Software Project Management

- The Management Spectrum:-
- Effective software project management focuses on the four P's: people, product, process, and project.

- The people
  - Deals with the cultivation of motivated, highly skilled people
  - Consists of the stakeholders, the team leaders, and the software team.

  The product
  - Product objectives and scope should be established before a project can be planned

- The process
  - The software process provides the framework from which a comprehensive plan for software development can be established

- The project
  - Planning and controlling a software project is done for one primary reason…it is the only known way to manage complexity
  - In a 1998 survey, 26% of software projects failed outright, 46% experienced cost and schedule overruns, 28% of project are successful.

- **PEOPLE**
- Managers argue that people are primary, but their actions sometimes belie their words.

- **The Stakeholders:-**
- The software process is populated by stakeholders who can be categorized into one of five constituencies:

1. *Senior managers* who define the business issues that often have a significant influence on the project.

2. *Project (technical) managers* who must plan, motivate, organize, and control the practitioners who do software work.

3. *Practitioners* who deliver the technical skills that are necessary to engineer a product or application.

4. *Customers* who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.

5. *End users* who interact with the software once it is released for production use.

- **Team Leaders**
- To be effective, the project team must be organized in a way that maximizes each person's skills and abilities. And that's the job of the team leader.

- **Motivation.**
  - The ability to encourage (by "push or pull") technical people to produce to their best ability.

- **Organization.**
  - The ability to mold existing processes (or invent new ones)that will enable the initial concept to be translated into a final product.

- **Ideas or innovation.**
  - The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

- Another view of the characteristics that define an effective project manager emphasizes four key traits:

- **Problem solving** – diagnose, structure a solution, apply lessons learned, remain flexible.

- **Managerial identity** – take charge of the project, have confidence to assume control, have assurance to allow good people to do their jobs.

- **Achievement** – reward initiative, demonstrate that controlled risk taking will not be punished.

- **Influence and team building** – be able to “read” people, understand verbal and nonverbal signals, be able to react to signals, remain under control in high-stress situations.

- **The Software Team**
- Seven project factors to consider when structuring a software development team
  - The difficulty of the problem to be solved
  - The size of the resultant program(s) in source lines of code
  - The time that the team will stay together
  - The degree to which the problem can be modularized
  - The required quality and reliability of the system to be built
  - The rigidity of the delivery date
  - The degree of sociability (communication) required for the project

- Four organizational paradigms for software development teams

  - **Closed paradigm** – traditional hierarchy of authority; works well when producing software similar to past efforts; members are less likely to be innovative

  - **Random paradigm** – depends on individual initiative of team members; works well for projects requiring innovation or technological breakthrough; members may struggle when orderly performance is required

- **Open paradigm** – hybrid of the closed and random paradigm; works well for solving complex problems; requires collaboration, communication, and consensus among members

- **Synchronous paradigm** – organizes team members based on the natural pieces of the problem; members have little communication outside of their subgroups

- To achieve a high-performance team:
  - Team members must have trust in one another.
  - The distribution of skills must be appropriate to the problem.
  - Mavericks may have to be excluded from the team, if team cohesiveness is to be maintained

- Five factors that cause team toxity (i.e., a toxic team environment):-
  - A frenzied work atmosphere
  - High frustration that causes friction among team members
  - A fragmented or poorly coordinated software process
  - An unclear definition of roles on the software team
  - Continuous and repeated exposure to failure

- How to avoid these problems
  - Give the team access to all information required to do the job
  - Do not modify major goals and objectives, once they are defined, unless absolutely necessary
  - Give the team as much responsibility for decision making as possible
  - Let the team recommend its own process model
  - Let the team establish its own mechanisms for accountability (i.e., reviews)
  - Establish team-based techniques for feedback and problem solving

- **Agile Teams:-**

- The small, highly motivated project team, also called an *agile team*.

- To make effective use of the competencies of each team member and to foster effective collaboration through a software project, agile teams are *self-organizing*.

- An agile team might conduct daily team meetings to coordinate and synchronize the work that must be accomplished for that day.

- **Coordination and Communication Issues:-**
- The scale of many development efforts is large, leading to complexity, confusion, and significant difficulties in coordinating team members.

- Interoperability has become a key characteristic of many systems.

- To deal with them effectively, you must establish effective methods for coordinating the people who do the work.

- To accomplish this, mechanisms for formal and informal communication among team members and between multiple teams must be established.

- Formal communication is accomplished through “ writing, structured meetings, and other relatively non-interactive and impersonal communication channels”

- Informal communication is more personal.

- **THE PRODUCT:-**
- Software Scope:-

- Scope is defined by answering the following questions:

- **Context.** How does the software to be built fit into a larger system, product, or business context, and what constraints are imposed as a result of the context?

- **Information objectives.** What customer-visible data objects are produced as output from the software? What data objects are required for input?

- **Function and performance.** What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?

- Software project scope must be unambiguous and understandable at the management and technical levels.

- Problem Decomposition:-

- Problem decomposition
  - Also referred to as partitioning or problem elaboration
  - Sits at the core of software requirements analysis
- Two major areas of problem decomposition
  - The functionality that must be delivered
  - The process that will be used to deliver it

- **THE PROCESS**
- Your team must decide which process model is most appropriate for

(1) the customers who have requested the product and the people who will do the work,

(2) the characteristics of the product itself, and

(3) the project environment in which the software team works.

- When a process model has been selected, the team then defines a preliminary project plan based on the set of process framework activities.

- Once the preliminary plan is established, process decomposition begins.

- Project planning begins with the melding of the product and the process.

- Generic framework activities— **communication, planning, modeling, construction, and deployment**

- Process Decomposition:-
- Once the process model has been chosen, the process framework is adapted to it.

- Small , relatively simple project might require the following work tasks for the communication activity:

1. Develop list of clarification issues.
2. Meet with stakeholders to address clarification issues.
3. Jointly develop a statement of scope.
4. Review the statement of scope with all concerned.
5. Modify the statement of scope as required

- For complex project, which has a broader scope and more significant business impact.
- Such a project might require the following work tasks for the **communication:**

1. Review the customer request.

2. Plan and schedule a formal, facilitated meeting with all stakeholders.

3. Conduct research to specify the proposed solution and existing approaches.

4. Prepare a “working document” and an agenda for the formal meeting.

5. Conduct the meeting

6. Jointly develop mini-specs that reflect data, functional, and behavioral features of the software. Alternatively, develop use cases that describe the software from the user's point of view.

7. Review each mini-spec or use case for correctness, consistency, and lack of ambiguity.

8. Assemble the mini-specs into a scoping document.

9. Review the scoping document or collection of use cases with all concerned.

10. Modify the scoping document or use cases as required.

- **THE PROJECT**
- In order to manage a successful software project, you have to understand what can go wrong so that problems can be avoided.

- 10 signs that indicate that an information systems project is in risk:

1. Software people don't understand their customer's needs.
2. The product scope is poorly defined.
3. Changes are managed poorly.
4. The chosen technology changes.

