COMP 3111
SOFTWARE ENGINEERING

MANAGING SOFTWARE DEVELOPMENT

LEARNING OBJECTIVES

- To know the principal tasks of software project managers.
- To understand the need for project planning in all software projects.
- To understand some of the requirements for staffing and scheduling in software projects.
- To know some techniques for estimating the size and cost of software development.
- To understand the importance of project tracking and control.

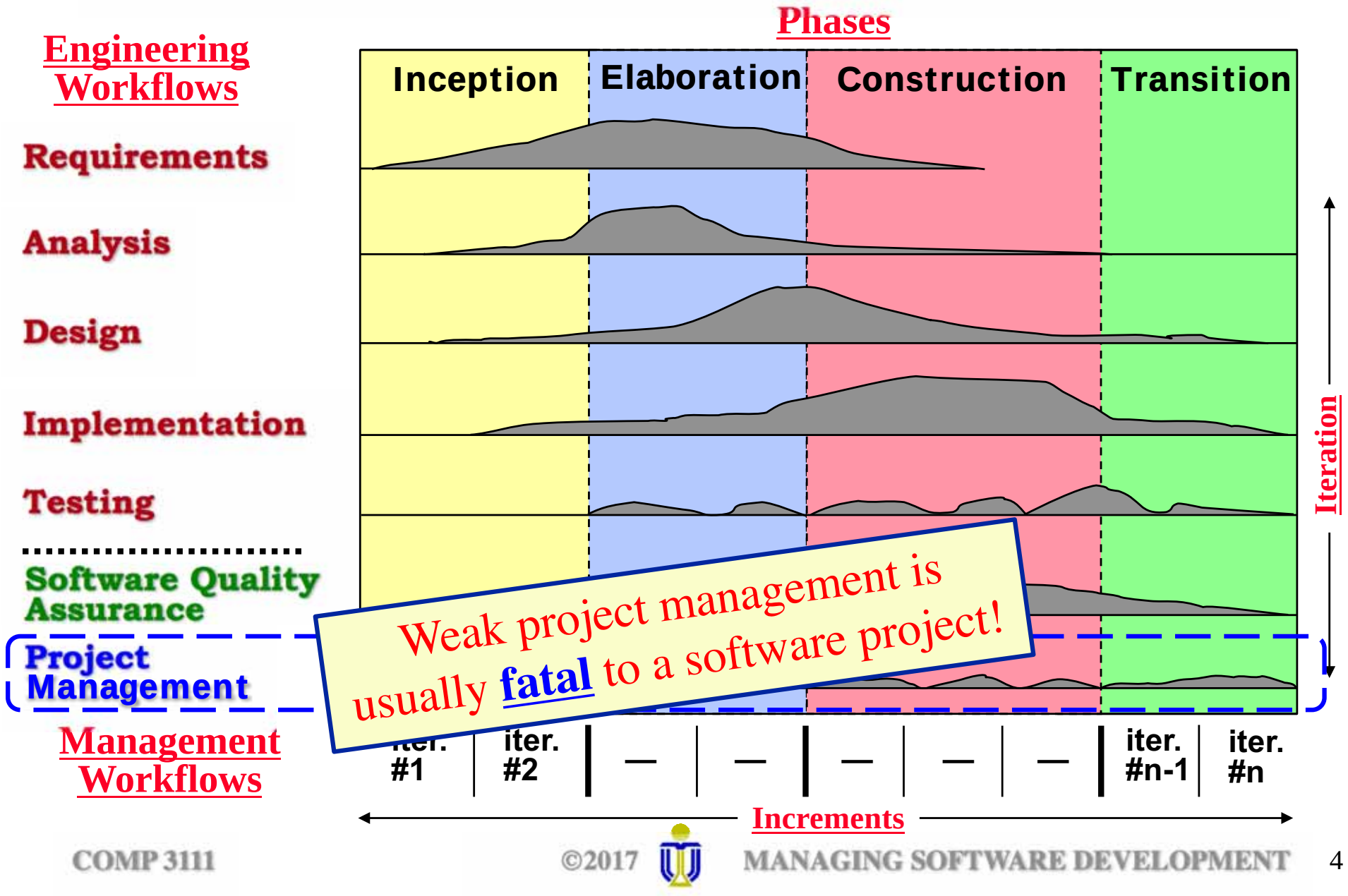
MANAGING SOFTWARE DEVELOPMENT: OUTLINE

Project Management
- Life Cycle Role
- The Challenge

The Software Development Plan
- Deliverables
- Development Environment
- Work Breakdown Structure (WBS)
- Staffing and Organization
- Schedules
- Estimates
- Metrics Plan
- Risk Planning
- Time-phased Budget

Project Tracking and Control

PROJECT MANAGEMENT: LIFE CYCLE ROLE



PROJECT MANAGEMENT: THE CHALLENGE

Up front we need to:

1. create a development plan while having to deal with:
   - incomplete knowledge (requirements, people, etc.).
   - limited resources (time, money, skills, etc.).

2. decide about many development issues.
   - What features are required?
   - Whether to build or buy?
   - What resources are required?
   - What are the risks?
   - What are the tasks to be done?
   - What effort to expend?
   - What schedule to follow?
   - What development tools to use?

   ⋮

Planning the development is a *continuous process*; we need to both "plan the work" and "work the plan".

THE SOFTWARE DEVELOPMENT PLAN (SDP)

The **SDP** documents the scope of the development effort and how the project will be managed.

☞ **The SDP defines the project!**

- Developing the **SDP** is a constrained optimization problem with incomplete data.

