

Management

Peter Dolog
dolog [at] cs [dot] aau [dot] dk
5.2.47
Information Systems
March 26, 2008

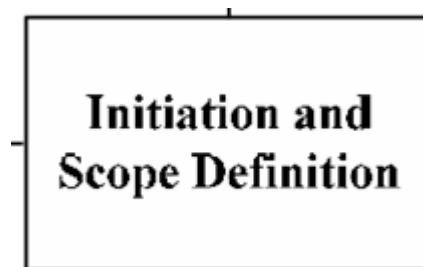
Learning Goals

To apply managements concepts

- To understand terminology
- To be able to select and apply various discussed techniques

To apply risk management concepts

- To understand risk management concepts
- To be able to apply it as needed in your future practice



Determination
► and Negotiation
of Requirements

► Feasibility
Analysis

Process for the
► Review and
Revision of
Requirements

➔ **Process Planning**

➔ **Determine Deliverables**

➔ **Effort, Schedule and Cost Estimation**

➔ **Resource Allocation**

➔ **Risk Management**

➔ **Quality Management**

➔ **Plan Management**

Software Engineering Management

Initiation and Scope Definition

Software Project Planning

Software Project Enactment

Review and Evaluation

Closure

SW Engineering Measurement

Software Project Planning

Software Engineering
Management

Initiation and
Scope Definition

Software Project
Planning

Software Project
Enactment

Review and
Evaluation

Closure

SW Engineering
Measurement

Software Project
Enactment

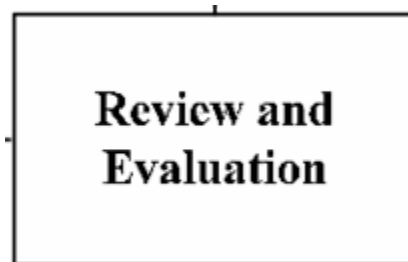
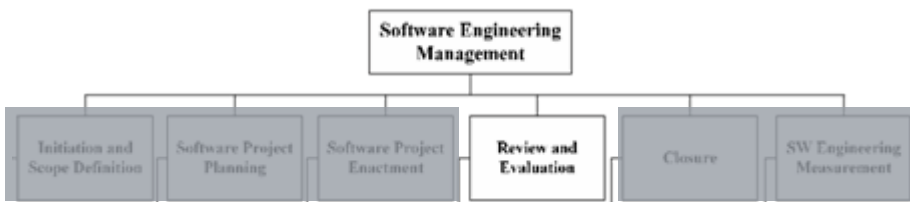
→ Supplier
Contract
Management

→ Implementation
of Measurement
Process

→ Monitor Process

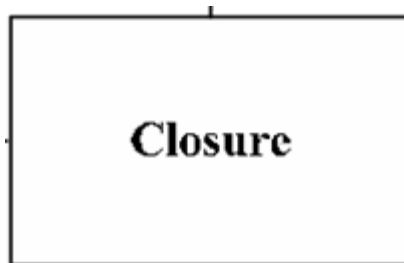
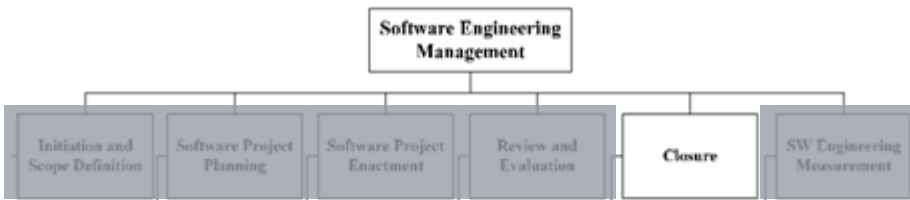
→ Control Process

→ Reporting



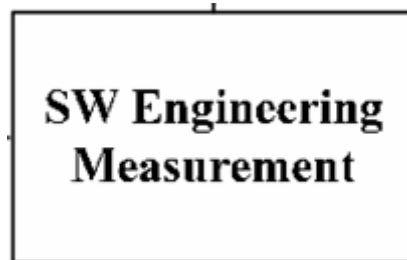
Determining
► Satisfaction of Requirements

Reviewing and
► Evaluating Performance



► Determining Closure

► Closure Activities



- Establish and
- ➔ Sustain Measurement Commitment
- Plan the
- ➔ Measurement Process
- Perform the
- ➔ Measurement Process
- ➔ Evaluate Measurement

Determination of software scope

Function

Performance

Constraints

Interfaces, and

Reliability

Understand the customers needs

Understand the business context

Understand the project boundaries

Understand the customer's motivation

Understand the likely paths for change

The Purpose of Planning

Reduce uncertainty of the future.

