule
    Initiate activities according to schedule
    Wait (for a while)
    Review project progress
    Revise estimates of project parameters
    Update the project schedule
    Re-negotiate project constraints and deliverables
    if (problems arise) **then**
        Initiate technical review and possible revision
    end if
end loop

THE PROJECT PLAN:

- The project plan sets out:
– The resources available to the project
– The work breakdown
– A schedule for the work

PROJECT PLAN STRUCTURE:

- Introduction
– Objectives, constraints (e.g., budget, time, etc…)
- Project organization
– People involved and their roles in the team
- Risk analysis
– Possible risks, their likelihood and reduction strategies
- Hardware and software resource requirements
- Work breakdown
– Breaks down the project into activities, identifies milestones, deliverables
- Project schedule
– Activity dependencies, estimated milestone time, people allocation
- Report generation

_ The structure of the project report, when it should be generated must be decided.

5.10 RISK MANAGEMENT

- The risk denotes the uncertainty that may occur in the choices due to past actions and risk is something which causes heavy losses.
- Risk management refers to the process of making decisions based on an evaluation of the factors that threats to the business.

Software Risks:

Two characteristics:

- **uncertainty**—the risk may or may not happen;
- **loss**—if the risk becomes a reality, unwanted consequences or losses will occur

✓ **Project risks**

- It threatens the project plan, that is, if project risks become real, it is likely that the project schedule will slip and that costs will increase.
- Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project.

✓ **Technical risks**

- It threatens the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible.
- Technical risks identify potential design, implementation, interface, verification, and maintenance problems.

✓ **Business risks** threaten the viability of the software to be built and often threaten the project or the product.

Candidates for the top five business risks are

- Building an excellent product or system that no one really wants **(market risk),**
- Building a product that no longer fits into the overall business strategy for the company **(Strategic risk),**
- Building a product that the sales force doesn't understand how tosell**(sales risk),**
- Losing the support of senior management due to a change in focus or a change in people **(management risk),** and
- Losing budgetary or personnel commitment **(budget risks).**

✓ **Known risks** are those that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date, lack of documented requirements or software scope, poor development environment).

✓ **Predictable risks** are extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests)

✓ **Unpredictable risks** can and do occur, but they are extremely difficult to identify in advance.

Reactive strategy:

- Reactive risk management is a risk management strategy in which when project gets into trouble then only corrective action is taken.
- But when such risks cannot be managed and new risks come up one after the other, the software team flies into action in an attempt to correct problems rapidly.

Proactive strategy:

- A proactive strategy begins long before technical work is initiated.
- Potential risks are identified, their probability and impact are assessed, and they are ranked by importance. Then, the software team establishes a plan for managing risk.
- The primary objective is to avoid risk, but because not all risks can be avoided, the team works to develop a contingency plan that will enable it to respond in a controlled and effective manner. Various activities that are carried out for risk management are-
  1. Risk Identification
  2. Risk Projection
  3. Risk Refinement
  4. Risk mitigation, monitoring and management

5.10.1 Risk Identification

- Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.).
- Risks identification can be done by identifying the known and predictable risks.
- There are two distinct types of risks for each of the categories
- **Generic risks** are a potential threat to every software project.
- **Product-specific risks** can be identified only by those with a clear understanding of the technology, the people, and the environment that is specific to the software that is to be built.
- **Method for identifying risks:**
- One method for identifying risks is **to create a risk item checklist**.
- The check list can be used for risk identification and focuses on some subset of known and predictable risks in the following generic subcategories**:**

- Product size—risks associated with the overall size of the software to be built or modified.
- Business impact—risks associated with constraints imposed by management or the marketplace.
- Stakeholder characteristics—risks associated with the sophistication of the stakeholders and the developer's ability to communicate with stakeholders in a timely manner.
- Process definition—risks associated with the degree to which the software process has been defined and is followed by the development organization.
- Development environment—risks associated with the availability and quality of the tools to be used to build the product.
- Technology to be built—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
- Staff size and experience—risks associated with the overall technical and project experience of the software engineers who will do the work.

Risk Components and Drivers:

- A set of "risk components and drivers" are listed along with their probability of occurrence.
- The U.S. Air Force has published a pamphlet that contains excellent guidelines for software risk identification and abatement.
- The Air Force approach requires that the project manager identify the risk drivers that affect software risk components—performance, cost, support, and schedule.

Performance risk—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.

Cost risk—the degree of uncertainty that the project budget will be maintained.

Support risk—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.

Schedule risk—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

Assessing Overall Project Risk:

The following questions have been derived from risk data obtained by surveying experienced software project managers in different parts of the world.

The questions are ordered by their relative importance to the success of a project.