- Input for developing the **SDP** is needed from:
  - development manager → organizes project, tasks, budget, etc.
  - experienced system architect → designs top-level system structure; estimates project's technical size
  - expert user or domain expert → provides requirements understanding

DELIVERABLES

The **SDP** specifies what should be delivered to who and when.

Client products → given to the client

- executable code
- user manuals
- help files
- installation scripts
- installation manuals
- developers manuals
- tutorials/examples
- templates
- license managers

Process artifacts → outcomes of the development process

- system requirements, analysis, design specifications
- object design files
- source code

Internal deliverables → of continuing value to the development organization

- source code libraries
- test libraries
- make files
- problem report database

Services → additional deliverables for the client

- training
- consulting
- installation
- on-site support
- customization

☞ **The deliverables affect staffing and team organization.**

DEVELOPMENT ENVIRONMENT

The **SDP** specifies the hardware and software development tools appropriate for the project.

NOTE: development tools ≠ development process!

- Tools are effective *only* if they make well-understood processes more efficient!

- In choosing a development support tool we need to evaluate:
  - support of the lifecycle:

    UML; management oversight and control; architectural control; collaboration support; developer efficiency; library integration; documentation support
  - risk of adoption to both cost and schedule:

    external cost; internal cost; time loss; product instability; investment protection

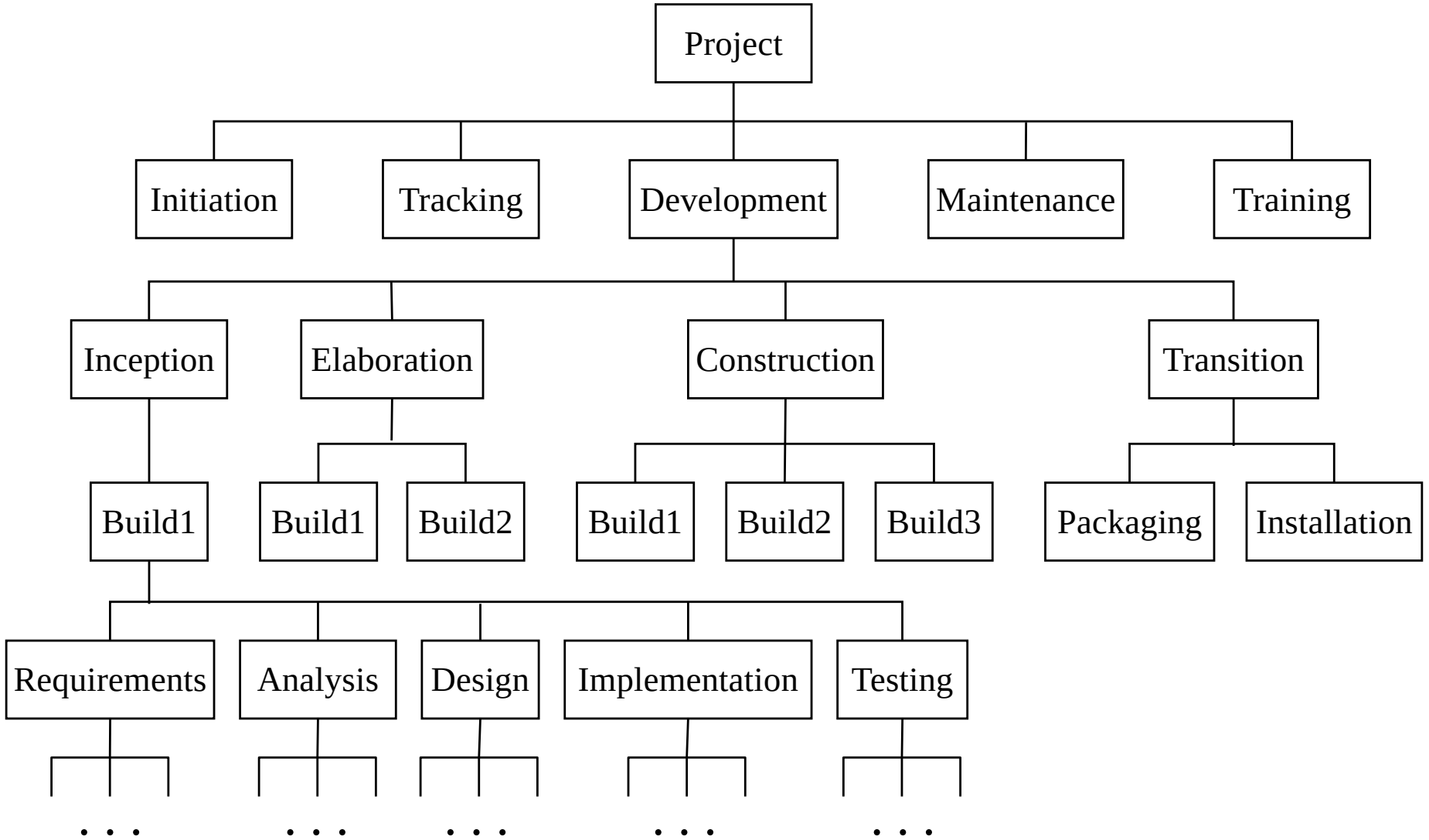
WORK BREAKDOWN STRUCTURE (WBS)

The **SDP** breaks the project into tasks/sub-tasks, so as to ease planning (estimating, monitoring, etc.) → ***divide and conquer***.

- Usually shown as a tree structure.
- Identifies all the activities/tasks required to complete the project.
- Estimates resources required for each leaf node and then "rolls-up" to get an estimate for the entire project.
- Used to track budgets (cost of task) and schedules (time to do task).

- The WBS should allow each task to be:
  - easily planned → Each task has a well-defined start and end.
  - easily assigned to individuals/teams.
  - tracked to monitor progress and to know who is working on it.
  - budgeted so that costs can be tracked.
  - of the right granularity → too small (hard to track); too large (hard to control costs and measure progress).

WORK BREAKDOWN STRUCTURE: EXAMPLE



STAFFING AND ORGANIZATION

The **SDP** defines a (hierarchical) project organization.

- roles and responsibilities; number of staff in each role; teams

☞ **One project member should have experience with a similar system!**

The team organization should:

- be modular to limit communication and complexity of interaction.
- assign clear responsibilities to each team member.
- form teams to have responsibility for the design and implementation of one or more subsystems.
- identify a PIC for each subsystem and the system.

The *key to success* is achieving the right level of communication!

SCHEDULES

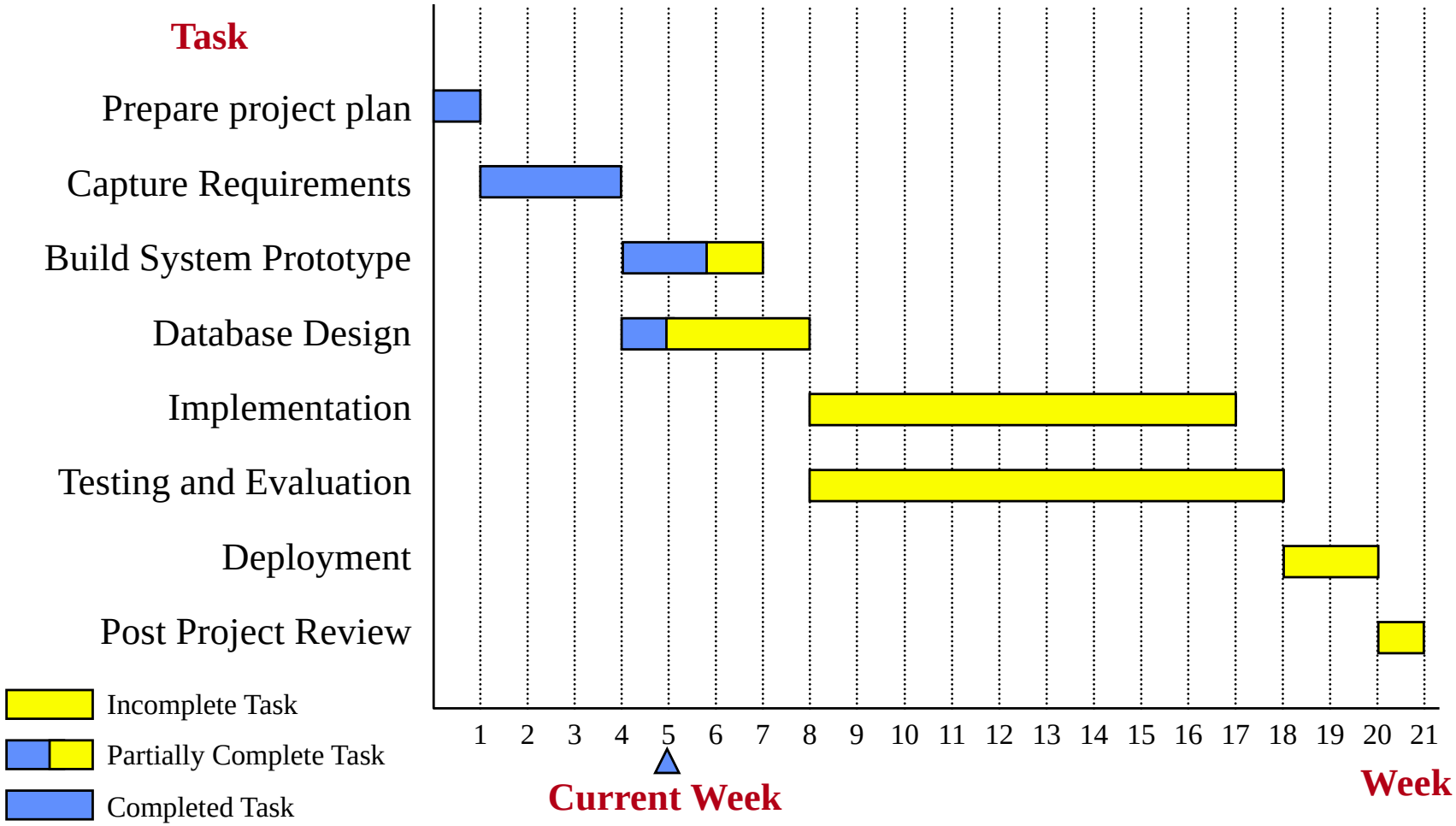
The **SDP** defines the project schedule (i.e., when things happen).

- **task ordering** → dependencies (sequential, parallel).
- **time estimates for each task** → start time, likely duration.
- **resource assignment** → people, hardware, software.
- **milestones** → important management decision points.
- **deliverables** → specifications, documents, code, etc.
- **critical path** → the chain of tasks that determine project duration.

- Usually three levels of schedule are maintained:
  - **master schedule**: for management, client communication → **rigid**.
  - **macroschedule**: for day-to-day project management → **semi-rigid**.
  - **microschedule**: for team management → **highly flexible**.

☞ **Gantt, PERT and burndown charts are commonly used to manage schedules.**

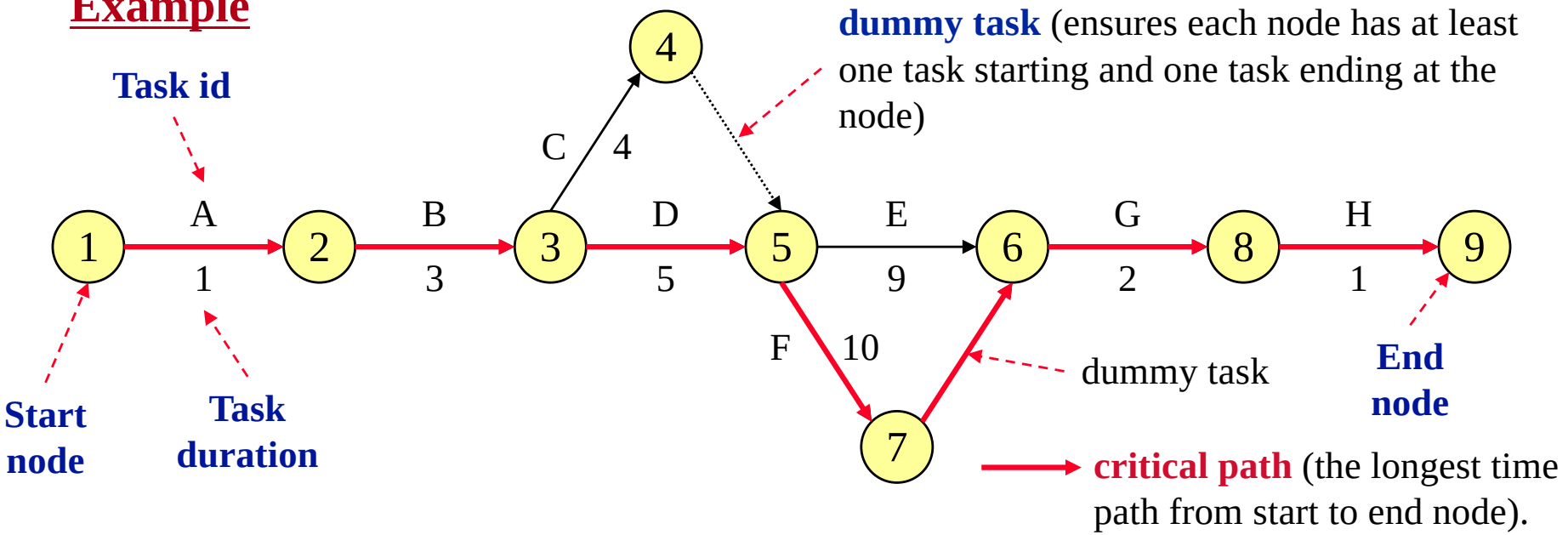
SCHEDULES: GANTT CHART



SCHEDULES: PERT CHART

A *PERT (Program Evaluation and Review Technique) chart* is a graphical representation of project tasks laid out in the form of a *critical path network*.

Example



The overall schedule depends on the critical path.

SCHEDULES: PERT CHART EXAMPLE

Task	Id	Duration	Precedents
Prepare project plan	A	1	-
Capture Requirements	B	3	A
Build System Prototype	C	4	B
Database Design	D	5	B
Implementation	E	9	C, D
Testing and Evaluation	F	10	C, D
Deployment	G	2	E, F
Post Project Review	H	1	G

Node	Earliest Completion Time (ECT)	Latest Completion Time (LCT)	Slack
1	0	0	0
2 (Task A)	1	1	0
3 (Task B)	4	4	0
4 (Task C)	8	9	1
5 (Task D)	9	9	0
6 (Task E)	18	19	1
7 (Task F)	19	19	0
8 (Task G)	21	21	0
9 (Task H)	22	22	0

Slack tells us how much the task can be delayed without putting the project behind schedule.

ESTIMATES

The **SDP** usually provides estimates of:

- size (lines of code (LOC), number of subsystems, number of classes, etc.)
- effort (persons X duration)
- duration (months until delivery)
- productivity (size / effort)
- development cost (labour)

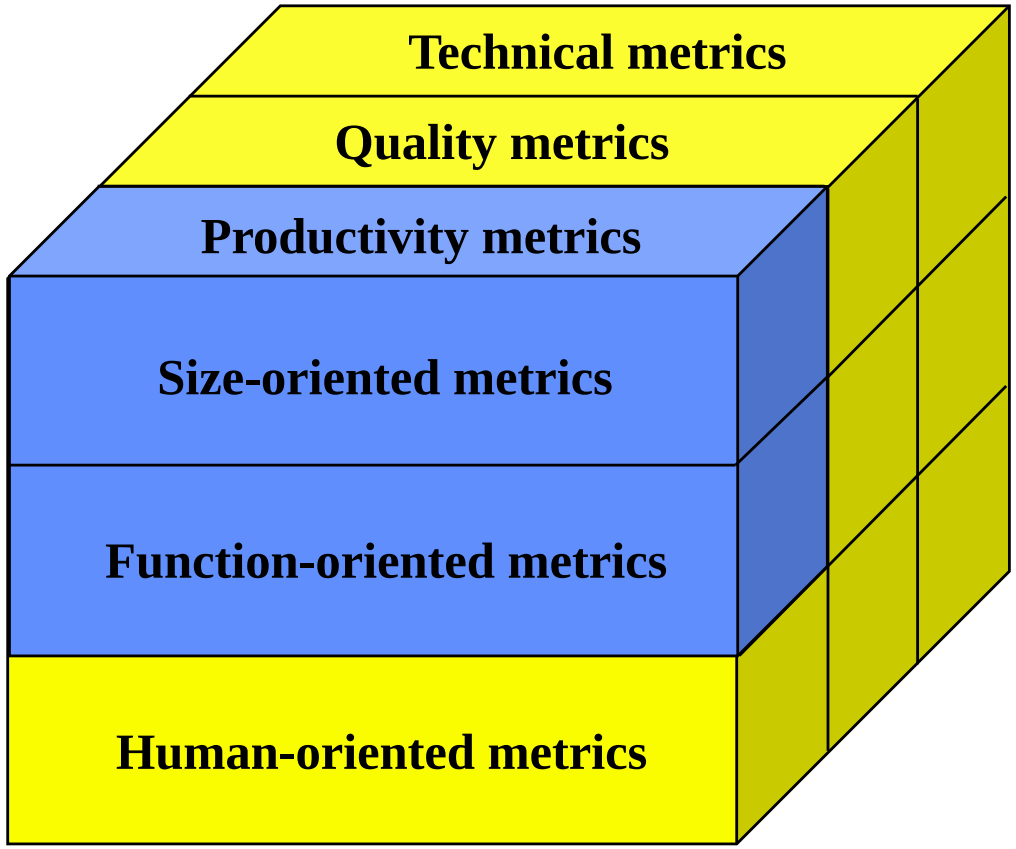
- Estimating is based on:
  - experience
  - historical data
  - models
  - **courage**!

Estimating carries inherent risk.
(Which we try to minimize as much as possible.)

- The risk associated with estimating is reduced if we:
  - establish project scope in advance.
  - use software metrics from past projects.
  - divide and conquer → break into small parts, estimate and sum.

ESTIMATES: USING SOFTWARE METRICS

We can collect many types of metrics about many aspects of software.



What metrics are useful for estimating
and how can they be used for estimating?

ESTIMATES: USING SOFTWARE METRICS

productivity metrics – focus on the output of the software engineering process

quality metrics – indicate how closely the software conforms to implicit and explicit client requirements (i.e., fitness for use)

technical metrics – focus on the properties of the software rather than the process through which the software was developed

size-oriented metrics – direct measures of software development output and quality

function-oriented metrics – indirect measures of software properties

human-oriented metrics – provide information on the manner in which people develop software and human perceptions about the effectiveness of tools and methods

ESTIMATES: USING SIZE-ORIENTED METRICS

- **Estimates are based on data from past software projects.**

To facilitate estimating, we can collect the following project data:

project	effort	$K	KLOC	pages	errors	people
A231	24	168	12.1	365	29	3
B752	62	440	27.2	1224	86	5
C812	43	314	20.2	1050	54	6

To estimate for the current project, we find a "similar" project in the above table and use its data to calculate:

Productivity = KLOC/effort	Quality = errors/KLOC
Cost = $K/KLOC	Documentation = pages/KLOC

☞ **What is the problem with this approach?**

ESTIMATES:
USING FUNCTION-ORIENTED METRICS

- Estimates are based on properties of the project software.

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

$$FP = \text{count-total} * [0.65 + 0.01 * \text{sum}(F_i)]$$

Productivity = FP/effort	Quality = errors/FP
Cost = $/FP	Documentation = pages/FP

ESTIMATES:
USING FUNCTION-ORIENTED METRICS

Number of user inputs – counts each user input that provides distinct application-oriented data to the software (distinguished from inquiries)

Number of user outputs – counts each user output that provides application-oriented output to the user (e.g., reports, screens, error messages, etc.; individual data items are not counted)

Number of user inquiries – counts each online input that results in the generation of some immediate response in the form of an online output

Number of files – counts each logical master file (i.e., a logical grouping of data that may be one part of a large database or a separate file)

Number of external interfaces – counts each machine-readable interface (e.g., data files on disk) that is used to transmit information to another system

ESTIMATES:
USING FUNCTION-ORIENTED METRICS

0	1	2	3	4	5
No influence	Incidental	Moderate	Average	Significant	Essential

F_1 Does the system require reliable backup and recovery?
F_2 Are data communications required?
F_3 Are there distributed processing functions?
F_4 Is performance critical?
F_5 Will the system run in an existing, heavily utilized operational environment?
F_6 Does the system require on-line data entry?
F_7 Does the on-line data entry require the input transaction to be built over multiple screens or operations?
F_8 Are the master files updated on-line?
F_9 Are the inputs,, outputs, files, or inquiries complex?
F_{10} Is the internal processing complex?
F_{11} Is the code designed to be reusable?
F_{12} Are conversion and installation included in the design?
F_{13} Is the system designed for multiple installations in different organizations?
F_{14} Is the application designed to facilitate change and ease of use by the user?

ESTIMATES: OTHER METHODS

System-level Analogy

- Use experience from a previous similar development.
  - May use Delphi technique - average 3 or more estimates.

Pert Estimation

- Each expert provides a range of values, typically:
  - optimistic
  - most likely
  - pessimistic
- The **expected value** is computed as a *weighted average* of optimistic (o), most likely (m) and pessimistic (p)

$$E = (o + 4m + p)/6 \qquad StdDev = (p - o)/6$$

☞ **StdDev is a measure of schedule and budget risk.**

- The actual size will fall between (E - StdDev) and (E + StdDev) 68% of the time.

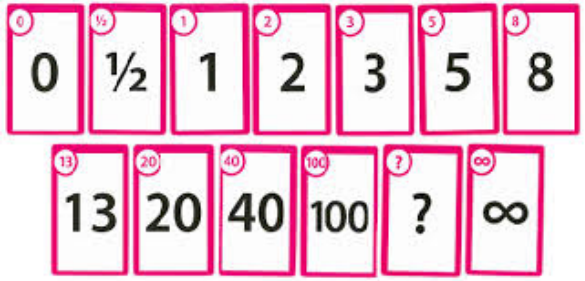
ESTIMATES: OTHER METHODS (CONT'D)

Planning Poker

- Team leader provides overview of feature to be estimated.
- Team members clarify assumptions and risks.

☞ **No numbers should be mentioned.**

- Team members make estimates by playing numbered cards face-down to the table, instead of speaking them aloud.



- The cards are all revealed simultaneously and the estimates are then discussed.

ESTIMATES: OTHER METHODS (CONT'D)

Parametric Models

- Use parametric formulas, empirically derived from a limited sample of projects, to predict a project resource (e.g., effort (E), project duration (D), etc.).

  Examples: Constructive Cost Model (COCOMO)

  Putnam Estimation Model

ESTIMATES: NOTES

- It is essential to have experienced developers do estimating.
- Estimation should always be performed in more than one way and the results cross-checked.
- Incomplete and imprecise requirements hinder accurate cost estimation.
- A cost estimation model is doing well if it can estimate software development costs within 20% of actual costs, 70% of the time on its "own turf".

METRICS PLAN

- For managing the project the **SDP** should:
  - identify which metrics to collect and how to collect them (i.e., tools and procedures to use).
- Project management metrics are usually related to size:
  - e.g., number of use cases; number of classes; lines of code.
- Need to compare planned sizes with current sizes to determine:
  - progress: How much of the planned development is completed?
  - stability: How much change has there been in project requirements and estimates?

RISK PLANNING

The **SDP** plans for risk by trying to:

- foresee **what** can go wrong (*before* it happens).
- estimate its **likelihood** of happening and its likely **impact**.
- develop cost-effective **contingency plans**. *(Action if it happens?)*

- Doing this well is an important quality of a good manager.

- Risk planning is related to preventive management (i.e., determine the risk and execute preventive action *before* the problem occurs).

TIME-PHASED BUDGET

The **SDP** specifies a time-phased budget that details:

- when the project's budget is planned to be spent.
- what is expected to be accomplished at each level of expenditure.

Manpower will likely be the major cost.

- To each WBS item costs are assigned based on duration, staffing level, and cost for each type of staff.

BUT, there are also other costs, such as

- training – software licenses – hardware – etc.
- plus some reserve (between 10-15%)

You need to track the spending!

Compare planned against actual money spent and planned against actual completion ***regularly***.

PROJECT TRACKING AND CONTROL

"Software projects fall behind schedule one day at a time."

- There needs to be constant, consistent, inoffensive monitoring of project activities.

  ☞ **The primary purpose of monitoring is to make sure the project is meeting the budget and schedule.**

- Change is almost inevitable despite the best efforts to minimize it.

  ☞ **The key is to handle it in a controlled manner → *Apply SCM!***

"Adding manpower to a late software project makes it later."

***The Mythical Man-Month*, Frederick P. Brooks**

PROJECT TRACKING & CONTROL: METHODS

- Hold periodic project status meetings → daily, weekly, monthly.
- Do project reviews and evaluate the results of each review.
- Check if milestones / key performance indicators (KPIs) are accomplished as planned.
- Compare the actual budget with the planned budget and the actual start dates with the planned start dates for activities.
- Have informal chats with project staff to obtain subjective assessments of the progress to date and problems on the horizon.

MANAGING SOFTWARE DEVELOPMENT: SUMMARY

Manage the process,
don't let the process manage you.

Khoa Nguyen, *CEO Videoserver*