Lecture 12

- **Software Project Management**

“What is happening in the project?”

Topics

- Software Project Management
- People in a Project
- Product, Process and Project Metrics
- Project Planning and Estimation
- Project Scheduling

1. Software Project Management

Project

- Definition: A group of tasks performed in a definable time period in order to meet a specific set of objectives
- Project Features:
  - likely to be unique (one-time program)
  - have specific start and end time (life cycle)
  - have work scope that can be categorised into definable tasks
  - has a budget, require use of resources

A Simple Project

“Going to the movies with friends”

Management

- The planning, organizing, staffing, directing and controlling of a company's resources to meet the company's objectives

Definition of Project Management

- The planning, organizing, directing, and controlling of resources for a <u>specific time period</u> to meet a specific set of <u>one-time objectives</u>

Primary Objectives of Project Management

- To meet specified performance
- ... within cost
- ... and on schedule

Project Management Activities

- Establish project objectives
- Defining work requirement
- Determining work timing
- Establishing resource availability and requirements
- Establishing a cost baseline
- Evaluating and optimising the baseline plan
- Freezing the baseline plan
- Tracking the actual costs
- Comparing the progress and cost to the baseline plan
- Evaluating performance
- Forecasting, analysing and recommending corrective action

Benefits of Project Management

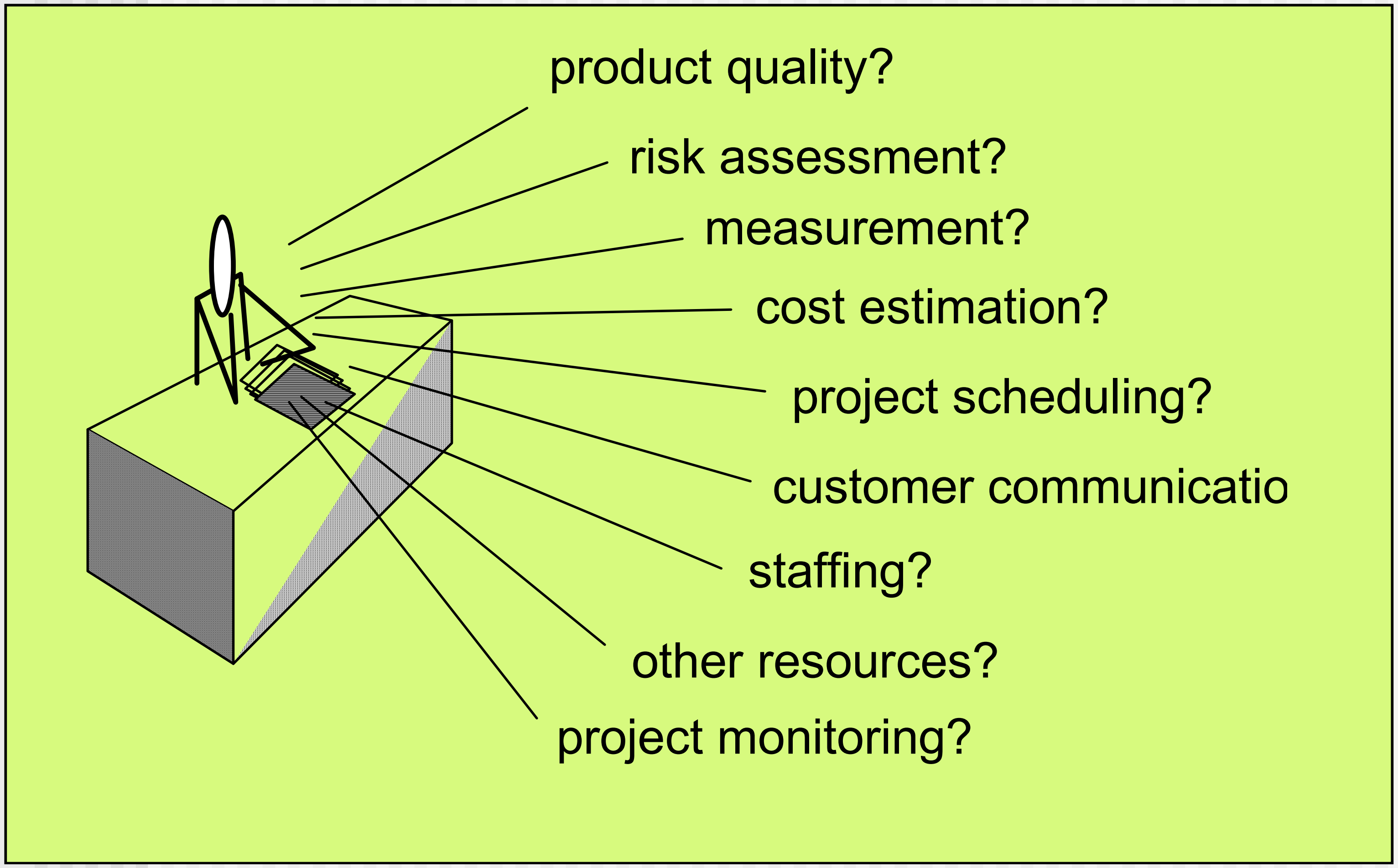
- Identification of function responsibilities to ensure that all activities are accounted for, regardless of personnel turnover
- Minimizing the need for continuous reporting
- Identification of time limits for scheduling
- Identification of a methodology for tradeoff analysis
- Measurement of accomplishment against plans
- Early identification of problems
- Improved estimating capabilities for future planning
- Knowing when objectives cannot be met or will be exceeded

Software Projects

Factors that influence the end result ...

- **size**
- **delivery deadline**
- **budgets and costs**
- **application domain**
- **technology to be implemented**
- **system constraints**
- **user requirements**
- **available resources**

Project Management Concerns



The Four P’s

- People — the most important element of a successful project
- Product — the software to be built
- Process — the set of framework activities and software engineering tasks to get the job done
- Project — all work required to make the product a reality

Project Management Problems

- Resources inadequate
- Meeting (“unrealistic”) deadlines
- Unclear goals/direction
- Team members uncommitted
- Insufficient planning
- Breakdowns in communications
- Changes in goals and resources
- Conflicts between departments or functions

Obstacles in Project Management

- Project complexity
- Customer’s special requirement
- Organizational restructuring
- Project risks
- Changes in technology
- Forward planning and pricing

Project Management Skills

- Communication Skills
- Organizational Skills
- Team Building Skills
- Leadership Skills
- Coping Skills
- Technological Skills

2. People in a Project

Stakeholders

- *Senior managers* who define the business issues that often have significant influence on the project.
- *Project (technical) managers* who must plan, motivate, organize, and control the practitioners who do software work.
- *Practitioners* who deliver the technical skills that are necessary to engineer a product or application.
- *Customers* who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
- *End-users* who interact with the software once it is released for production use.

Software Teams

How to lead?

How to organize?

How to collaborate?

How to motivate?

How to create good ideas?

Team Leader

- The MOI Model
  - **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.

Software Teams

The following factors must be considered when selecting a software project team structure ...