1. Have top software and customer managers formally committed to support the project?
2. Are end users enthusiastically committed to the project and the system/product to be built?
3. Are requirements fully understood by the software engineering team and its customers?
4. Have customers been involved fully in the definition of requirements?
5. Do end users have realistic expectations?
6. Is the project scope stable?
7. Does the software engineering team have the right mix of skills?
8. Are project requirements stable?
9. Does the project team have experience with the technology to be implemented?
10. Is the number of people on the project team adequate to do the job?
11. Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

5.10.2 Risk Projection:

Risk projection, also called **risk estimation**, attempts to rate each risk in two ways—

(1) The likelihood or probability that the risk is real and

(2) The consequences of the problems associated with the risk, should it occur.

Managers and technical staff to perform four risk projection steps:

1. Establish a scale that reflects the perceived likelihood of a risk.
2. Delineate the consequences of the risk.
3. Estimate the impact of the risk on the project and the product.
4. Assess the overall accuracy of the risk projection so that there will be no misunderstandings.

5.10.2.1 Developing A Risk Table:

A risk table provides you with a simple technique for risk projection. A sample risk table is illustrated in Figure 5.10

- The probability of occurrence of each risk is entered in the next column of the table.
- The probability value for each risk can be estimated by team members individually.
- Next, the impact of each risk is assessed.
- The categories for each of the four risk components—performance, support, cost, and schedule—are averaged to determine an overall impact value.
- Once the first four columns of the risk table have been completed, the table is sorted by probability and by impact.

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	

Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

Figure 5.10 A Risk table for Risk Projection

- High-probability, high-impact risks percolate to the top of the table, and low-probability risks drop to the bottom. This accomplishes **first-order risk prioritization.**
- The **cutoff line** (drawn horizontally at some point in the table) implies that only risks that lie above the line will be given further attention. Risks that fall below the line are reevaluated to accomplish **second-order prioritization.**

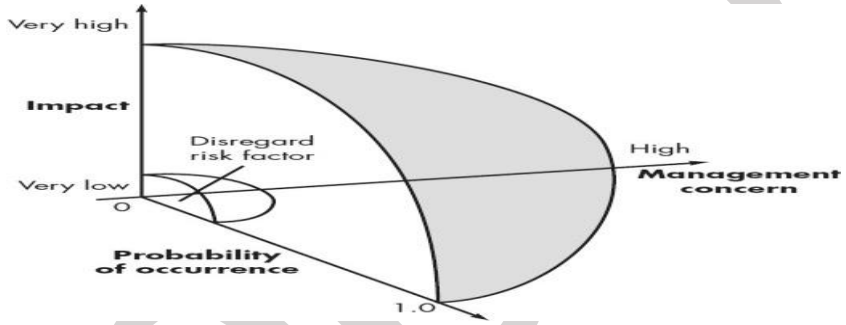


Figure 5.11 Probability Of Occurrence Of Risk Factor

Risk and Management Concern

A risk factor that has a high impact but a very low probability of occurrence should not absorb a significant amount of management time. However, high-impact risks with moderate to high probability and low-impact risks with high probability should be carried forward into the risk analysis.

5.10.2.2 Assessing Risk Impact

While assessing the risks impact three factors are considered

- Nature of risk
- Scope of risk
- Timing at which risk occurs.

The **nature of the risk** indicates the problems that are likely if it occurs. For example, a poorly defined external interface to customer hardware (a technical risk) will preclude early design and testing and will likely lead to system integration problems late in a project.

The **scope of a risk** combines the severity with its overall distribution (how much of the project will be affected or how many stakeholders are harmed?).

The **timing of a risk** considers when and for how long the impact will be felt.

U.S. Air Force suggests the following steps to determine the overall consequences of a risk:

(1) determine the average probability of occurrence value for each risk component;
(2) Determine the impact for each component based on the criteria shown, and
(3) Complete the risk table and analyze the results

The overall risk exposure RE is determined using the following relationship
RE=PXC

where P is the probability of occurrence for a risk, and C is the cost to the project should the risk occur.

5.10.3 Risk Refinement:

- ✓ Risk refinement is a process of specifying the risk. The risk refinement can be represented using CTC format.
- ✓ The CTC stands for condition-transition-consequence.
- ✓ The condition is first stated and then based on this condition sub conditions can be derived.
- ✓ Then determine the effects of these sub conditions in order to refine the risk. This refinement helps in exposing the underlying risks.
- ✓ This approach makes it easier for the project manager to analyze the risk in greater detail.

5.10.4 Risk Mitigation, Monitoring, and Management(RMMM)

An effective strategy must consider three issues:

- Risk avoidance,
- Risk monitoring, and
- Risk management

Risk Mitigation:

Risk mitigation means preventing the risks to occur (risk avoidance). Following are the steps to be taken for mitigating the risks.

- Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, and competitive job market).
- Mitigate those causes that are under your control before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
- Organize project teams so that information about each development activity is widely dispersed.
- Define work product standards and establish mechanisms to be sure that all models and documents are developed in a timely manner.