5. Business needs change [or are ill defined].

6. Deadlines are unrealistic.

7. Users are resistant.

8. Sponsorship is lost [or was never properly obtained].

9. The project team lacks people with appropriate skills.

10. Managers [and practitioners] avoid best practices and lessons learned.

- Five -part commonsense approach to software projects:

- Start on the right foot
  - Understand the problem; set realistic objectives and expectations; form a good team

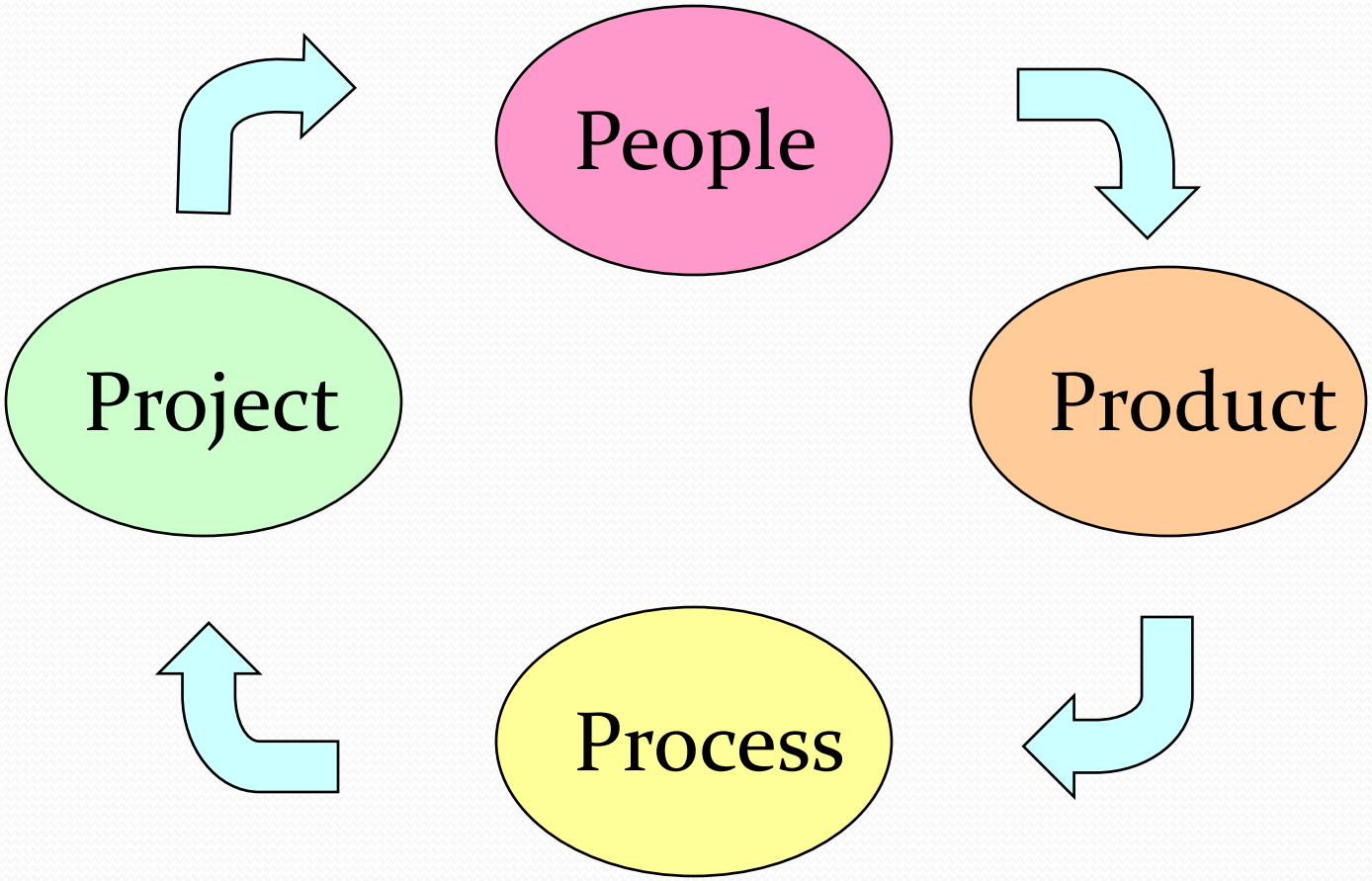
- Maintain momentum
  - Provide incentives to reduce turnover of people; emphasize quality in every task; have senior management stay out of the team's way.

- Track progress
  - Track the completion of work products; collect software process and project measures; assess progress against expected averages

- Make smart decisions
  - Keep it simple; use existing software before writing new code; follow standard approaches; identify and avoid risks; always allocate more time than you think you need to do complex or risky tasks

- Conduct a post mortem analysis
  - Track lessons learned for each project; compare planned and actual schedules; collect and analyze software project metrics; get feedback from teams members and customers; record findings in written form

Summary



- **THE W5HH PRINCIPLE:-**
- An approach that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches, and required resources. He calls it the *W5HH Principle*.

- A series of questions that lead to a definition of key project characteristics and the resultant project plan:-

- *Why is the system being developed?*
- All stakeholders should assess the validity of business reasons for the software work. Does the business purpose justify the expenditure of people, time, and money?

- *What will be done?*
- The task set required for the project is defined.

- *When will it be done?*
- The team establishes a project schedule by identifying when project tasks are to be conducted and when milestones are to be reached.

- *Who is responsible for a function?*
- The role and responsibility of each member of the software team is defined.

- *Where are they located organizationally?*
- Not all roles and responsibilities reside within software practitioners. The customer, users, and other stakeholders also have responsibilities.

- *How will the job be done technically and managerially?*
- Once product scope is established, a management and technical strategy for the project must be defined.

- *How much of each resource is needed?*
- The answer to this question is derived by developing estimates based on answers to earlier questions.

- **Critical Practices:-**
- Critical practices include:

- Metric -based project management

- Empirical cost and schedule estimation

- Earned value tracking defect tracking against quality targets

- People aware management

Estimation for Software Projects

- Software project management begins with a set of activities that are collectively called *project planning*.

- Before the project can begin, the software team should estimate the work to be done, the resources that will be required, and the time that will elapse from start to finish.

- **Observations On Estimation**
- Planning requires you to make an initial commitment, even though it's likely that this "commitment" will be proven wrong.

- Estimation carries natural risk and this risk leads to uncertainty:-
- *Project complexity.*
- *Project size*
- The *degree of structural uncertainty.*

- **The Project Planning Process**

1) Establish project scope
2) Determine feasibility
3) Analyze risks
4) Define required resources
    a) Determine human resources required
    b) Define reusable software resources
    c) Identify environmental resources
5) Estimate cost and effort
    a) Decompose the problem
    b) Develop two or more estimates using different approaches
    c) Reconcile the estimates
6) Develop a project schedule
    a) Establish a meaningful task set
    b) Define a task network
    c) Use scheduling tools to develop a timeline chart
    d) Define schedule tracking mechanisms

- **Software Scope And Feasibility**

- Software scope describes
  - The functions and features that are to be delivered to end users
  - The data that are input to and output from the system
  - The "content" that is presented to users as a consequence of using the software
  - The performance, constraints, interfaces, and reliability that bound the system

- Scope can be define using two techniques
  - A narrative description of software scope is developed after communication with all stakeholders
  - A set of use cases is developed by end users

- After the scope has been identified, two questions are asked
  - Can we build software to meet this scope?
  - Is the project feasible?