- the difficulty of the problem to be solved
- the size of the resultant program(s) in lines of code or function points
- the time that the team will stay together (team lifetime)
- 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

Agile Teams

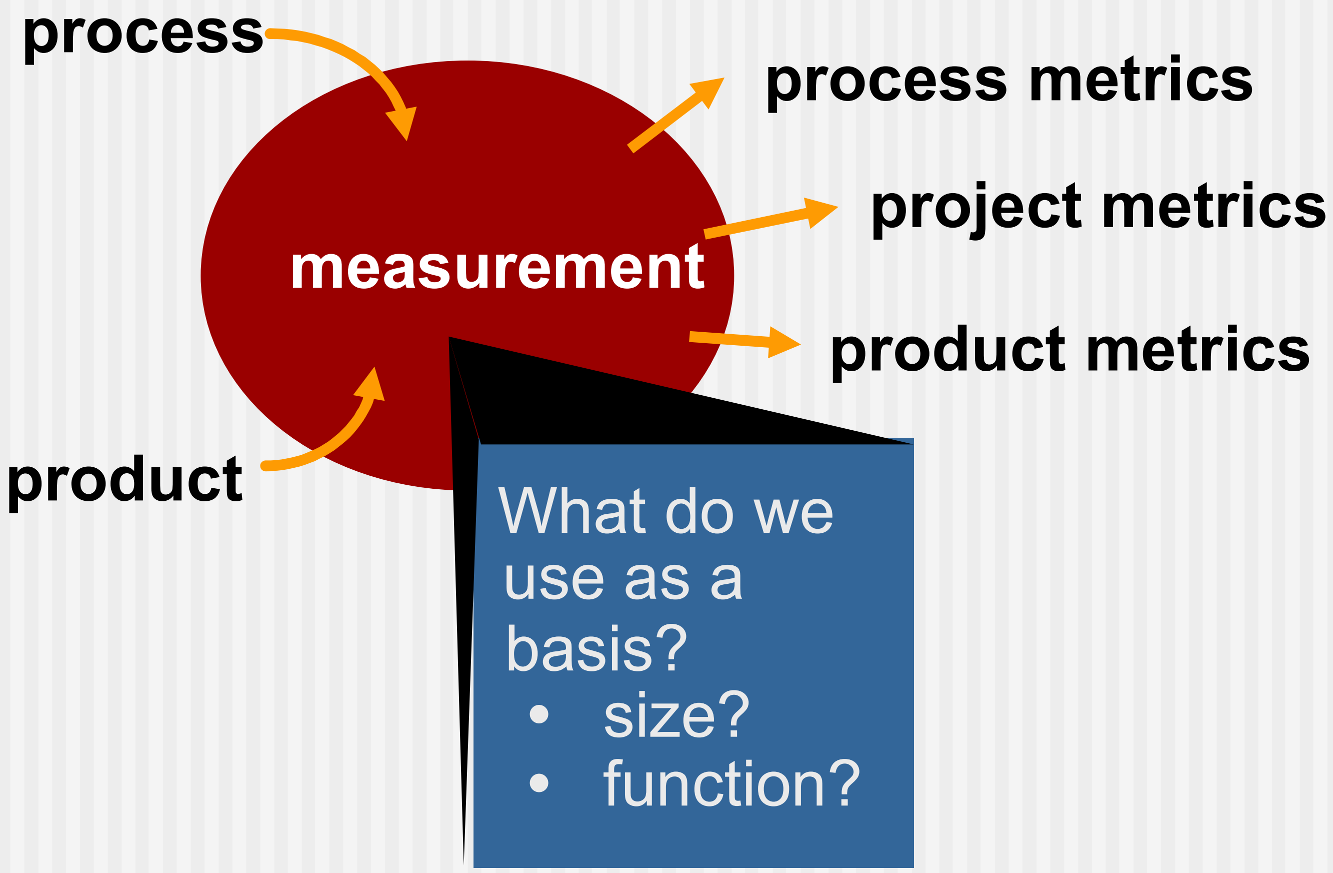
- Team members must have trust in one another.
- The distribution of skills must be appropriate to the problem.
- Mavericks may have to be excluded from the team, if team cohesiveness is to be maintained.
- Team is “self-organizing”
  - An adaptive team structure
  - Uses elements of Constantine’s random, open, and synchronous paradigms
  - Significant autonomy

Team Coordination & Communication

- *Formal, impersonal approaches* include software engineering documents and work products (including source code), technical memos, project milestones, schedules, and project control tools (Chapter 23), change requests and related documentation, error tracking reports, and repository data (see Chapter 26).
- *Formal, interpersonal procedures* focus on quality assurance activities (Chapter 25) applied to software engineering work products. These include status review meetings and design and code inspections.
- *Informal, interpersonal procedures* include group meetings for information dissemination and problem solving and "collocation of requirements and development staff."
- *Electronic communication* encompasses electronic mail, electronic bulletin boards, and by extension, video-based conferencing systems.
- *Interpersonal networking* includes informal discussions with team members and those outside the project who may have experience or insight that can assist team members.

3. Product, Process and Project Metrics

A Good Manager Measures



What do we use as a basis?

- size?
- function?

Why Do We Measure?

- assess the status of an ongoing project
- track potential risks
- uncover problem areas before they go “critical,”
- adjust work flow or tasks,
- evaluate the project team’s ability to control quality of software work products.

Measures, Metrics and Indicators

- A *measure* provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process
- The IEEE glossary defines a *metric* as “a quantitative measure of the degree to which a system, component, or process possesses a given attribute.”
- An *indicator* is a metric or combination of metrics that provide insight into the software process, a software project, or the product itself

Measurement Principles

- The objectives of measurement should be established before data collection begins;
- Each technical metric should be defined in an unambiguous manner;
- Metrics should be derived based on a theory that is valid for the domain of application (e.g., metrics for design should draw upon basic design concepts and principles and attempt to provide an indication of the presence of an attribute that is deemed desirable);
- Metrics should be tailored to best accommodate specific products and processes [Bas84]

Measurement Process

- *Formulation.* The derivation of software measures and metrics appropriate for the representation of the software that is being considered.
- *Collection.* The mechanism used to accumulate data required to derive the formulated metrics.
- *Analysis.* The computation of metrics and the application of mathematical tools.
- *Interpretation.* The evaluation of metrics results in an effort to gain insight into the quality of the representation.
- *Feedback.* Recommendations derived from the interpretation of product metrics transmitted to the software team.

Metrics for the Requirements Model

- Function-based metrics: use the function point as a normalizing factor or as a measure of the "size" of the specification
- Specification metrics: used as an indication of quality by measuring number of requirements by type

Function-Based Metrics

- The *function point metric* (FP), first proposed by Albrecht [ALB79], can be used effectively as a means for measuring the functionality delivered by a system.
- Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity
- Information domain values are defined in the following manner:
  - number of external inputs (EIs)
  - number of external outputs (EOs)
  - number of external inquiries (EQs)
  - number of internal logical files (ILFs)
  - Number of external interface files (EIFs)

Function Points

Information Domain Value	Count		Weighting factor simple	average	complex		
External Inputs (EIs)		3	3	4	6	=	
External Outputs (EOs)		3	4	5	7	=	
External Inquiries (EQs)		3	3	4	6	=	
Internal Logical Files (ILFs)		3	7	10	15	=	
External Interface Files (EIFs)		3	5	7	10	=	
Count total							

Class-Oriented Metrics

Proposed by Chidamber and Kemerer [Chi94]

- weighted methods per class
- depth of the inheritance tree
- number of children
- coupling between object classes
- response for a class
- lack of cohesion in methods

Class-Oriented Metrics

Proposed by Lorenz and Kidd [Lor94]:

- class size
- number of operations overridden by a subclass
- number of operations added by a subclass
- specialization index

Process Measurement

- We measure the efficacy of a software process indirectly.
  - That is, we derive a set of metrics based on the outcomes that can be derived from the process.
  - Outcomes include
    - measures of errors uncovered before release of the software
    - defects delivered to and reported by end-users
    - work products delivered (productivity)
    - human effort expended
    - calendar time expended
    - schedule conformance
    - other measures.
- We also derive process metrics by measuring the characteristics of specific software engineering tasks.