Risk Monitoring:

- The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely.
- In the case of high staff turnover, the general attitude of team members based on project pressures, the degree to which the team has jelled, interpersonal Relationships among team members, potential problems with compensation and benefits, and the availability of jobs within the company and outside it are all monitored.
- In addition to monitoring these factors, a project manager should monitor the effectiveness of risk mitigation steps.

- This is one mechanism for ensuring continuity, should a critical individual leave the project.
- The project manager should monitor work products carefully to ensure that each can stand on its own and that each imparts information that would be necessary if a newcomer were forced to join the software team somewhere in the middle of the project.

Risk Management:

- Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality.
- Continuing the example, the project is well under way and a number of people announce that they will be leaving.
- If the mitigation strategy has been followed, backup is available, information is documented, and knowledge has been dispersed across the team.
- In addition, you can temporarily refocus resources (and readjust the project schedule) to those functions that are fully staffed, enabling newcomers who must be added to the team to "get up to speed."
- Those individuals who are leaving are asked to stop all work and spend their last weeks in "**knowledge transfer mode."**
- This might include video-based knowledge capture, the development of "commentary documents or Wikis," and/or meeting with other team members who will remain on the project. **THE RMMM PLAN**
- A risk management strategy can be included in the software project plan, or the risk management steps can be organized into a separate risk mitigation, monitoring, and management plan (RMMM).
- The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan.
- Each risk is documented individually using a **risk information sheet (RIS).** In most cases, the RIS is maintained using a database system so that creation and information entry, priority ordering, searches, and other analysis may be accomplished easily.
- The format of the RIS is illustrated in Figure 5.12

Risk information sheet			
Risk ID: P02-4-32	Date: 5/9/09	Prob: 80%	Impact: high
Description: Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.			
Refinement/context: Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards. Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components. Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.			
Mitigation/monitoring: 1. Contact third party to determine conformance with design standards. 2. Press for interface standards completion; consider component structure when deciding on interface protocol. 3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.			
Management/contingency plan/trigger: *RE* computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly. Trigger: Mitigation steps unproductive as of 7/1/09.			
Current status: 5/12/09: Mitigation steps initiated.			
Originator: D. Gagne		Assigned: B. Laster	

Figure 5.12 Risk Information Sheet

5.11 COMPUTER-AIDED SOFTWARE ENGINEERING(CASE) TOOLS:

- Computer-aided software engineering (CASE) tools assist software engineering managers and practitioners in every activity associated with the software process
- They automate project management activities, manage all work products produced throughout the process, and assist engineers in their analysis, design, coding and test work.
- CASE tools can be integrated within a sophisticated environment. CASE provides the software engineer with the ability to automate manual activities and to improve engineering insight.
- The integration framework is a collection of specialized programs that enables individual CASE tools to communicate with one another, to create a project database, and to exhibit the same look and feel to the end-user.

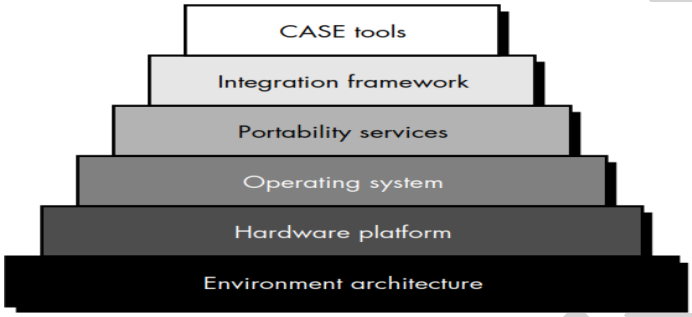


Figure 5.13Case Tools Building Blocks

- Portability services allow CASE tools and their integration framework to migrate across different *hardware platforms and operating systems* without significant adaptive maintenance.

1. The building blocks depicted in Figure 5.13 represent a comprehensive foundation for the integration of CASE tools. However, most CASE tools in use today have not been constructed using all these building blocks.
2. In fact, some CASE tools remain "point solutions." That is, a tool is used to assist in a particular software engineering activity (e.g., analysis modeling) but does not directly communicate with other tools, is not tied into a project database, is not part of an integrated CASE environment (I-CASE).
3. Although this situation is not ideal, a CASE tool can be used quite effectively, even if it is a point solution.

A taxonomy of case tools

- CASE tools can be classified by function, by their role as instruments for managers or technical people, by their use in the various steps of the software engineering process, by the environment architecture (hardware and software) that supports them, or even by their origin or cost
- The taxonomy presented here uses function as a primary criterion.

Project planning tools	Interface design and development tools
Risk analysis tools	Prototyping tools
Project managementtools	Programming tools
Requirements tracing tools	Web development tools
Metrics and management tools	Integration and testing tools
Documentation tools	Static analysis tools
System software tools	Dynamic analysis tools
Quality assurance tools	Test management tools
Database management tools	Client/server testing tools
Software configuration management tools	Reengineering tools