- Software engineers too often rush (or are pushed) past these questions

- **Feasibility**
- After the scope is resolved, feasibility is addressed

- Software feasibility has four dimensions
  - **Technology** – Is the project technically feasible? Is it within the state of the art? Can defects be reduced to a level matching the application's needs?

  - **Finance** – Is it financially feasible? Can development be completed at a cost that the software organization, its client, or the market can afford?

  - **Time** – Will the project's time-to-market beat the competition?

  - **Resources** – Does the software organization have the resources needed to succeed in doing the project?

- **Resource Estimation**
- The second planning task is estimation of the resources required to accomplish the software development effort.

- Three major categories of software engineering resources
  - People
  - Development environment
  - Reusable software components
    - Often neglected during planning but become a paramount concern during the construction phase of the software process

- Each resource is specified with

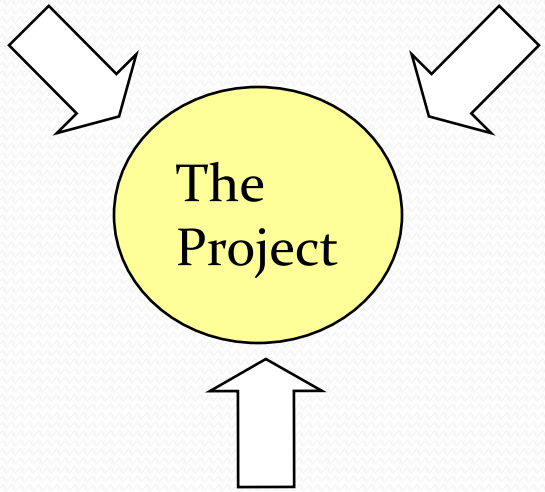
  - A description of the resource
  - A statement of availability
  - The time when the resource will be required
  - The duration of time that the resource will be applied

Time window

Categories of Resources

People
- Number required
- Skills required
- Geographical location

Development Environment
- Software tools
- Computer hardware
- Network resources



Reusable Software Components
- Off-the-shelf components
- Full-experience components
- Partial-experience components
- New components

- **Human Resources**:-
- Planners need to select the number and the kind of people skills needed to complete the project
- They need to specify the organizational position and job specialty for each person
- Small projects of a few person-months may only need one individual
- Large projects spanning many person-months or years require the location of the person to be specified also
- The number of people required can be determined only after an estimate of the development effort

- **Development Environment Resources:-**
- A software engineering environment (SEE) incorporates hardware, software, and network resources that provide platforms and tools to develop and test software work products.

- Most software organizations have many projects that require access to the SEE provided by the organization.

- Planners must identify the time window required for hardware and software and verify that these resources will be available

- **Reusable Software Resources:-**
- Component-based software engineering (CBSE) emphasizes reusability—that is, the creation and reuse of software building blocks.

- Such building blocks, often called *components*.

- Four software resource categories that should be considered as planning proceeds:

- Off-the-shelf components
  - Components are from a third party or were developed for a previous project.
  - Ready to use; fully validated and documented; virtually no risk

- Full-experience components
  - Components are similar to the software that needs to be built
  - Software team has full experience in the application area of these components
  - Modification of components will incur relatively low risk

- Partial-experience components
  - Components are related somehow to the software that needs to be built but will require substantial modification
  - Software team has only limited experience in the application area of these components
  - Modifications that are required have a fair degree of risk

- New components
  - Components must be built from scratch by the software team specifically for the needs of the current project
  - Software team has no practical experience in the application area
  - Software development of components has a high degree of risk

- **Software Project Estimation**
- Software cost and effort estimation will never be an exact science
- Options for achieving reliable cost and effort estimates

1) Delay estimation until late in the project (we should be able to achieve 100% accurate estimates after the project is complete)

2) Base estimates on similar projects that have already been completed

3) Use relatively simple decomposition techniques to generate project cost and effort estimates

4) Use one or more empirical estimation models for software cost and effort estimation

- Unfortunately,
- Option #1 is not practical, but results in good numbers
- Option #2 can work reasonably well, but it also relies on other project influences being roughly equivalent
- Options #3 and #4 can be done in tandem to cross check each other

- Project Estimation Approaches:-

- Decomposition techniques
  - These take a "divide and conquer" approach
  - Cost and effort estimation are performed in a stepwise fashion by breaking down a project into major functions and related software engineering activities

- Empirical estimation models
  - Offer a potentially valuable estimation approach if the historical data used to seed the estimate is good

- **Decomposition Techniques**

- Before an estimate can be made and decomposition techniques applied, the planner must
  - Understand the scope of the software to be built
  - Generate an estimate of the software's size.

- Then one of two approaches are used
  - Problem-based estimation
    - Based on either source lines of code or function point estimates
  - Process-based estimation
    - Based on the effort required to accomplish each task

- **Software Sizing:-**

- The accuracy of a software project estimate is predicated on
  - The degree to which the planner has properly estimated the size of the product to be built
  - The ability to translate the size estimate into human effort, calendar time, and money
  - The degree to which the project plan reflects the abilities of the software team
  - The stability of both the product requirements and the environment that supports the software engineering effort

- *Software sizing is a problem.*

- Project estimate is only as good as the estimate of the size of the work to be accomplished, sizing represents your first major challenge as a planner.

- In the context of project planning, size refers to a quantifiable outcome of the software project.

- 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).

- Four different approaches to the sizing problem:

- *"Fuzzy logic" sizing:-* 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:-*
  - Estimate the number of occurrences of each standard component
  - Use historical project data to determine the delivered LOC size per standard component.

- *Change sizing:-*

  - Used when changes are being made to existing software
  - Estimate the number and type of modifications that must be accomplished
  - Types of modifications include reuse, adding code, changing code, and deleting code
  - An effort ratio is then used to estimate each type of change and the size of the change

- **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

(2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

- Baseline productivity metrics (e.g., LOC/pm or FP/pm) are then applied to the appropriate estimation variable, and cost or effort for the function is derived.

- In general, the LOC/pm and FP/pm metrics should be computed by project domain
  - Important factors are team size, application area, and complexity .
- LOC and FP estimation differ in the level of detail required for decomposition with each value
  - For LOC, decomposition of functions is essential and should go into considerable detail (the more detail, the more accurate the estimate)
  - For FP, decomposition occurs for the five information domain characteristics and the 14 adjustment factors
    - External inputs, external outputs, external inquiries, internal logical files, external interface files

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**

- 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.*

- $S = (S_{opt} + 4S_m + S_{pess})/6$

- Historical LOC or FP data is then compared to S in order to cross-check it.