Process Metrics

- Quality-related
  - focus on quality of work products and deliverables
- Productivity-related
  - Production of work-products related to effort expended
- Statistical SQA data
  - error categorization & analysis
- Defect removal efficiency
  - propagation of errors from process activity to activity
- Reuse data
  - The number of components produced and their degree of reusability

Project Metrics

- used to minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
- used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.
- every project should measure:
  - *inputs*—measures of the resources (e.g., people, tools) required to do the work.
  - *outputs*—measures of the deliverables or work products created during the software engineering process.
  - *results*—measures that indicate the effectiveness of the deliverables.

Typical Project Metrics

- Effort/time per software engineering task
- Errors uncovered per review hour
- Scheduled vs. actual milestone dates
- Changes (number) and their characteristics
- Distribution of effort on software engineering tasks

Typical Size-Oriented Metrics

- errors per KLOC (thousand lines of code)
- defects per KLOC
- $ per LOC
- pages of documentation per KLOC
- errors per person-month
- errors per review hour
- LOC per person-month
- $ per page of documentation

Typical Function-Oriented Metrics

- errors per FP
- defects per FP
- $ per FP
- pages of documentation per FP
- FP per person-month

4. Project Planning and Estimation

Software Project Planning

The overall goal of project planning is to establish a pragmatic strategy for controlling, tracking, and monitoring a complex technical project.

Why?

So the end result gets done on time, with quality!

Project Planning Task Set-I

- Establish project objectives
- Define project scopes
- Determine feasibility
- Analyze risks
- Define required resources
  - Determine require human resources
  - Define reusable software resources
  - Identify environmental resources

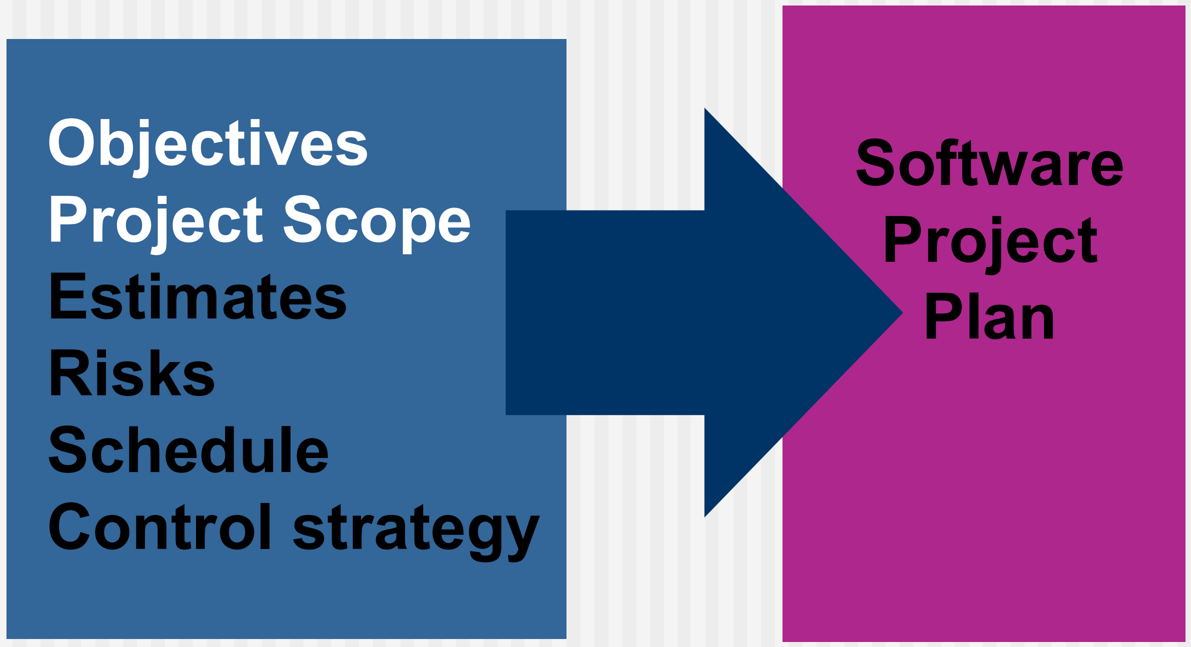
Project Planning Task Set-II

- Estimate cost and effort
  - Decompose the problem
  - Develop two or more estimates using size, function points, process tasks or use-cases
  - Reconcile the estimates
- Develop a project schedule
  - Establish a meaningful task set
  - Define a task network
  - Use scheduling techniques
  - Define schedule tracking mechanisms

Estimation

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

Write it Down!



What are Objectives?

- A series of statements that express the quantitative and qualitative goals to be achieved in the project.
- Must be measureable and unambiguous.
- Each objective should be unique and distinguishable.
- Normally establish 3 objectives (minimum).
- Objectives that are established will act as a guidance on the direction of the project.

To Understand Scope ...

- Understand the customers needs
- Understand the business context
- Understand the project boundaries
- Understand the customer’s motivation
- Understand the likely paths for change
- Understand that ...

Even when you understand, nothing is guaranteed!

What is Scope?

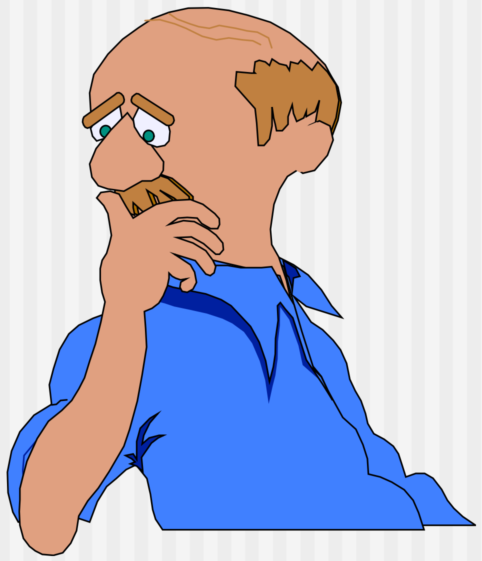
- *Software scope* describes
  - the functions and features that are to be delivered to end-users
  - the data that are input and output
  - the “content” that is presented to users as a consequence of using the software
  - the performance, constraints, interfaces, and reliability that *bound* the system.
- Scope is defined using one of 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.