- All estimates are wrong, but any estimate is better than no idea at all.

To ensure that resources are being used as profitably as possible at all times.

To provide an objective measure of how well a project is progressing.

- If we have made an estimate it will be clear if we have not achieved it.

The Steps

Scoping—understand the problem and the work that must be done

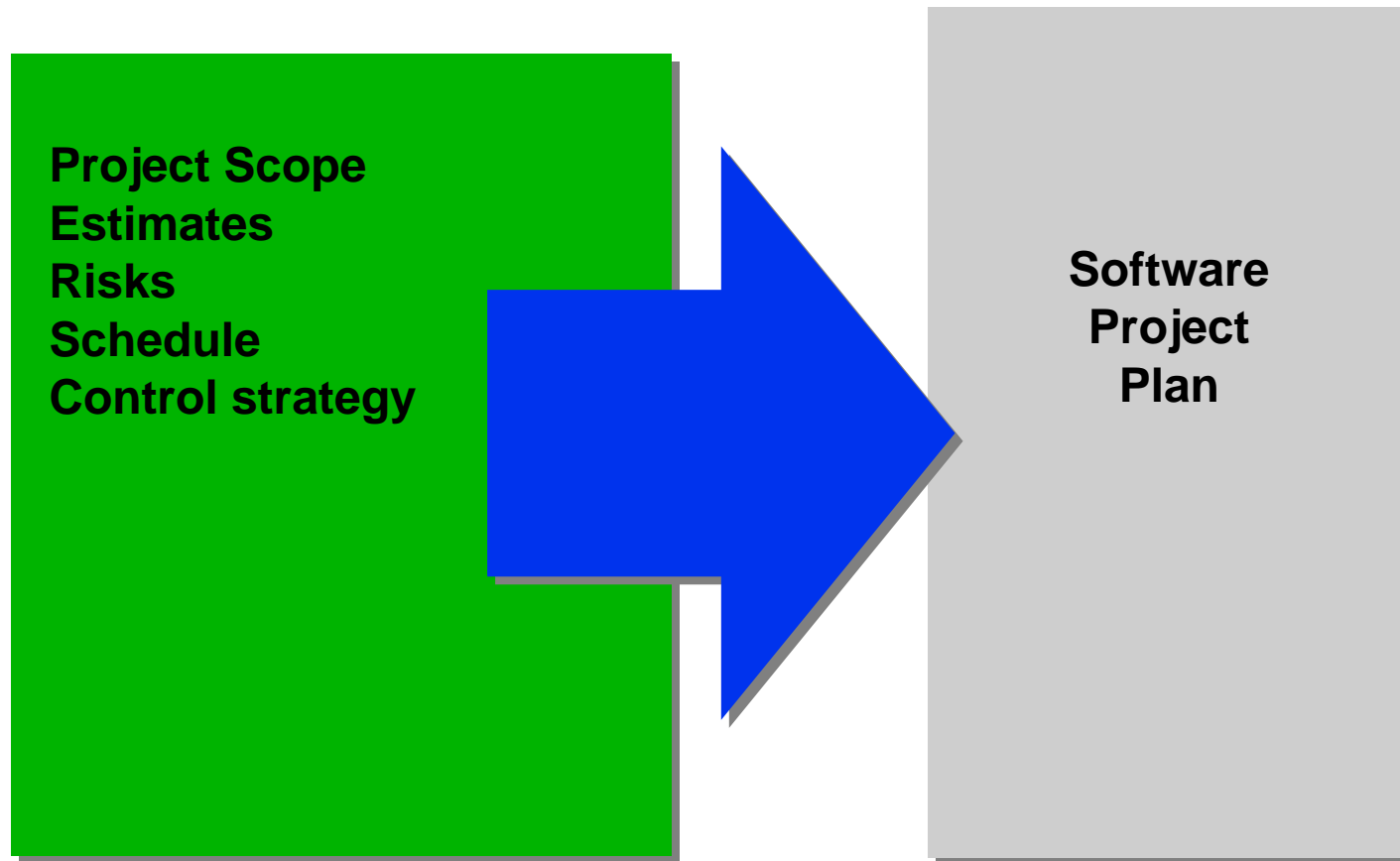
Estimation—how much effort? how much time?

Risk—what can go wrong? how can we avoid it? what can we do about it?

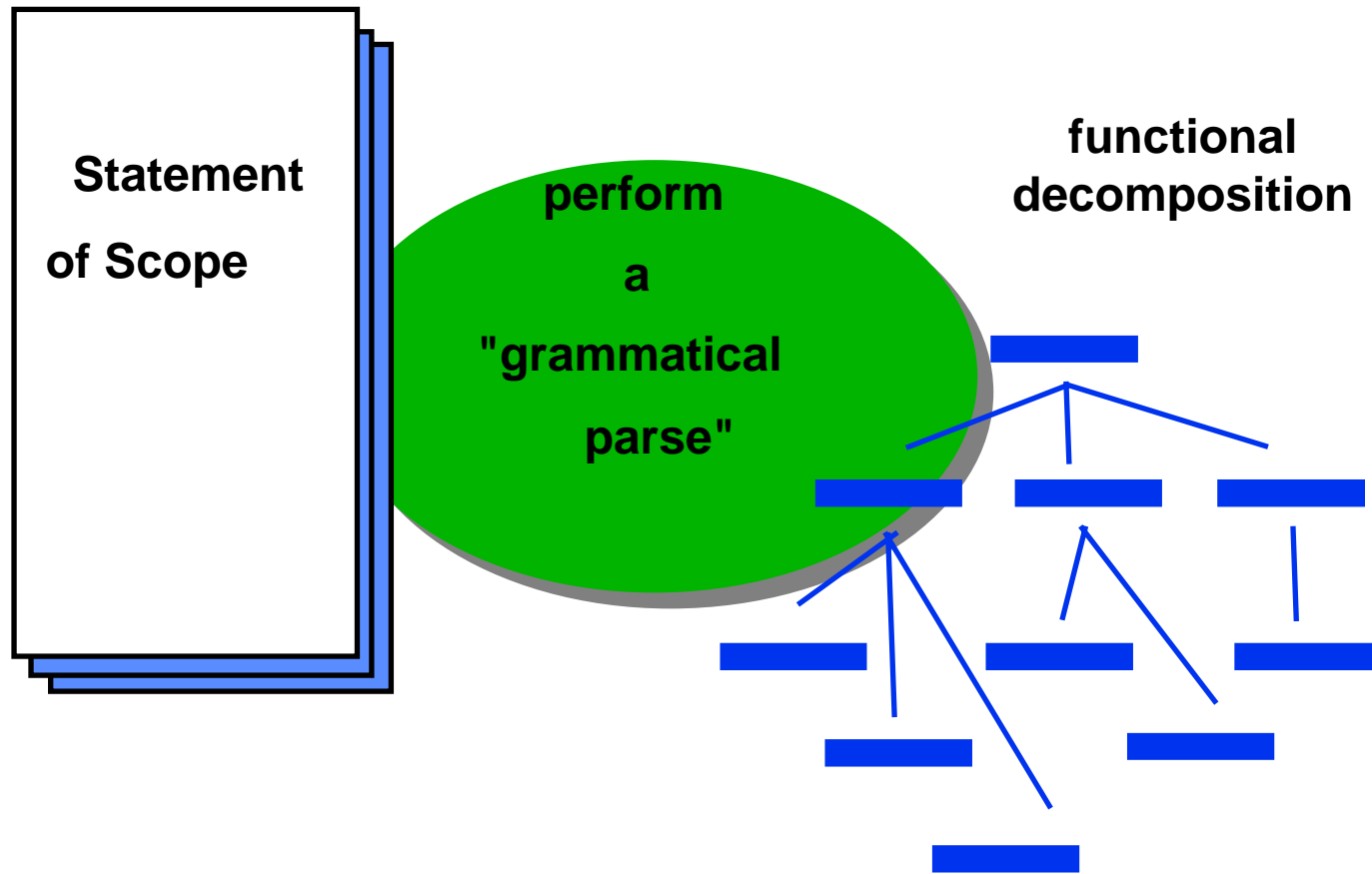
Schedule—how do we allocate resources along the timeline? what are the milestones?

Control strategy—how do we control quality? how do we control change?

Write it Down!



Functional Decomposition



Estimation of resources

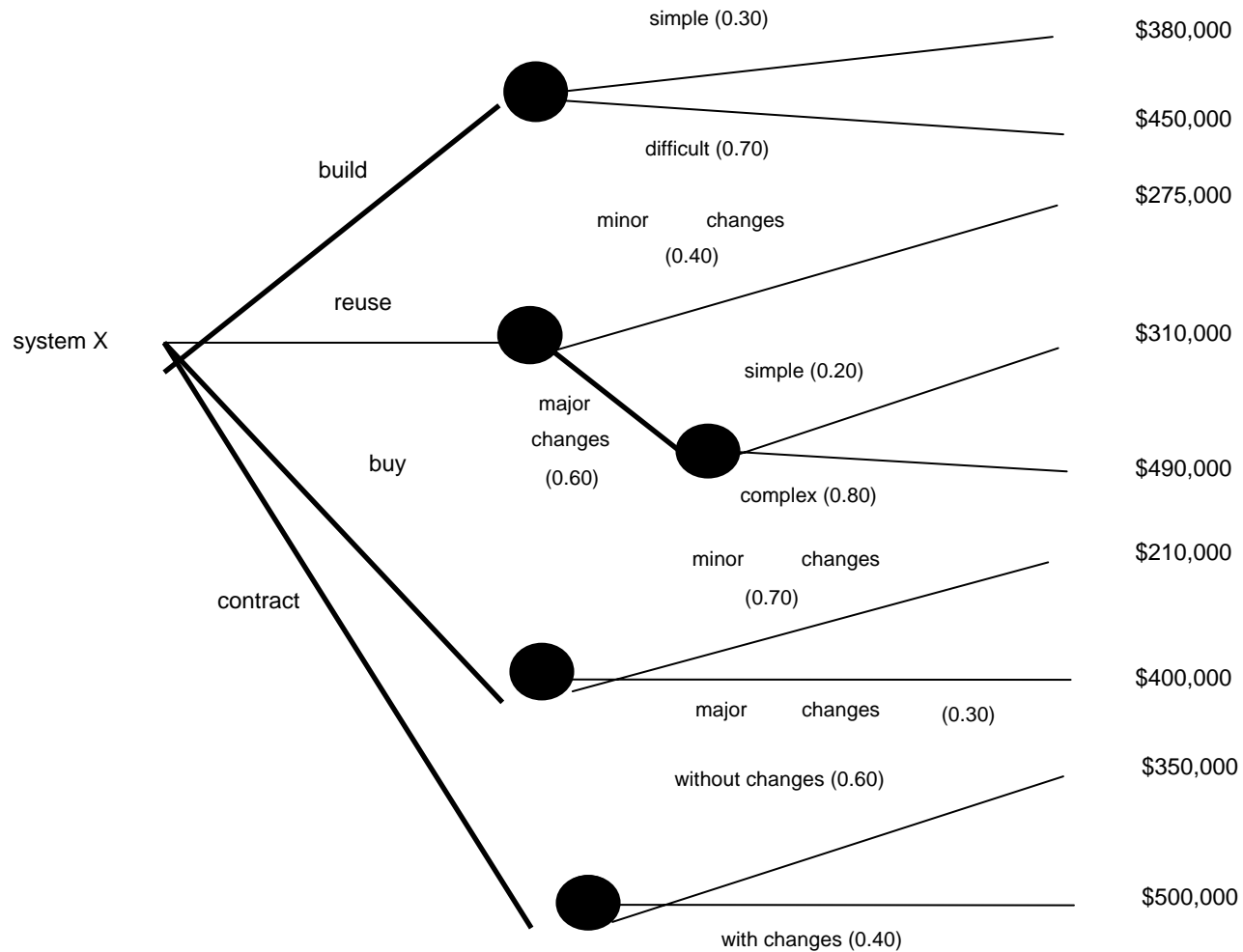
People

Reusable Software Components

- Off-the-shelf components
- Full-experience components
- Partial-experience components
- New components

Environment (Hardware/Software Tools)

The Make-Buy Decision



Computing Expected Cost

$$\text{expected cost} = \sum (\text{path probability})_i \times (\text{estimated path cost})_i$$

For example, the expected cost to build is:

$$\begin{aligned} \text{expected cost}_{\text{build}} &= 0.30(\$380\text{K}) + 0.70(\$450\text{K}) \\ &= \$429 \text{ K} \end{aligned}$$

Similarly,

$$\text{expected cost}_{\text{reuse}} = \$382\text{K}$$

$$\text{expected cost}_{\text{buy}} = \$267\text{K}$$

$$\text{expected cost}_{\text{contr}} = \$410\text{K}$$

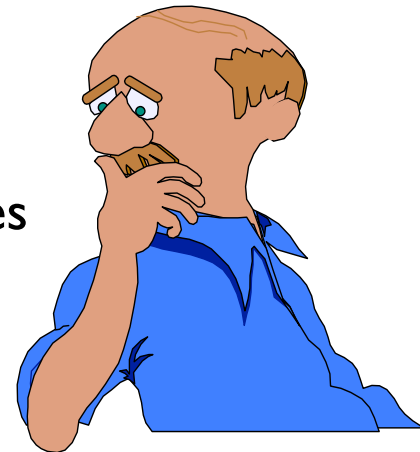
Estimation Techniques

Past (similar) project experience

Conventional estimation techniques

- Task breakdown and effort estimates
- Size (e.g., FP) estimates

Tools



Estimation: Implicit Techniques

Characteristics:

- Based on implicit relation of experience, knowledge, expectations and estimate
- Mainly based on tacit knowledge

Typical examples:

- Expert-judgement
- Wideband Delphi

Estimation: Explicit Techniques

Decomposition techniques:

- Software sizing
 - “Fuzzy-logic” sizing
 - Function point sizing
 - Standard component sizing
 - Change sizing

Estimation: Explicit Techniques

Decomposition techniques:

- Problem-based estimation
 - Estimate size by functional decomposition
 - Combine the size estimate with historical data relating size with effort and costs

Process-Based Estimation

Bases its estimate on the process that will be used

- The process is decomposed into a relatively small set of activities or tasks
- Problem functions and process activities are melded, then the planner estimates the effort that will be required to accomplish each software process activity for each software function
- Finally, costs and effort for each function and software process activity are computed

Problem-based Example: LOC

Functions	estimated LOC	LOC/pm	\$/LOC	Cost	Effort (months)
UICF	2340	315	14	32,000	7.4
2DGA	5380	220	20	107,000	24.4
3DGA	6800	220	20	136,000	30.9
DSM	3350	240	18	60,000	13.9
CGDF	4950	200	22	109,000	24.7
PCF	2140	140	28	60,000	15.2
DAM	8400	300	18	151,000	28.0
Totals	33,360			655,000	145.0

Problem-based Example : FP

<u>measurement parameter</u>	<u>count</u>	<u>weight</u>			
number of user inputs	40	x	4	=	160
number of user outputs	25	x	5	=	125
number of user inquiries	12	x	4	=	48
number of files	4	x	7	=	28
number of ext.interfaces	4	x	7	=	28
algorithms	60	x	3	=	180
count-total					569
complexity multiplier					.84
feature points					478

×

$$\boxed{0.25 \text{ p-m / FP}} = 120 \text{ p-n}$$

A Common Process Framework

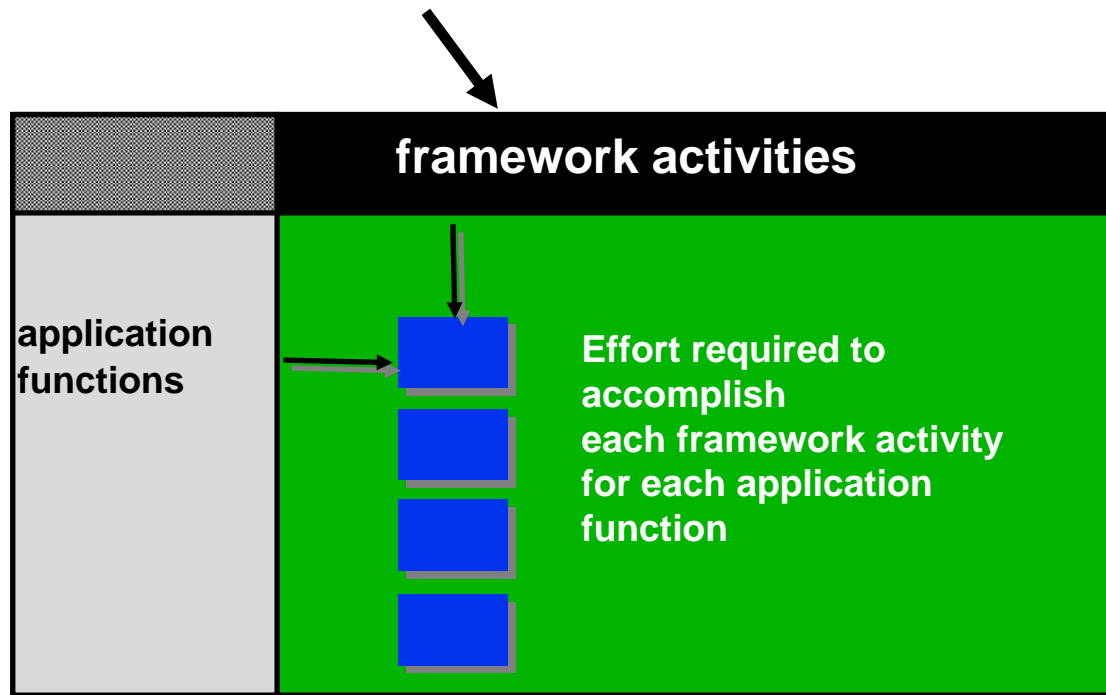
Framework activities

- work tasks
- work products
- milestones & deliverables
- QA checkpoints

Umbrella Activities

Process-based Estimation

Obtained from “process framework”



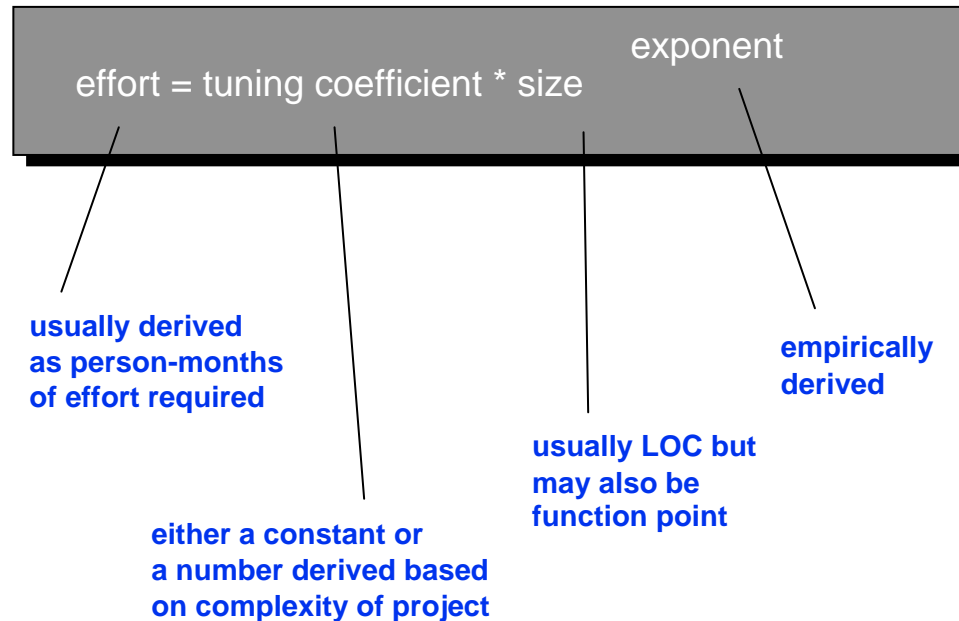
Process-Based Estimation

COMMON PROCESS FRAMEWORK ACTIVITIES	customer communication				planning				risk analysis				engineering				
Software Engineering Tasks																	
Product Functions																	
Text input																	
Editing and formating																	
Automatic copy edit																	
Page layout capability																	
Automatic indexing and TOC																	
File management																	
Document production																	

The Task Matrix

Empirical Estimation Models

General form:



Basic COCOMO 1981 Model

The Basic COCOMO equations:

- $E = ab * (KLOC)^{bb}$
- $D = cb * (E)^{db}$

E is the effort applied in person-months

D is the development time in chronological months

KLOC is the estimated number of delivered lines of code (in thousands).

Software Project	ab	bb	cb	db
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Organic: relatively small teams developing software in a highly familiar, in-house environment.

Semi-Detached: team members have some experience related to some aspects of the system under development but not others and the team is composed of experienced and inexperienced people.

Embedded: the project must operate within a strongly coupled complex of hardware, software, regulations, and operational procedures, such as real-time systems.

The COCOMO II model

Offers estimating capability at **three levels of granularity**, capturing three stages of software development activity, and providing three levels of model precision:

Prototyping: Applications Composition model, input sized in **Object Points**.

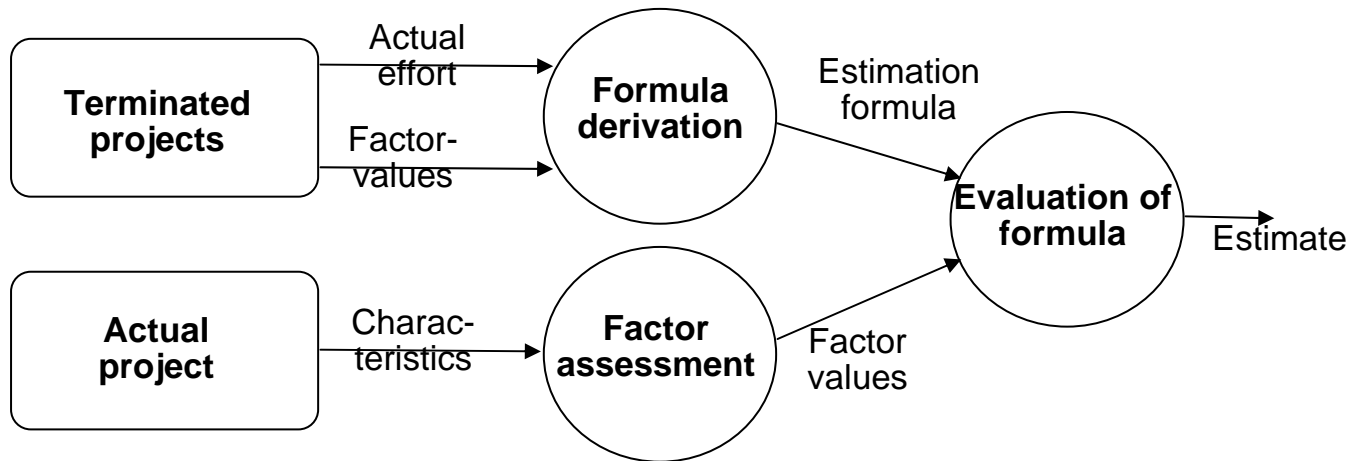
Early Design: input sized in **source statements** or **Function Points**, with **7 cost drivers**.

Post-architecture: input sized in **source statements** or **Function Points**, with **17 cost drivers**.

Five Scale Factors based upon 1) **project precededentedness**, 2) **development flexibility**, 3) **architecture/risk resolution**, 4) **team cohesion**, and 5) **development process maturity**

Multiplicative Cost Drivers applied at the *component* level.

Estimation: Empirical Models



Estimation Guidelines

- Estimate using at least two techniques
- Get estimates from independent sources
- Avoid over-optimism, assume difficulties
- You've arrived at an estimate, sleep on it
- Adjust for the people who'll be doing the job —they have the highest impact

Programmer Productivity Variations

In 1968, a study by Sackman, Erikson, and Grant revealed that programmers with the same level of experience exhibit variations of more than 20 to 1 in the time required to solve particular programming problems.

More recent studies [Curtis 1981, DeMarco and Lister 1985, Brian 1997] confirm this high variability.

Many employers in Silicon Valley argue that this productivity variance is even higher today, perhaps as much as 100 to 1.

Sackman et al's Study

TABLE III. RANGE OF INDIVIDUAL DIFFERENCES
IN PROGRAMMING PERFORMANCE

<i>Performance measure</i>	<i>Worst score</i>	<i>Best score</i>	<i>Ratio</i>
1. Debug hours Algebra	170	6	28:1
2. Debug hours Maze	26	1	26:1
3. CPU time Algebra (sec)	3075	370	8:1
4. CPU time Maze (sec)	541	50	11:1
5. Code hours Algebra	111	7	16:1
6. Code hours Maze	50	2	25:1
7. Program size Algebra	6137	1050	6:1
8. Program size Maze	3287	651	5:1
9. Run time Algebra (sec)	7.9	1.6	5:1
10. Run time Maze (sec)	8.0	.6	13:1