Line No.	
1.	// Header file is initialized.
2.	#include<stdio.h>
3.	#include<conio.h>
4.	// declare main function
5.	void main()
6.	{
7.	int i,n=5,fact=1;
8.	// variable declaration and initialization
9.	for(i=n;i≥1;i--)
10.	{
11.	fact = fact *n;
12.	}
13.	// Display output.
14.	printf("Factorial: %d", fact);
15.	getch();
16.	}

: According to the definition of the LOC, above Program (shown in Fig. 1) has a 12 LOC.

S. No.	Functional Units	Weight factors		
		simple	Average	complex
1.	Input or set of inputs to the system	3	4	6
2.	Outputs from the system	4	5	7
3.	Enquiries	3	4	6
4.	Internal logical information	7	10	15
5.	External interface required	5	7	10

- **Process-Based Estimation:-**

1) Identify the set of functions that the software needs to perform as obtained from the project scope

2) Identify the series of framework activities that need to be performed for each function

3) Estimate the effort (in person months) that will be required to accomplish each software process activity for each function

4) Apply average labor rates (i.e., cost/unit effort) to the effort estimated for each process activity

4) Compute the total cost and effort for each function and each framework activity (See table)

5) Compare the resulting values to those obtained by way of the LOC and FP estimates

- If both sets of estimates agree, then your numbers are highly reliable
- Otherwise, conduct further investigation and analysis concerning the function and activity breakdown

Process-based estimation table

Activity →	CC	Planning	Risk analysis	Engineering		Construction release		CE	Totals
Task →				Analysis	Design	Code	Test		
Function ↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

CC = customer communication CE = customer evaluation

- **Reconciling Estimates**
- The results gathered from the various estimation techniques must be reconciled to produce a single estimate of effort, project duration, and cost

- If widely different estimates occur, investigate the following causes
  - The scope of the project is not adequately understood or has been misinterpreted by the planner
  - Productivity data used for problem-based estimation techniques is inappropriate for the application, obsolete (i.e., outdated for the current organization), or has been misapplied

- The planner must determine the cause of deviation and then reconcile the estimates

- **Empirical Estimation Models:-**
- Estimation models for computer software use empirically derived formulas to predict effort as a function of LOC or FP

- Resultant values computed for LOC or FP are entered into an estimation model

- The empirical data for these models are derived from a limited sample of projects
  - Consequently, the models should be calibrated to reflect local software development conditions.

- The COCOMO II Model:-
- COCOMO, for *COnstructive COst MOdel*.

- COCOMO model became one of the most widely used and discussed software cost estimation models in the industry.

- 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.

- *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.

- Three different sizing options are available as part of the model hierarchy: object points, function points, and lines of source code.

- Like function 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,
- (3) components likely to be required to build the application.

- Once complexity is determined, the number of screens, reports, and components are weighted according to the table.

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

- The object point count is then determined

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

- To derive an estimate of effort based on the computed NOP value, a "productivity rate" must be derived.

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

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

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

- The Software Equation:-

- The software equation is a dynamic multivariable model that assumes a specific distribution of effort over the life of a software development project.

- We derive an estimation model of the form

$$E = \frac{\text{LOC} \times B^{0.333}}{P^3} \times \frac{1}{t^4} \qquad (26.4)$$

where

E = effort in person-months or person-years
t = project duration in months or years
B = "special skills factor"[13]
P = "productivity parameter" that reflects: overall process maturity and management practices, the extent to which good software engineering practices are used, the level of programming languages used, the state of the software environment, the skills and experience of the software team, and the complexity of the application

- Typical values might be
- P 2000 for development of real-time embedded software,
- P 10,000 for telecommunication and systems software,
- P 28,000 for business systems applications.

- **Estimation For Object-oriented Projects:-**

1. Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications.

2. Using the requirements model , develop use cases and determine a count. Recognize that the number of use cases may change as the project progresses.

3. From the requirements model, determine the number of key classes.

4. Categorize the type of interface for the application and develop a multiplier for support classes:

Interface Type	**Multiplier**
No GUI	2.0
Text-based user interface	2.25
GUI	2.5
Complex GUI	3.0

5. Multiply the total number of classes (key + support) by the average number of work units per class. Lorenz and Kidd suggest 15 to 20 person-days per class.

6. Cross-check the class-based estimate by multiplying the average number of work units per use case.

- **Specialized Estimation Techniques:-**
- Two specialized estimation techniques:-
- Estimation for Agile Development:-
- Estimation for agile projects uses a decomposition approach that encompasses the following steps:

1. Each user scenario (the equivalent of a mini use case created at the very start of a project by end users or other stakeholders) is considered separately for estimation purposes.

2. The scenario is decomposed into the set of software engineering tasks that will be required to develop it.

3a. The effort required for each task is estimated separately. Note: Estimation can be based on historical data, an empirical model, or “experience.”

3b. Alternatively, the “volume” of the scenario can be estimated in LOC, FP, or some other volume-oriented measure (e.g., use-case count).

4a. Estimates for each task are summed to create an estimate for the scenario.

4b. Alternatively, the volume estimate for the scenario is translated into effort using historical data.

5. The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment.

- This estimation approach serves two purposes:

- 1) to be certain that the number of scenarios to be included in the increment conforms to the available resources,