Project Estimation

- Project scope must be understood
- Elaboration (decomposition) is necessary
- Historical metrics are very helpful
- At least two different techniques should be used
- Uncertainty is inherent in the process

Estimation Techniques

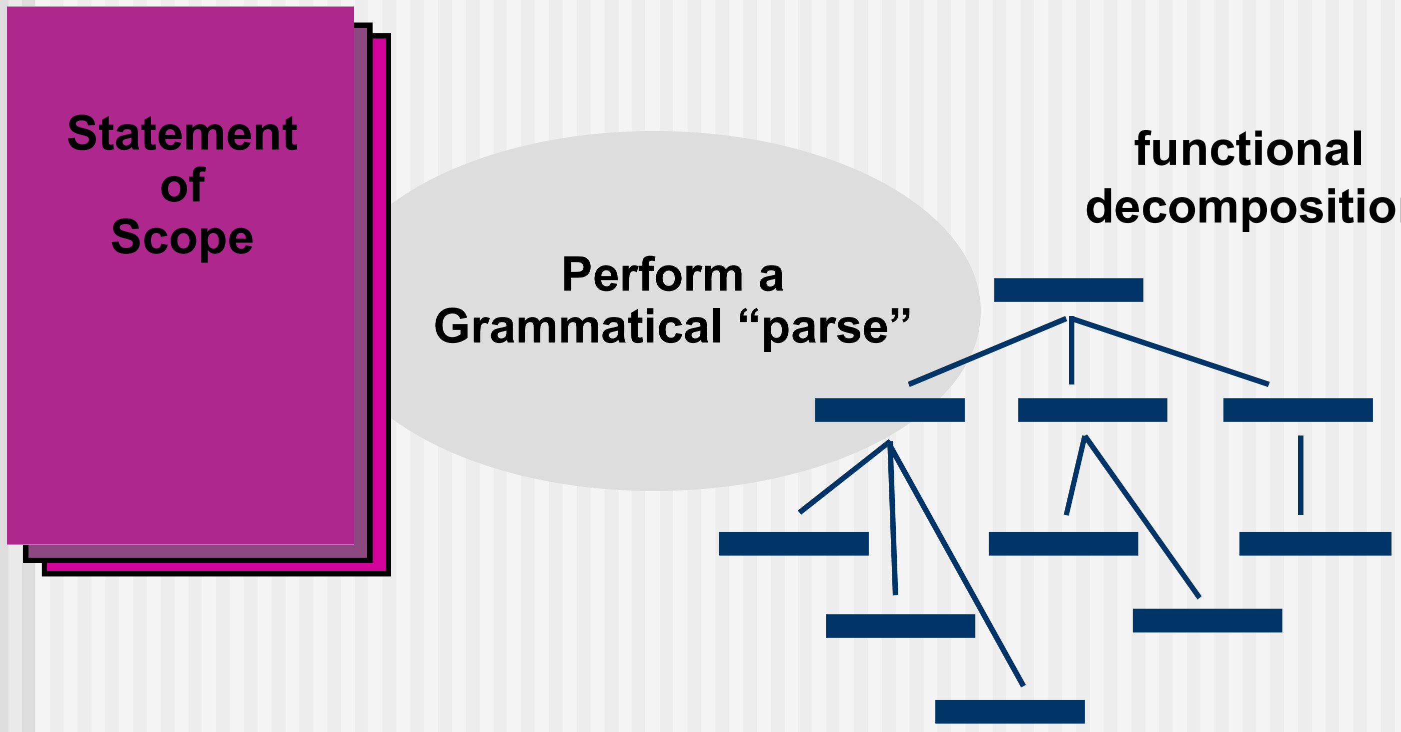
- Past (similar) project experience
- Conventional estimation techniques
    - task breakdown and effort estimates
    - size (e.g., FP) estimates
- Empirical models
- Automated tools



Estimation Accuracy

- 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 dollars (a function of the availability of reliable software metrics from past projects)
  - the degree to which the project plan reflects the abilities of the software team
  - the stability of product requirements and the environment that supports the software engineering effort.

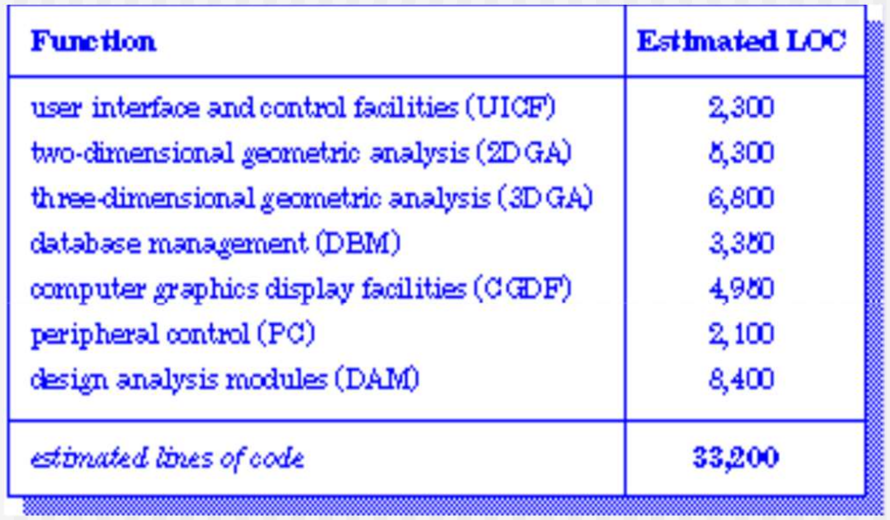
Functional Decomposition



Conventional Methods: LOC/FP Approach

- compute LOC/FP using estimates of information domain values
- use historical data to build estimates for the project

Example: LOC Approach



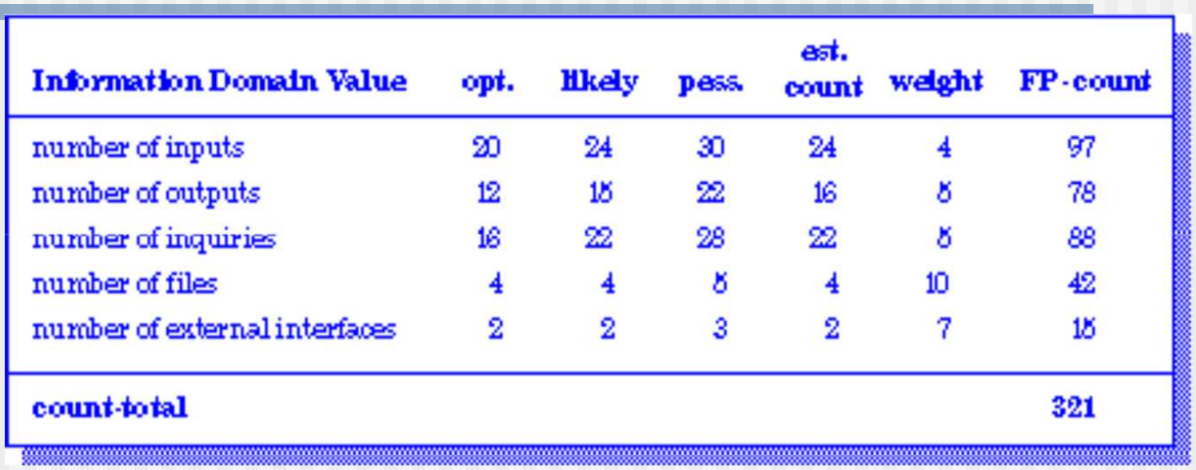
Function	Estimated LOC
user interface and control facilities (UICF)	2,300
two-dimensional geometric analysis (2DGA)	5,300
three-dimensional geometric analysis (3DGA)	6,800
database management (DBM)	3,350
computer graphics display facilities (CGDF)	4,950
peripheral control (PC)	2,100
design analysis modules (DAM)	8,400
estimated lines of code	**33,200**

Average productivity for systems of this type = 620 LOC/pm.

Burdened labor rate =$8000 per month, the cost per line of code is approximately $13.

Based on the LOC estimate and the historical productivity data, the total estimated project cost is **$431,000 and the estimated effort is 54 person-months.**

Example: FP Approach



Information Domain Value	opt.	likely	pess.	est. count	weight	FP-count
number of inputs	20	24	30	24	4	97
number of outputs	12	15	22	16	5	78
number of inquiries	16	22	28	22	5	88
number of files	4	4	5	4	10	42
number of external interfaces	2	2	3	2	7	15
count-total						321

The estimated number of FP is derived:

$$FP_{estimated} = \text{count-total } 3\ [0.65 + 0.01\ 3\ S\ (F_i)]$$

$$FP_{estimated} = 375$$

organizational average productivity = 6.5 FP/pm.

burdened labor rate = \$8000 per month, approximately \$1230/FP.

Based on the FP estimate and the historical productivity data, **total estimated project cost is \$461,000 and estimated effort is 58 person-months.**

5. Project Scheduling

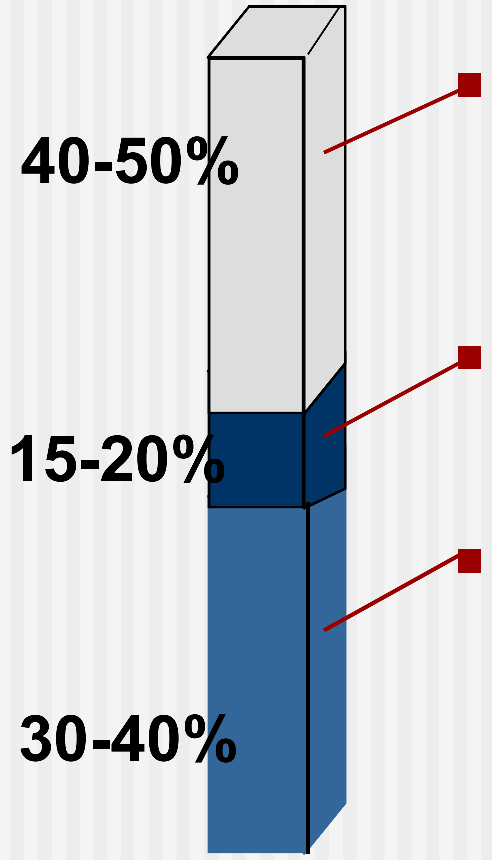
Why Are Projects Late?

- an unrealistic deadline established by someone outside the software development group
- changing customer requirements that are not reflected in schedule changes;
- an honest underestimate of the amount of effort and/or the number of resources that will be required to do the job;
- predictable and/or unpredictable risks that were not considered when the project commenced;
- technical difficulties that could not have been foreseen in advance;
- human difficulties that could not have been foreseen in advance;
- miscommunication among project staff that results in delays;
- a failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem

Scheduling Principles

- compartmentalization—define distinct tasks
- interdependency—indicate task interrelationship
- effort validation—be sure resources are available
- defined responsibilities—people must be assigned
- defined outcomes—each task must have an output
- defined milestones—review for quality

Effort Allocation



- “front end” activities
  - customer communication
  - analysis
  - design
  - review and modification
- construction activities
  - coding or code generation
- testing and installation
  - unit, integration
  - white-box, black box
  - regression

Defining Task Sets

- determine type of project
- assess the degree of rigor required
- identify adaptation criteria
- select appropriate software engineering tasks

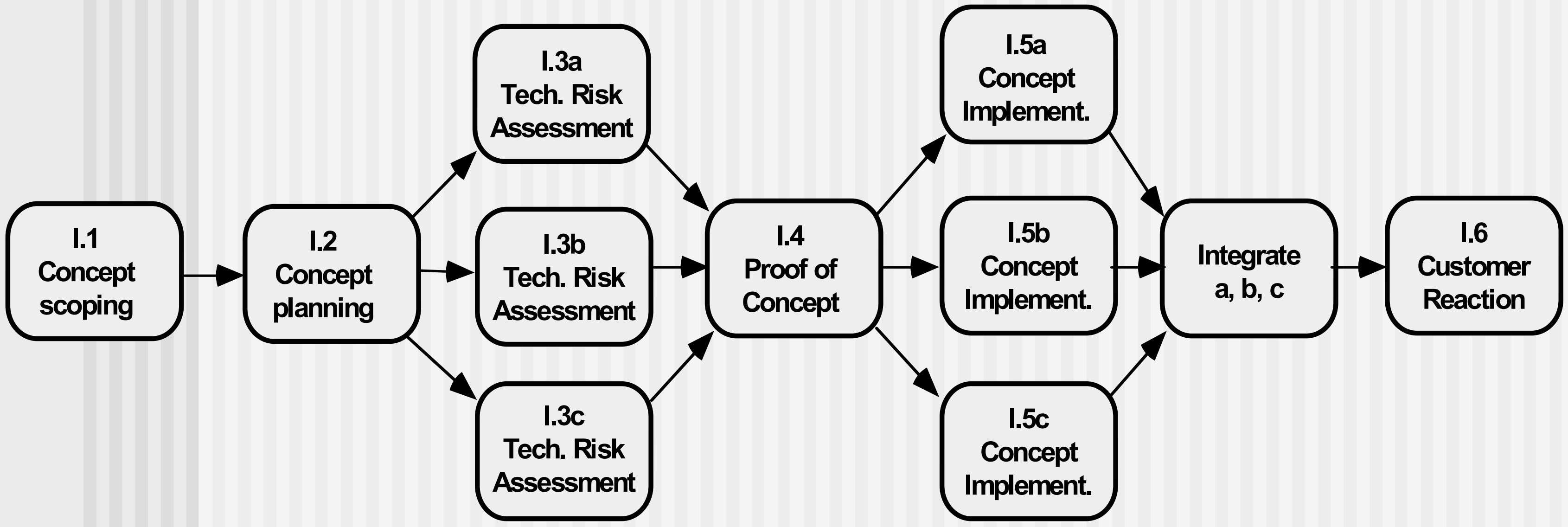
Task Set Refinement

1.1 **Concept scoping** determines the overall scope of the project.

is refined to

```
Task definition:  Task 1.1  Concept Scoping
1.1.1          Identify need, benefits and potential customers;
1.1.2          Define desired output/control and input events that drive the application;
               Begin Task 1.1.2
               1.1.2.1        FTR:  Review written description of need
          FTR indicates that a formal technical review (Chapter 26) is to be conducted.
               1.1.2.2        Derive a list of customer visible outputs/inputs
               1.1.2.3        FTR:  Review outputs/inputs with customer and revise as required;
               endtask Task 1.1.2
1.1.3          Define the functionality/behavior for each major function;
               Begin Task 1.1.3
               1.1.3.1        FTR:  Review output and input data objects derived in task 1.1.2;
               1.1.3.2        Derive a model of functions/behaviors;
               1.1.3.3        FTR:  Review functions/behaviors with customer and revise as required;
               endtask Task 1.1.3
1.1.4          Isolate those elements of the technology to be implemented in software;
1.1.5          Research availability of existing software;
1.1.6          Define technical feasibility;
1.1.7          Make quick estimate of size;
1.1.8          Create a Scope Definition;
endTask definition:   Task 1.1
```