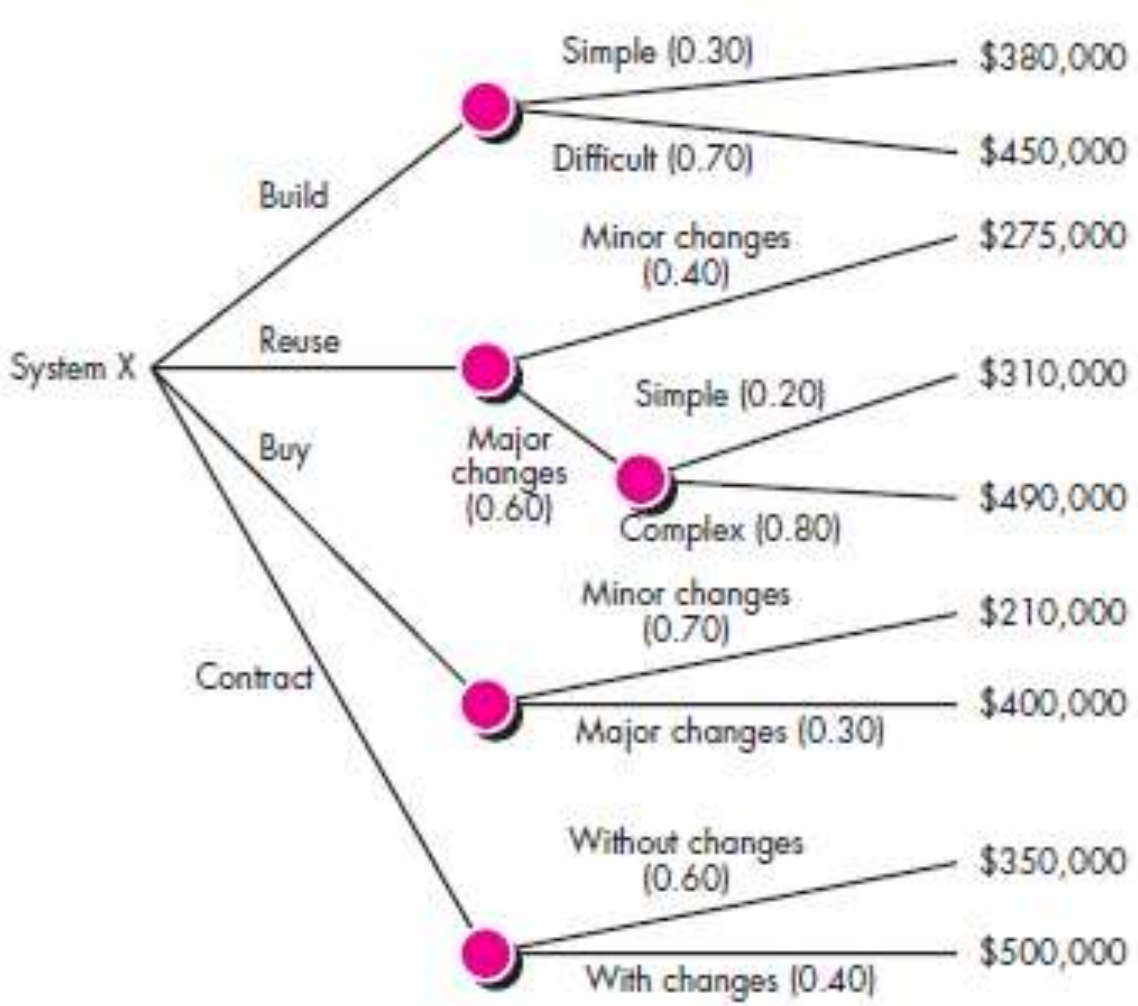
- (2) to establish a basis for allocating effort as the increment is developed.

- Estimation for WebApp Projects:-
- Function points are a reasonable indicator of volume for a WebApp.

- **The Make/Buy Decision:-**
- It is often more cost effective to acquire rather than develop computer software.
- 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 and then modified and integrated to meet specific needs,
- (3) software may be custom built by an outside contractor to meet the purchaser’s specifications.

- The make/buy decision is made based on the following conditions:
- (1) Will the delivery date of the software product be sooner than that for internally developed software?
- (2) Will the cost of acquisition plus the cost of customization be less than the cost of developing the software internally?
- (3) Will the cost of outside support (e.g., a maintenance contract) be less than the cost of internal support?

- Creating a Decision Tree:-
- The software engineering organization can
- (1) build system X from scratch,
- (2) reuse existing partial-experience components to construct the system,
- (3) buy an available software product and modify it to meet local needs,
- (4) contract the software development to an outside vendor.


System X
Build
Reuse
Buy
Contract
Simple (0.30)
Difficult (0.70)
$380,000
$450,000
Minor changes (0.40)
$275,000
Major changes (0.60)
Simple (0.20)
Complex (0.80)
$310,000
$490,000
Minor changes (0.70)
$210,000
Major changes (0.30)
$400,000
Without changes (0.60)
$350,000
With changes (0.40)
$500,000

PROJECT SCHEDULING

- The objective of software project scheduling is to create a set of engineering tasks that will enable to complete the job in time.

- **Eight Reasons for Late Software Delivery:-**

1. An unrealistic deadline established by someone outside the software team and forced on managers and practitioners.

2. Changing customer requirements that are not reflected in schedule changes.

3. An honest underestimate of the amount of effort and/or the number of resources that will be required to do the job.

4. Predictable and/or unpredictable risks that were not considered when the project commenced.

5. Technical difficulties that could not have been foreseen in advance.

6. Human difficulties that could not have been foreseen in advance.

7. Miscommunication among project staff that results in delays.

8. A failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem.

- **Handling Unrealistic Deadlines:-**
- Perform a detailed estimate using historical data from past projects. Determine the estimated effort and duration for the project.

- Using an incremental process model, develop a software engineering strategy that will deliver critical functionality by the imposed deadline, but delay other functionality until later. Document the plan.

- Meet with the customer and explain why the imposed deadline is unrealistic
  - Be certain to note that all estimates are based on performance on past projects
  - Also be certain to indicate the percent improvement that would be required to achieve the deadline as it currently exists

- **Project Scheduling:-**
- The reality of a technical project is that hundreds of small tasks must occur to accomplish a larger goal.
  - Some of these tasks lie outside the mainstream and may be completed without worry of impacting on the project completion date
  - Other tasks lie on the critical path; if these tasks fall behind schedule, the completion date of the entire project is put into the risk.

- Project manager's objectives
  - Define all project tasks
  - Build an activity network that depicts their interdependencies
  - Identify the tasks that are critical within the activity network
  - Build a timeline depicting the planned and actual progress of each task
  - Track task progress to ensure that delay is recognized "one day at a time“
  - To do this, the schedule should allow progress to be monitored and the project to be controlled.

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

- During early stages of project planning, a macroscopic schedule is developed identifying all major process framework activities and the product functions to which they apply

- Later, each task is refined into a detailed schedule where specific software tasks are identified and scheduled

- Scheduling for projects can be viewed from two different perspectives
  - In the first view, an end-date for release of a computer-based system has already been established and fixed
    - The software organization is constrained to distribute effort within the prescribed time frame.
  - In the second view, assume that rough chronological bounds have been discussed but that the end-date is set by the software engineering organization
    - Effort is distributed to make best use of resources and an end-date is defined after careful analysis of the software
  - The first view is encountered far more often that the second

- Basic Principles for Project Scheduling:-
- Compartmentalization
  - The project must be compartmentalized into a number of manageable activities, actions, and tasks; both the product and the process are decomposed

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

- Time allocation
  - Each task to be scheduled must be allocated some number of work units
  - In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies
  - Start and stop dates are also established based on 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 team
  - As time allocation occurs, the project manager must ensure that no more than the allocated number of people have been scheduled at any given time

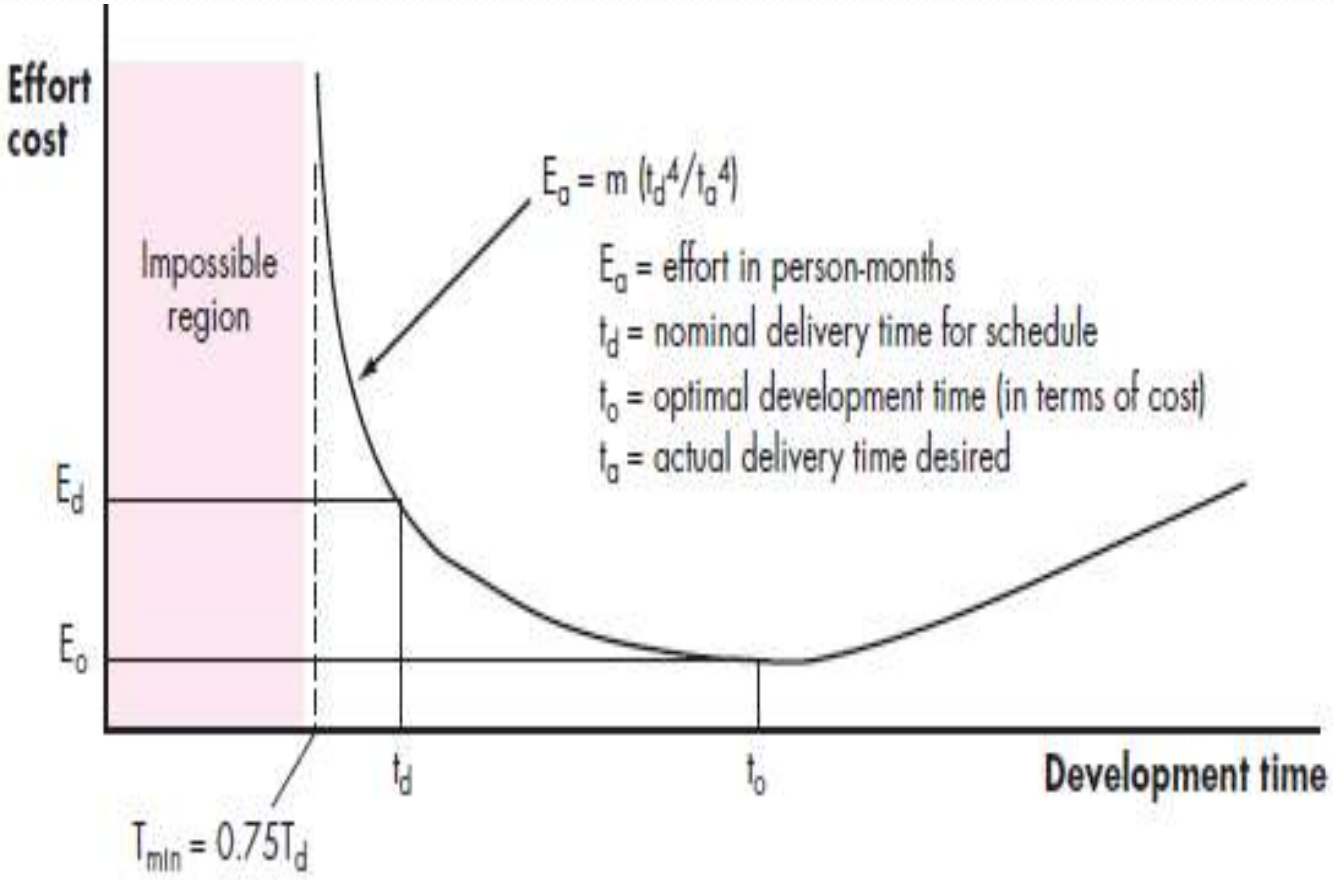
- Defined responsibilities
  - Every task that is scheduled should be assigned to a specific team member
- Defined outcomes
  - Every task that is scheduled should have a defined outcome for software projects such as a work product or part of a work product
  - Work products are often combined in deliverables
- 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

- **The Relationship Between People and Effort:-**
- There is a common myth that is still believed by many managers who are responsible for software development projects:

  "If we fall behind schedule, we can always add more programmers and catch up later in the project."

- The Putnam-Norden-Rayleigh (PNR) Curve provides an indication of the relationship between effort applied and delivery time for a software project.

The relation-ship between effort and delivery time



- Also, delaying project delivery can reduce costs significantly as shown in the equation

$$E = L^3/(P^3t^4)$$

  - E = development effort in person-months
  - L = source lines of code delivered
  - P = productivity parameter (ranging from 2000 to 12000)
  - t = project duration in calendar months

- **Effort Distribution:-**
- A recommended distribution of effort across the software process is 40% (analysis and design), 20% (coding), and 40% (testing)

- Work expended on project planning rarely accounts for more than 2 - 3% of the total effort.

- Requirements analysis may comprise 10 - 25%
  - Effort spent on prototyping and project complexity may increase this

- Software design normally needs 20 – 25%
- Coding should need only 15 - 20% based on the effort applied to software design
- Testing and subsequent debugging can account for 30 - 40%
  - Safety or security-related software requires more time for testing

- **Defining A Task Set For The Software Project:-**
- A task set is the work breakdown structure for the project

- No single task set is appropriate for all projects and process models
  - It varies depending on the project type and the degree of rigor (based on influential factors) with which the team plans to work

- The task set should provide enough discipline to achieve high software quality
  - But it must not burden the project team with unnecessary work

- Types of Software Projects:-
- Concept development projects
  - Explore some new business concept or application of some new technology
- New application development
  - Undertaken as a consequence of a specific customer request
- Application enhancement
  - Occur when existing software undergoes major modifications to function, performance, or interfaces that are observable by the end user

- Application maintenance
  - Correct, adapt, or extend existing software in ways that may not be immediately obvious to the end user.

- Reengineering projects
  - Undertaken with the intent of rebuilding an existing (legacy) system in whole or in part

- Factors that Influence a Project's Schedule:-

- Size of the project
- Number of potential users
- Mission criticality
- Application longevity
- Stability of requirements
- Ease of customer/developer communication
- Maturity of applicable technology
- Performance constraints
- Embedded and non-embedded characteristics
- Project staff
- Reengineering factors

- A Task Set Example:-
- Concept development projects are approached by applying the following actions:
- **Concept scoping**
- **Preliminary concept planning**
- **Technology risk assessment**
- **Proof of concept**
- **Concept implementation**
- **Customer reaction**

- **Defining A Task Network:-**
- Individual tasks and subtasks have interdependencies based on their sequence.

- Also called an activity network

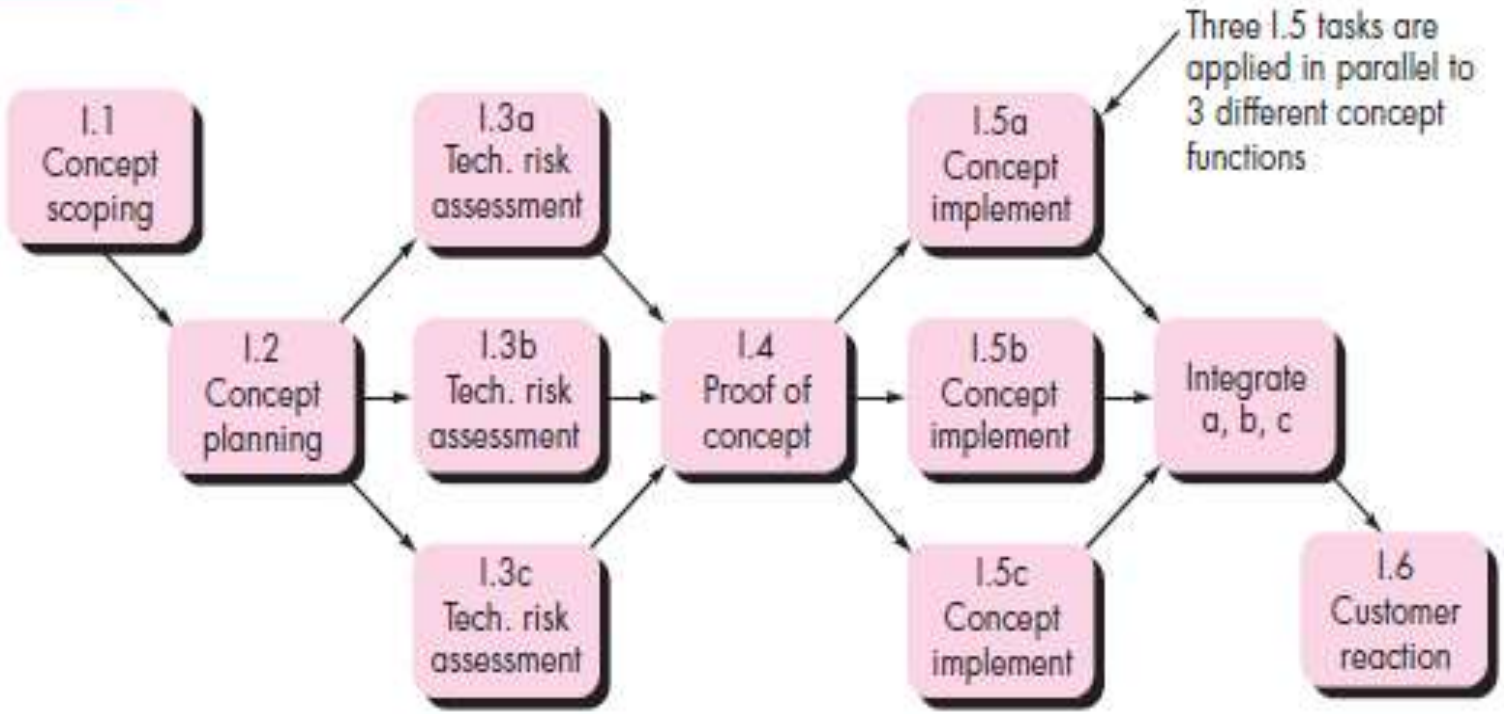
- It is a graphic representation of the task flow for a project

- It represent task length, sequence, concurrency, and dependency

- Points out inter-task dependencies to help the manager ensure continuous progress toward project completion

- The critical path
  - A single path leading from start to finish in a task network
  - It contains the sequence of tasks that must be completed on schedule if the project as a whole is to be completed on schedule
  - It also determines the minimum duration of the project

FIGURE 27.2 A task network for concept development



- **Scheduling:-**
- *Program evaluation and review technique (PERT) and the critical path method (CPM)* are two project scheduling methods that can be applied to software development.

- Time-Line Charts:-
- Effort, duration, and start date are then input for each task.

- 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.
- All project tasks (for concept scoping) are listed in the left-hand column.
- The horizontal bars indicate the duration of each task.
- When multiple bars occur at the same time on the calendar, task concurrency is implied.
- The diamonds indicate milestones.

- Once the information necessary for the generation of a time-line chart has been input, the majority of software project scheduling tools produce *project tables—a tabular* listing of all project tasks, their planned and actual start and end dates, and a variety of related information.

FIGURE 27.3 An example time-line chart

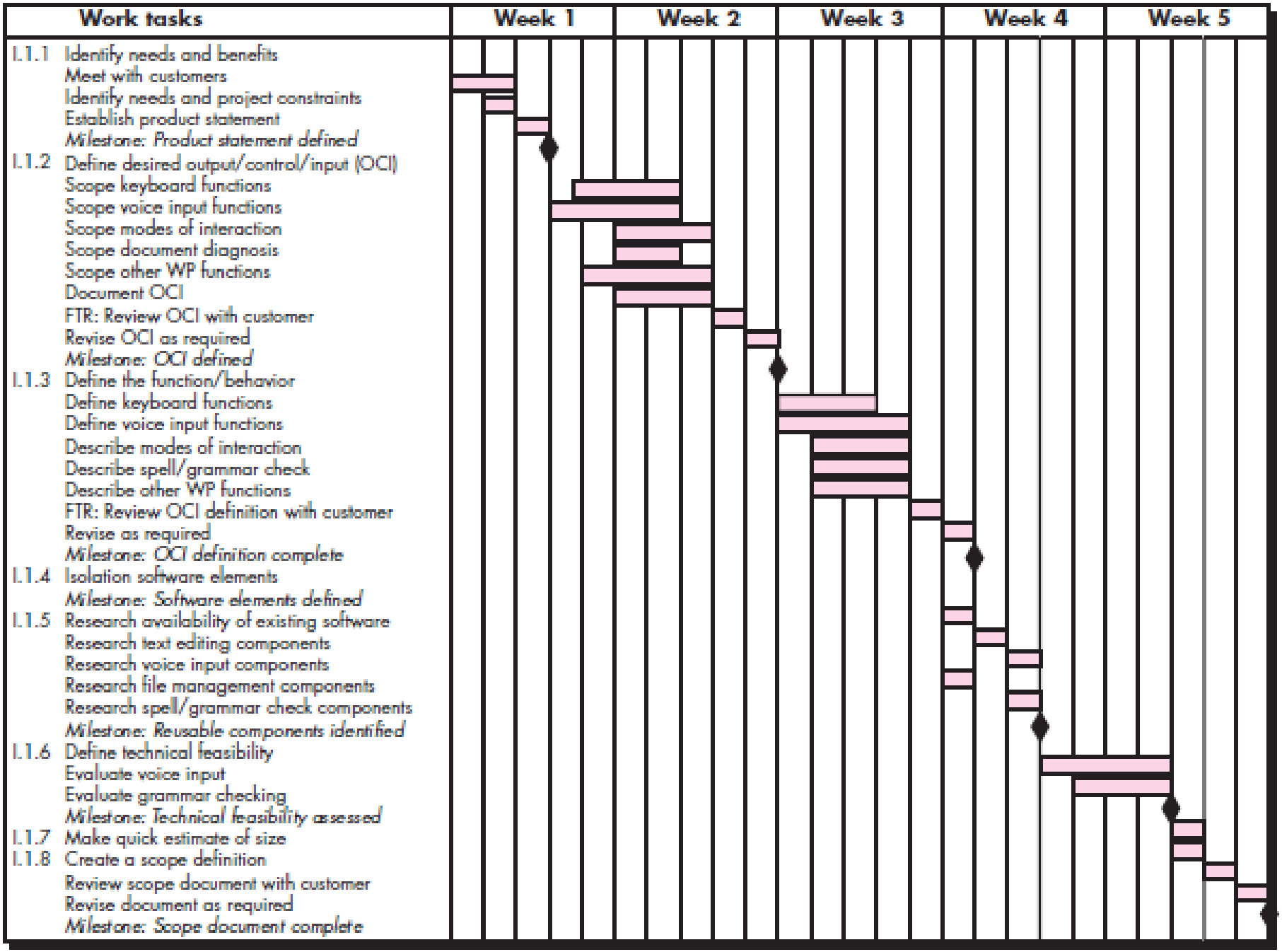


FIGURE 27.4 An example project table

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 pd	
Identify needs and project constraints	wk1, d2	wk1, d2	wk1, d2	wk1, d2	JPP	1 pd	
Establish product statement	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 pd	
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 pd	
Scope voice input functions	wk1, d3	wk1, d3	wk2, d2		JPP	2 pd	
Scope modes of interaction	wk2, d1		wk2, d3		MLL	1 pd	
Scope document diagnostics	wk2, d1		wk2, d2		BLS	1.5 pd	
Scope other WP functions	wk1, d4	wk1, d4	wk2, d3		JPP	2 pd	
Document OCI	wk2, d1		wk2, d3		MLL	3 pd	
FTR: Review OCI with customer	wk2, d3		wk2, d3		all	3 pd	
Revise OCI as required	wk2, d4		wk2, d4		all	3 pd	
Milestone: OCI defined	wk2, d5		wk2, d5				
I.1.3 Define the function/behavior							

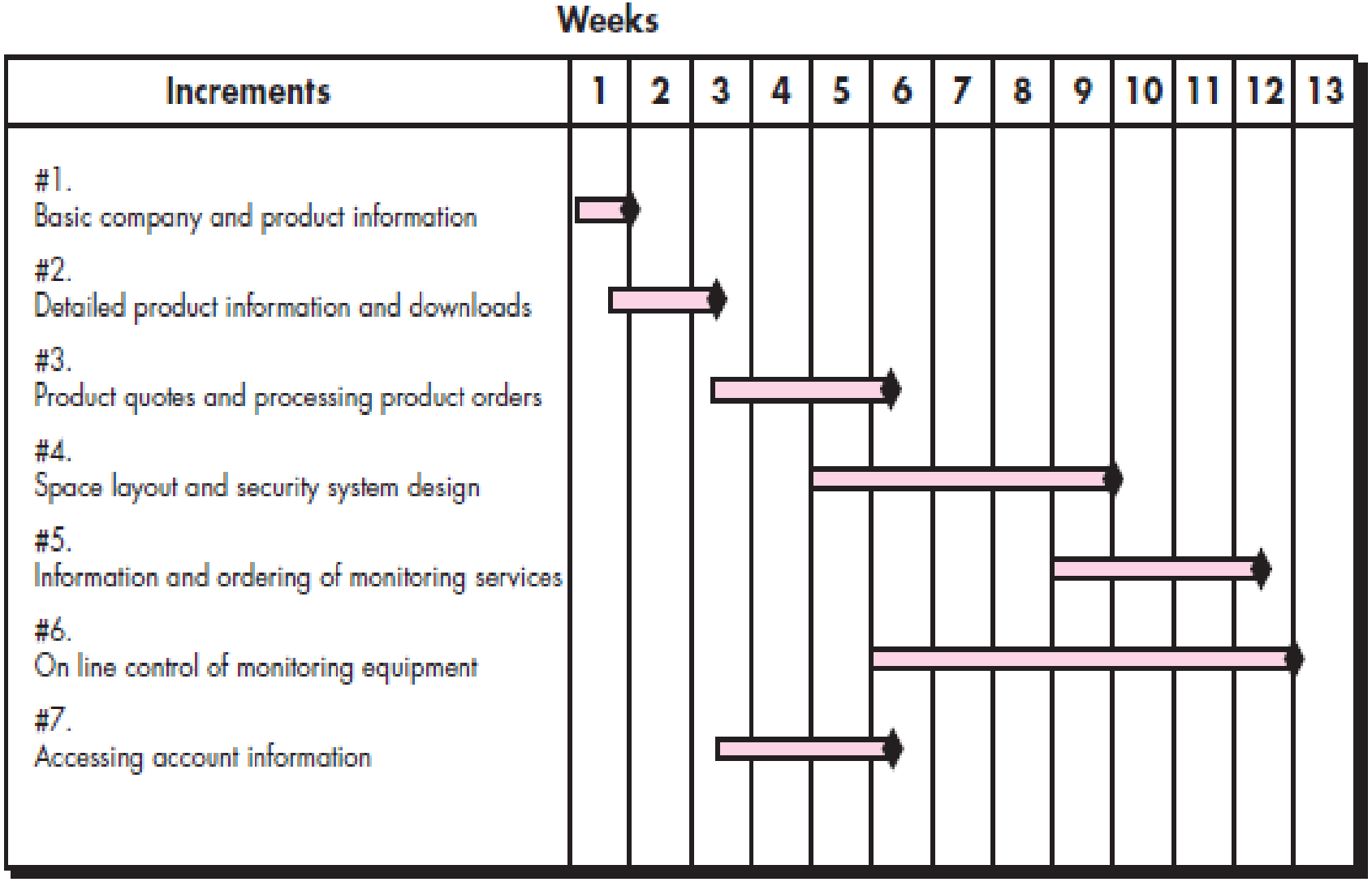
- Tracking the Schedule:-
- Tracking can be accomplished in a number of different ways:
- Conducting periodic project status meetings in which each team member reports progress and problems
- Evaluating the results of all reviews conducted throughout the software engineering process
- Determining whether formal project milestones 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
- Meeting informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon
- Using earned value analysis to assess progress quantitatively

- Scheduling for WebApp Projects:-
- Seven increments can be identified for the Web-based component of the project:

- Increment 1: Basic company and product information
- Increment 2: Detailed product information and downloads
- Increment 3: Product quotes and processing product orders
- Increment 4: Space layout and security system design
- Increment 5: Information and ordering of monitoring services
- Increment 6: Online control of monitoring equipment
- Increment 7: Accessing account information

FIGURE 27.5 Time line for macroscopic project schedule



- **Earned Value Analysis:-**
- Earned value analysis is a measure of progress by assessing the percent of completeness for a project
- It gives accurate and reliable readings of performance very early into a project
- It provides a common value scale (i.e., time) for every project task, regardless of the type of work being performed
- The total hours to do the whole project are estimated, and every task is given an earned value based on its estimated percentage of the total

- 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, BCWS*i is* the effort planned for work task *i*.

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

- *Hence,*
  - BAC =sum(BCWS*k) for all tasks k*

3) The value for *budgeted cost of work performed (BCWP) is computed*

Schedule performance index (SPI), SPI = BCWP/BCWS

Schedule variance (SV), SV = BCWP – BCWS

- SPI is an indication of the efficiency with which the project is utilizing scheduled resources

- Percent scheduled for completion (PSFC)

  PSFC = BCWS/BAC

Percent complete (PC)

PC = BCWP/BAC

Actual cost of work performed (ASWP)

ACWP = sum of BCWP as of time t

A cost performance index (CPI)

CPI = BCWP/ACWP

The cost variance

CV = BCWP – ACWP