Define a Task Network



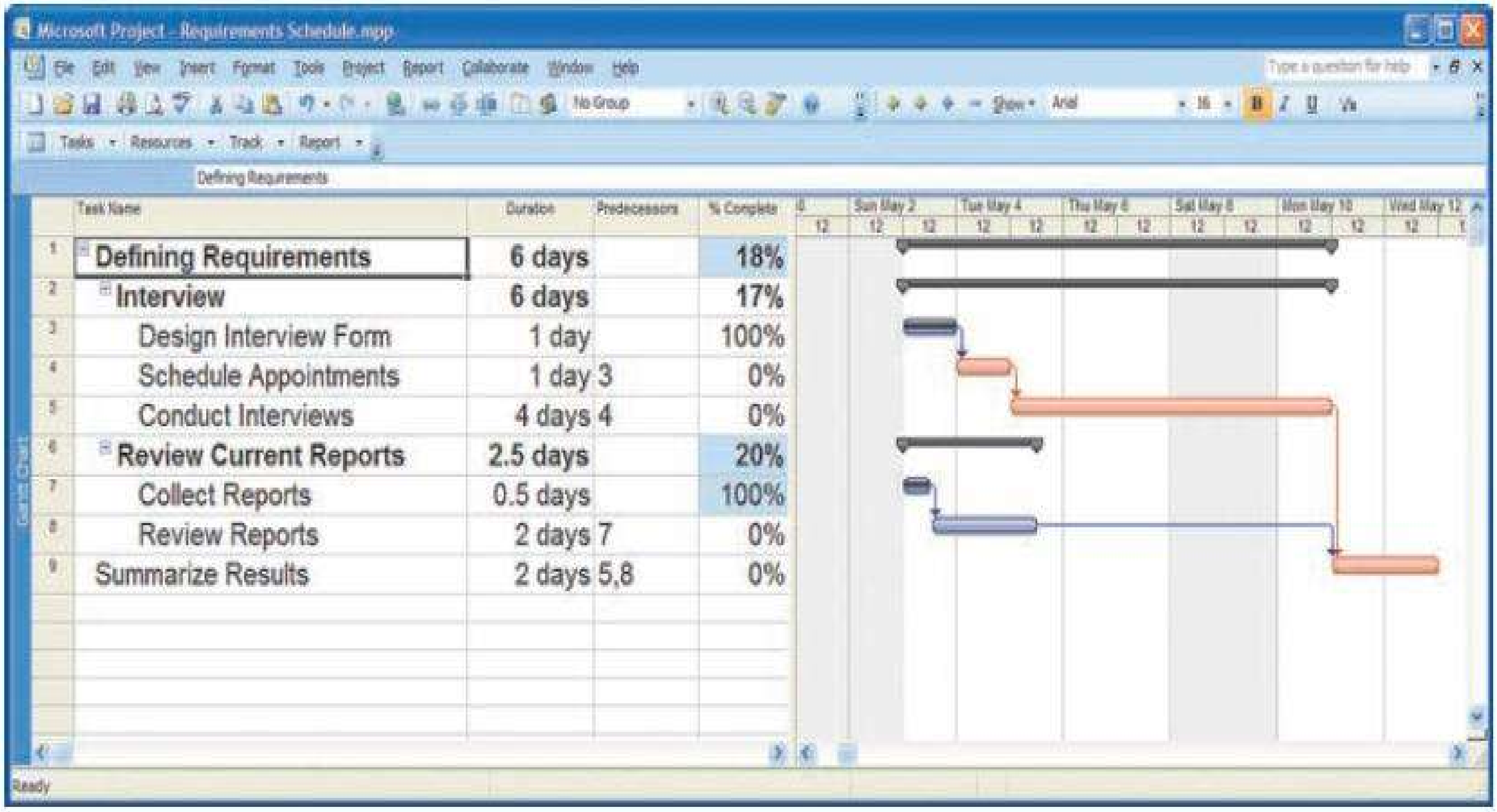
Three I.3 tasks are applied in parallel to 3 different concept functions

Three I.3 tasks are applied in parallel to 3 different concept functions

Scheduling Techniques

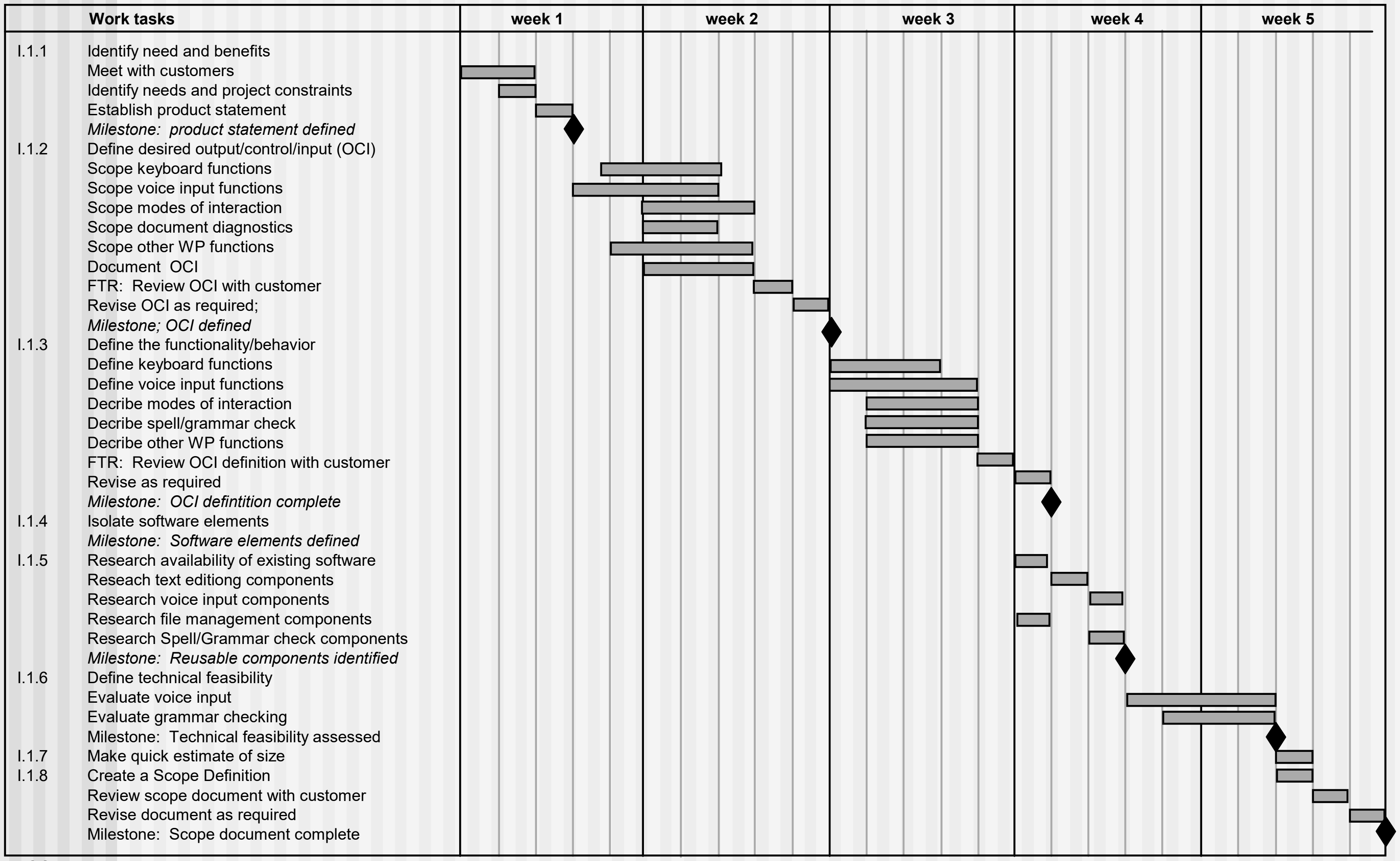
- Gantt Chart
- Timeline Chart
- Network Diagram
- PERT Analysis
- Critical Path Scheduling

Gantt Chart



	Task Name	Duration	Predecessors	% Complete
1	**Defining Requirements**	**6 days**		**18%**
2	**Interview**	**6 days**		**17%**
3	Design Interview Form	1 day		100%
4	Schedule Appointments	1 day	3	0%
5	Conduct Interviews	4 days	4	0%
6	**Review Current Reports**	**2.5 days**		**20%**
7	Collect Reports	0.5 days		100%
8	Review Reports	2 days	7	0%
9	Summarize Results	2 days	5,8	0%

Use Automated Tools to Derive a Timeline Chart

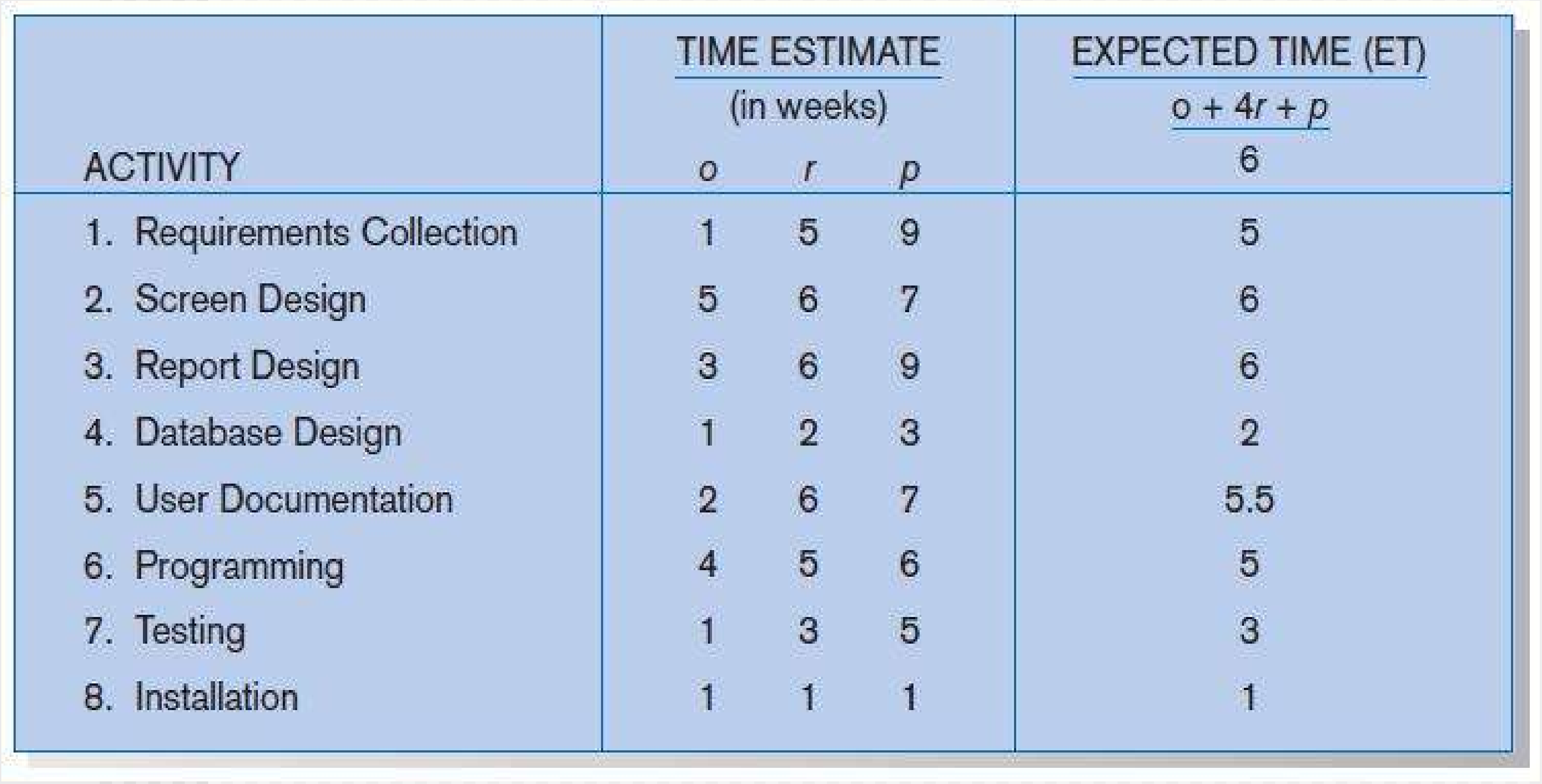


	Work tasks
I.1.1	Identify need and benefits
	Meet with customers
	Identify needs and project constraints
	Establish product statement
	Milestone: product statement defined
I.1.2	Define desired output/control/input (OCI)
	Scope keyboard functions
	Scope voice input functions
	Scope modes of interaction
	Scope document diagnostics
	Scope other WP functions
	Document OCI
	FTR: Review OCI with customer
	Revise OCI as required;
	Milestone; OCI defined
I.1.3	Define the functionality/behavior
	Define keyboard functions
	Define voice input functions
	Decribe modes of interaction
	Decribe spell/grammar check
	Decribe other WP functions
	FTR: Review OCI definition with customer
	Revise as required
	Milestone: OCI defintition complete
I.1.4	Isolate software elements
	Milestone: Software elements defined
I.1.5	Research availability of existing software
	Reseach text editiong components
	Research voice input components
	Research file management components
	Research Spell/Grammar check components
	Milestone: Reusable components identified
I.1.6	Define technical feasibility
	Evaluate voice input
	Evaluate grammar checking
	Milestone: Technical feasibility assessed
I.1.7	Make quick estimate of size
I.1.8	Create a Scope Definition
	Review scope document with customer
	Revise document as required
	Milestone: Scope document complete

PERT Analysis

- PERT: Program Evaluation Review Technique
- Technique that uses optimistic (*o*), pessimistic (*p*), and realistic (*r*) time estimates to determine expected task duration.
- Formula for Estimated Time (ET): **$ET = (o + 4r + p)/6$**

Schedule Tracking

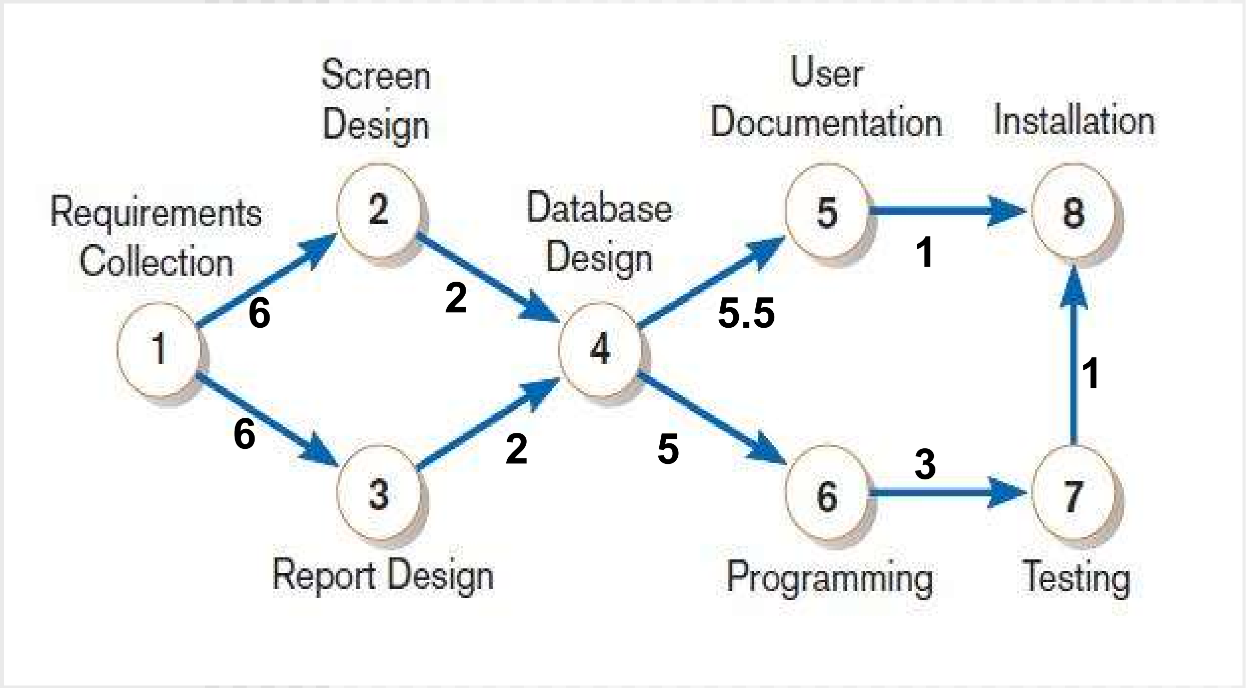


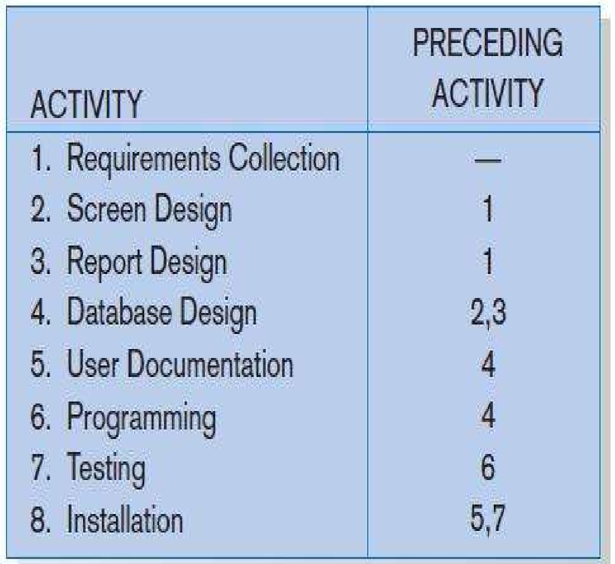
	TIME ESTIMATE (in weeks)			EXPECTED TIME (ET) $\frac{o + 4r + p}{6}$
ACTIVITY	o	r	p	
1. Requirements Collection	1	5	9	5
2. Screen Design	5	6	7	6
3. Report Design	3	6	9	6
4. Database Design	1	2	3	2
5. User Documentation	2	6	7	5.5
6. Programming	4	5	6	5
7. Testing	1	3	5	3
8. Installation	1	1	1	1

Critical Path Scheduling

- A scheduling technique whose order and duration of a sequence of task activities directly affect the completion
- *Critical path:* the shortest time in which a project can be completed
- *Slack time:* the time an activity can be delayed without delaying the project

Critical Path Scheduling





ACTIVITY	PRECEDING ACTIVITY
1. Requirements Collection	—
2. Screen Design	1
3. Report Design	1
4. Database Design	2,3
5. User Documentation	4
6. Programming	4
7. Testing	6
8. Installation	5,7

Critical Path Scheduling

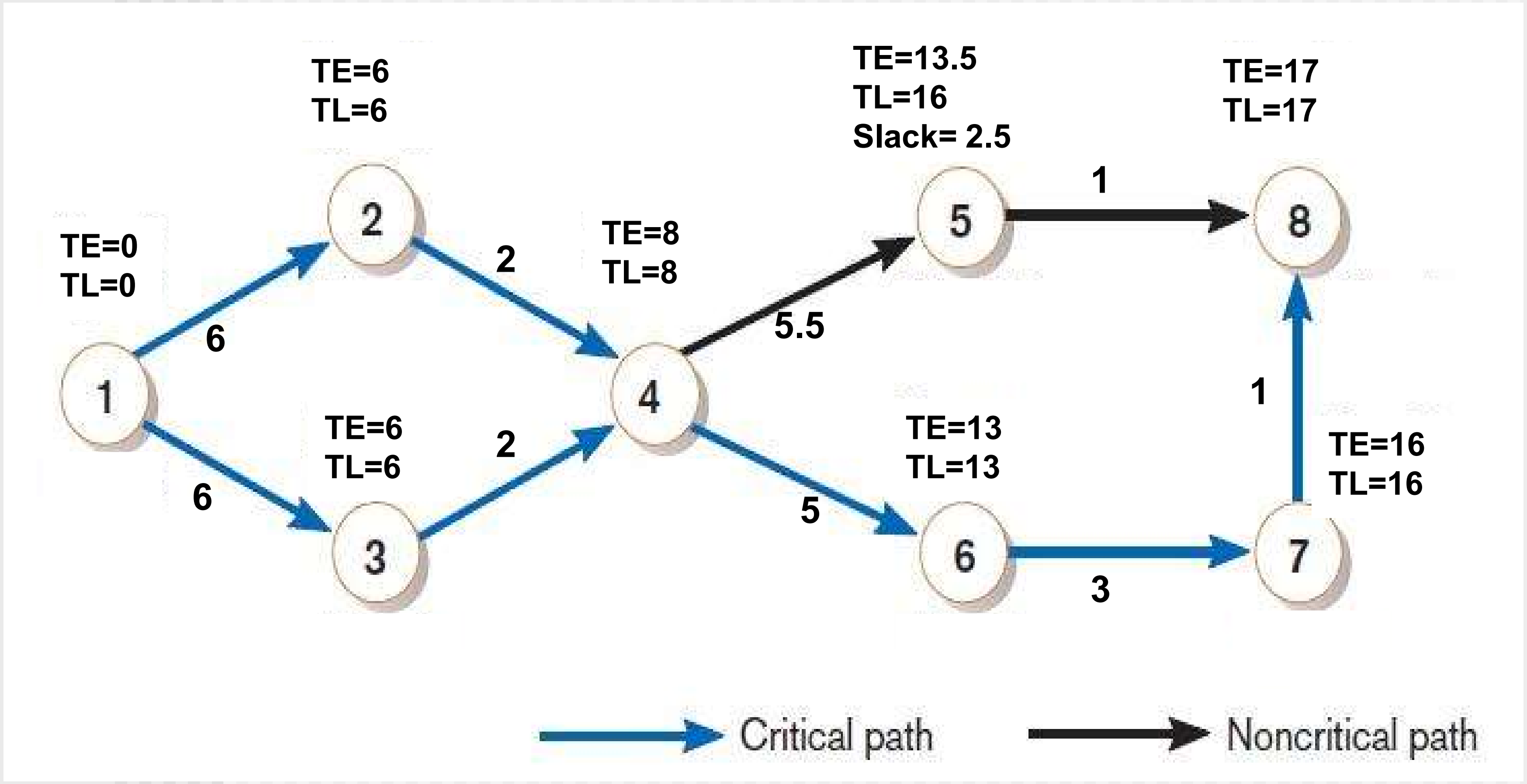
- Calculate the **earliest** possible completion time (TE) for each activity by summing the activity times in the longest path to the activity. This gives total expected project time.

- Calculate the **latest** possible completion time (TL) for each activity by subtracting the activity times in the path following the activity from the total expected time. This gives slack time for activities.

- **Critical path** – contains no activities with slack time.

- Activity (with slack time) can begin late without affecting the project completion time.

Critical Path Scheduling



Schedule Tracking

- conduct **periodic project status meetings** in which each team member reports progress and problems.
- evaluate the results of all reviews conducted throughout the software engineering process.
- determine whether formal project milestones (the diamonds shown in previous slide) have been accomplished by the scheduled date.
- compare actual start-date to planned start-date for each project task.
- meet informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon.