2. Project Planning(1)

Estimation

Fall 2022

Sungwon Kang

In this part of the course . . .

- We will mainly study
  - Project Estimation
  - Project Scheduling
  - Other remaining issues

  that are needed **to start designing and implementation**

Table of Contents

Slides copyright © 1996, 2001, 2005, 2015 by Roger S. Pressman

1. Observation on Estimation

Frederick Brooks [Bro95]:

> . . . our techniques of estimating are poorly developed. More seriously, they reflect an unvoiced assumption that is quite untrue, i.e., that all will go well. . . . because we are uncertain of our estimates, software managers often lack the courteous stubbornness to make people wait for a good product.

[Bro 95] F. Brooks, *The Mythical Man-Month*, Addison-Wesley, 1995.

2. The Project Planning Process

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

Why?

So that the end product is produced

on time,

on budget,

with the required functionality and quality!

Project Planning Task Set

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

+
project management plan
+
configuration management plan
+
software maintenance plan
(+
software quality assurance plan)

Estimation

- Estimation of resources, cost, and schedule for a software engineering effort requires

  <= Experience

  <= Access to good historical data (& metrics)

  <= The courage to commit to quantitative predictions when qualitative information is all that exists

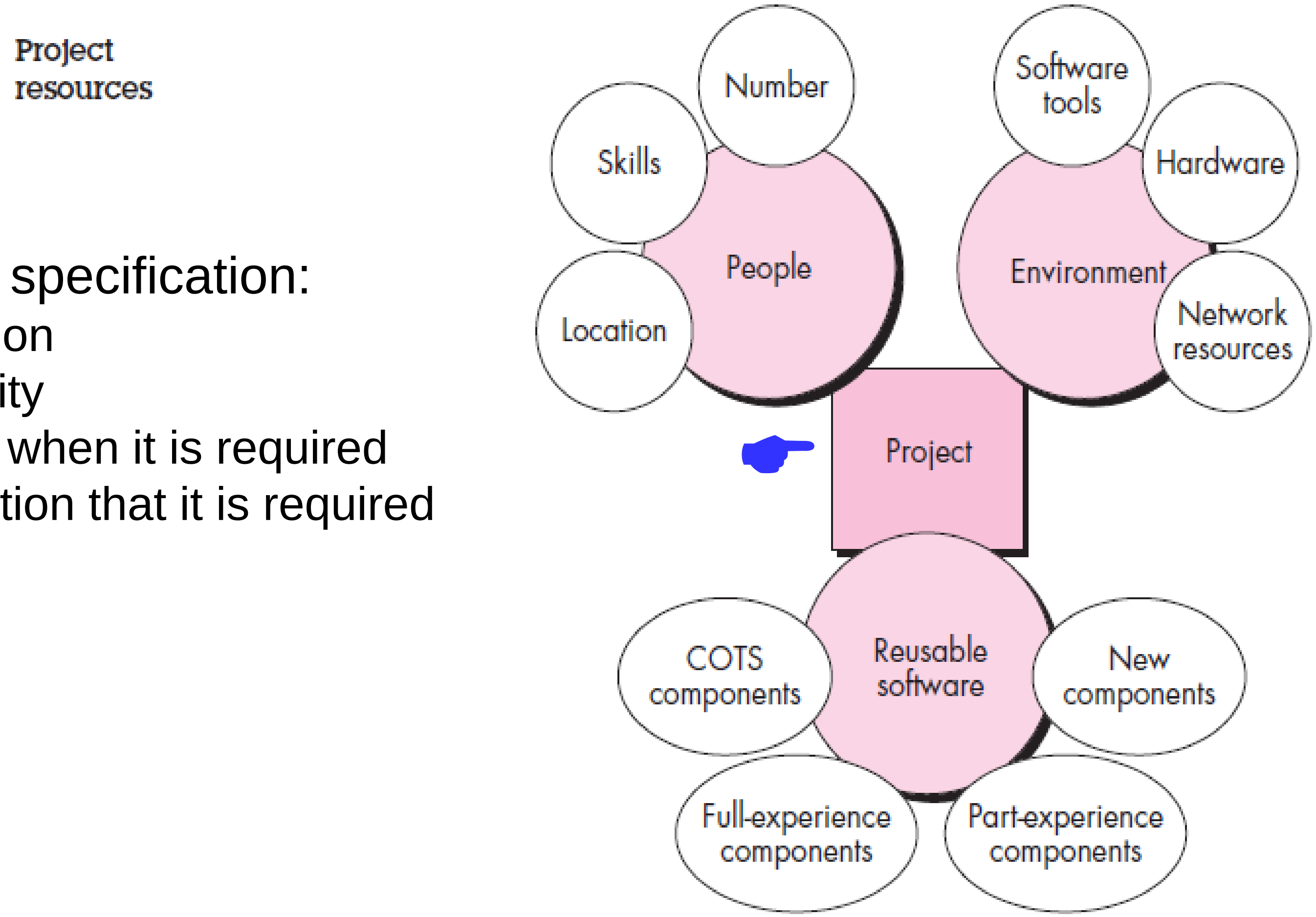
- Estimation carries inherent *risk*, which leads to uncertainty

Risk: Potential danger (as opposed to realized danger)

3. Software Scope and Feasibility

- Software scope describes
  - the functions and features 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.
- Two techniques to define scope:
    - A narrative description
    - A set of use-cases
- Feasibility
  - Can we build software to meet the scope?

4. Resources



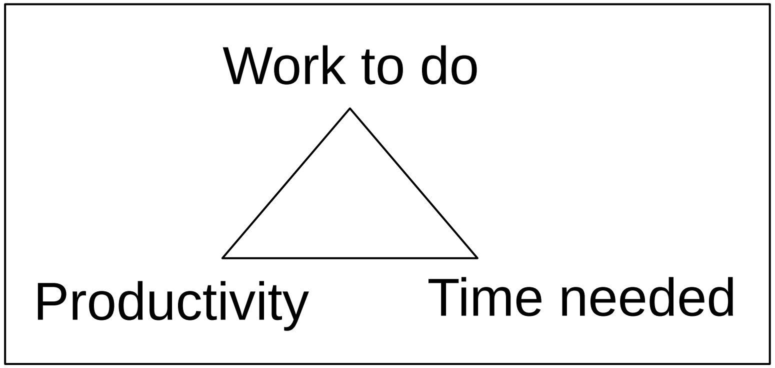
Resource specification:
1) description
2) availability
3) the time when it is required
4) the duration that it is required

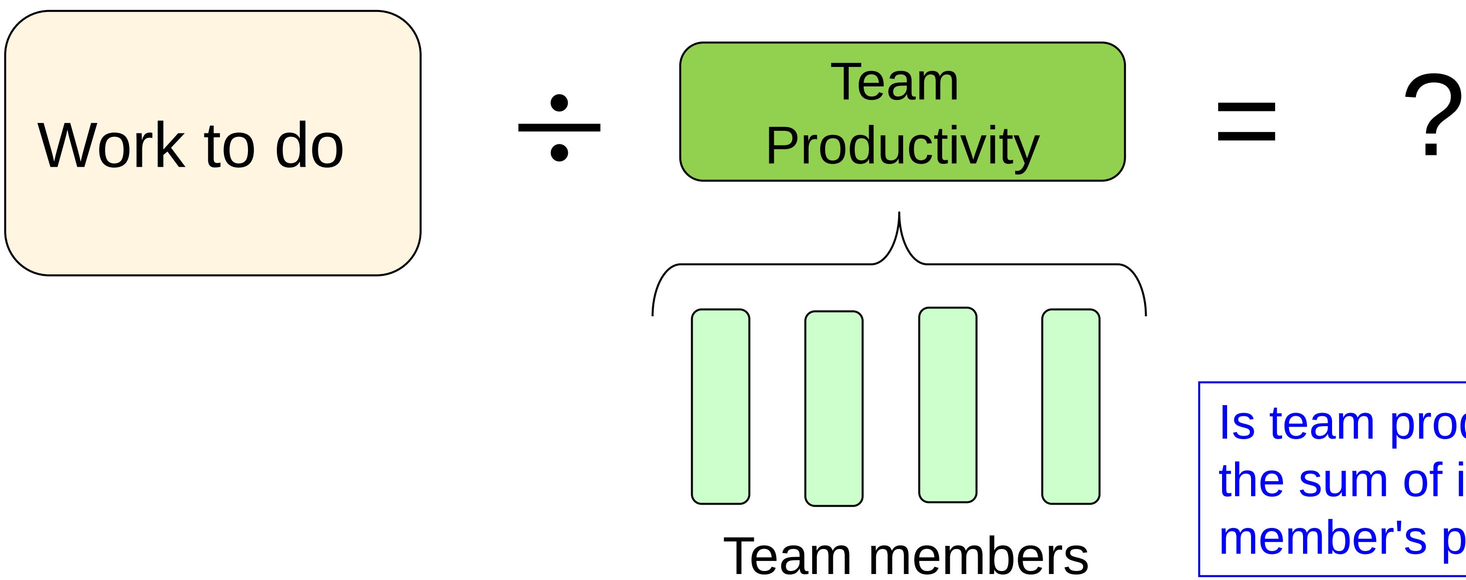
5. Software Project Estimation

- For estimation of resources, cost, and schedule
  - Project scope must be understood
  - Decomposition is necessary
  - Historical data are very helpful
  - At least two different techniques should be used

- Uncertainty is inherent in estimation

Basic Idea





Is team productivity
the sum of individual
member's productivities?

Estimation Techniques

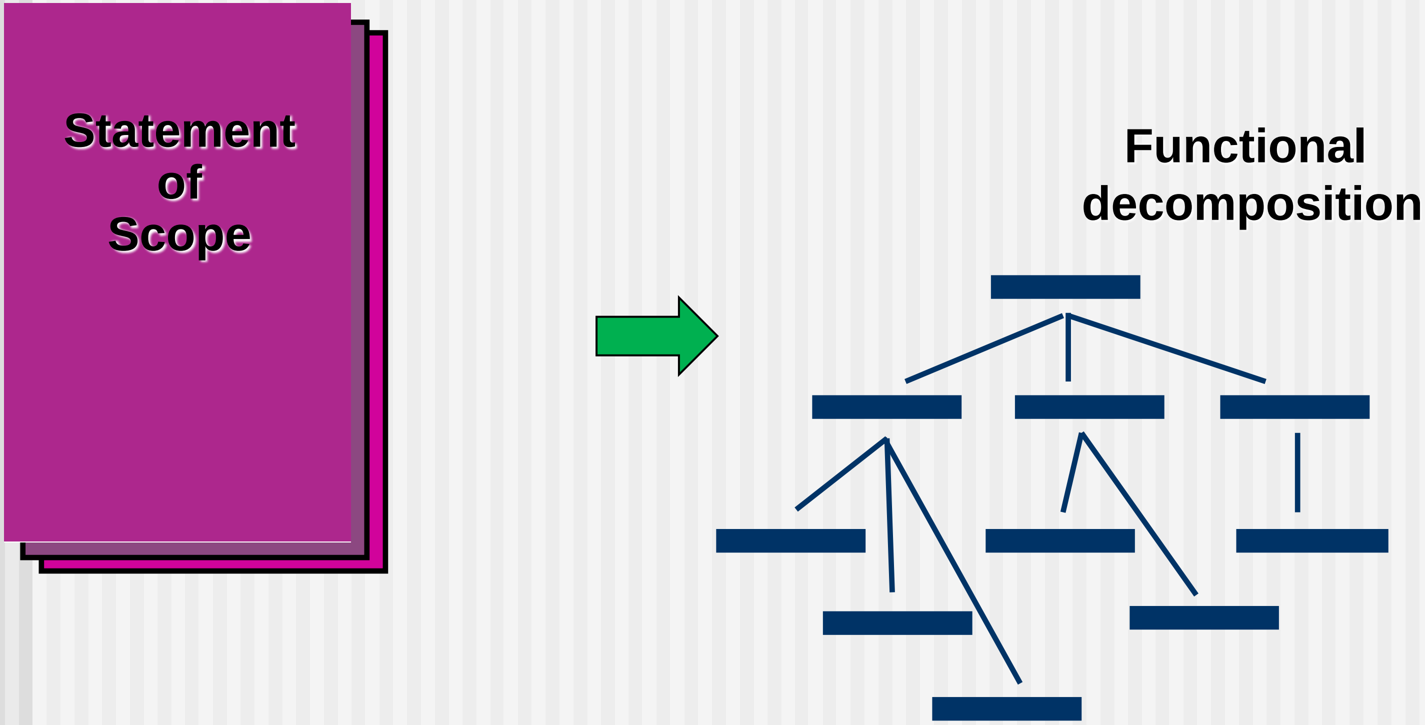
- Past (similar) project experience
- Conventional estimation techniques
  - task breakdown and effort estimates
  - size (e.g., FP) estimates
- Empirical models
- Automated tools

Estimation Accuracy

- Depends on
  - how well the planner has estimated the size of the product
  - the ability to translate the size estimate into
    - human effort,
    - calendar time, and
    - dollars

  (☛ need reliable data from past projects)
  - the abilities of the software team
  - the stability of product requirements
  - the environment that supports the development effort

6. Decomposition Techniques



(A) Problem Decomposition

- Also called problem partitioning or problem elaboration

- Once scope is defined …
  - It is decomposed into **constituent functions**
  - It is decomposed into user-visible **data objects**

  or
  - It is decomposed into a set of **problem classes**

- Decomposition continues until all functions or problem classes have been defined

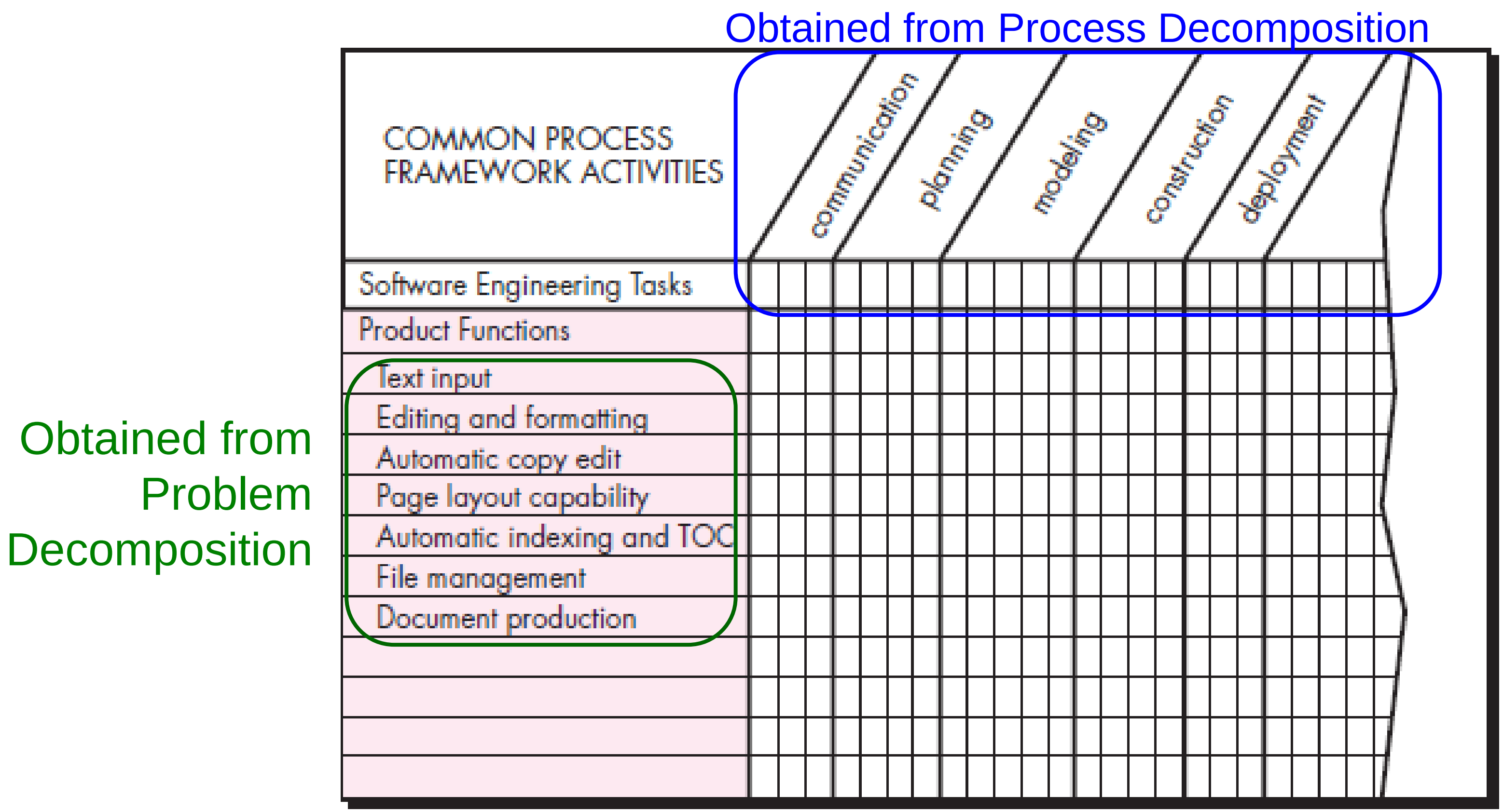
(B) Process Decomposition

Process is "a series of actions or operations conducing to an end; *especially*, a continuous operation or treatment especially in manufacture."
- Merriam-Webster Dictionary

- **Process Framework:** Core Process Activities => Generic Process Activities
  - communication (≈ requirements engineering)
  - planning
  - modeling (≈ design)
  - construction
  - deployment (on the client's site)

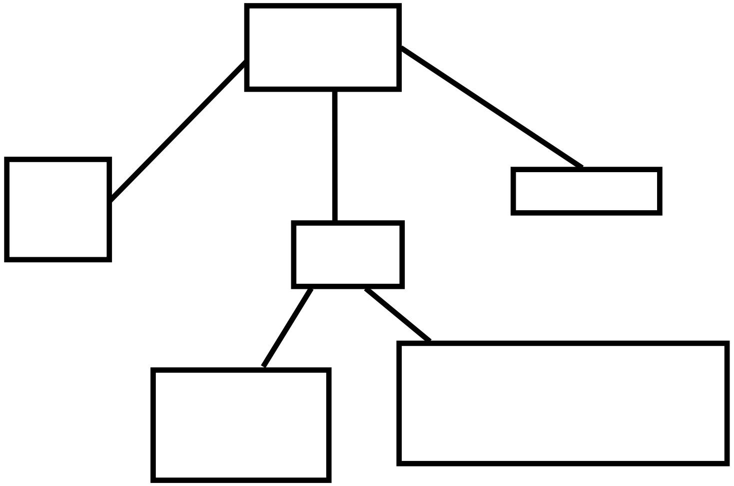
- Once a process framework has been established
  - Consider project characteristics
  - Determine the degree of rigor required
  - Define a task set for each software engineering activity
    - Task set =
      - Software engineering tasks
      - Work products
      - Quality assurance points
      - Milestones

Melding the Problem and the Process



Work Breakdown Structure (WBS)

- Purposes :
  - Identify (work) tasks
  - Identify resources
  - Provide a means of calibrating the accuracy of the development plan (How fine-grained should the plan be?)



- 1 week <= Activity duration <= 8~10 weeks

WBS - Example

1000 Project Inception

...

2000 System Definition

    2100 System Definition Management

    2200 Produce Functional Specification

        2210 Interview Customer

        2220 Model Customer Process

        2230 Produce Interface Rapid Prototype

            2231 Create Screen Formats

            2232 Create Database Interface

            2233 Review Format With Customer

            ...

        ...

    2300 Produce Project Plan

    ...

3000 System Definition

...

Tasks for individuals

7. Software Sizing

- Determining the “size” of software to be built

- Approaches:
  - **Function-based sizing**: The planner develops estimates of the information domain characteristics.
  - **Change sizing**: Used when a project encompasses the use of existing software that must be modified in some way.
  - **Standard component sizing**: Software is composed of a number of different “standard components” that are generic to a particular application area.

- Software sizing measures
  - LOC(Lines of Code) = SLOC(Source Lines of Code)
  - FP(Function Point)
  - Use Case Point
  - AP(Application Point) = OP(Object Point)

8. Function-based Cost Estimation

- Software Sizing => Software Cost Estimation

  ☛ Software Size in Unit ÷ (Cost / Software Unit) = Software Cost

- Software Cost Estimation

  1) Function-based Cost Estimation (=> Section 8)

     A) Problem-based

     B) Process-based

     C) Use Case-based

  2) Empirical Estimation Models (=> Section 9)
     (= Algorithmic Cost Estimation)

A) Problem-based Estimation

- Loc Approach & FP Approach
  - begin with a bounded statement of software scope
  - attempt to decompose it into problem functions that can each be estimated individually.
- To use LOC/FP
  - 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 function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

<u>Historical productivity data</u>

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

Burdened labor rate =$8000 per month

(= Salary + Expenses)

pm = person-month

- ☛ The cost per LOC ≈ $13
- ☛ The total estimated project cost = $431,000
- ☛ The estimated effort = 54 pm

Example: FP Approach

- Function Point (FP)
  - Proposed by [Albrecht 79]
  - An effective means for measuring the functionality delivered by a system.
  - Derived using an empirical relationship based on

    (1) countable (direct) measures of software's information domain and

    (2) assessments of software complexity

- Information domain values consists of the counts of :

  amount of work

  - External inputs (EIs)
  - External outputs (EOs)
  - External inquiries (EQs)
  - Internal logical files (ILFs)
  - External interface files (EIFs)

  countable works of
  different categories

Computing function points

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

Unadjusted FP

$$\underbrace{\text{FP}}_{\text{Adjusted FP}} = \underbrace{\text{count total}}_{\text{Unadjusted FP}} \times [0.65 + 0.01 \times \underbrace{\Sigma (F_i)}_{\text{value adjustment factors (VAF)}}]$$

Adjusted FP

Unadjusted FP

value adjustment factors (VAF)
(See the next slide)

Which is most challenging to handle?
Which is easiest to handle?

Value adjustment factors (VAF)

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

How difficult is your software development?

Each rated on scales equivalent to the following:

Not present	= 0
Incidental Influence	= 1
Moderate Influence	= 2
Average Influence	= 3
Significant Influence	= 4
Strong Influence	= 5

Example 1(1/2)

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 of **Value adjustment factor** s	$\boxed{\textbf{1.17}}$

52

Finally, the estimated number of FP is derived:

Adjusted FP

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

Example 1 (2/2)

Estimating information domain values

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

Unadjusted

$FP_{estimated}$ = $\underbrace{\text{count-total}}_{320} \times \underbrace{[0.65 + 0.01 \times \Sigma (F_i)]}_{1.17} = 375$

Adjusted

Historical productivity data

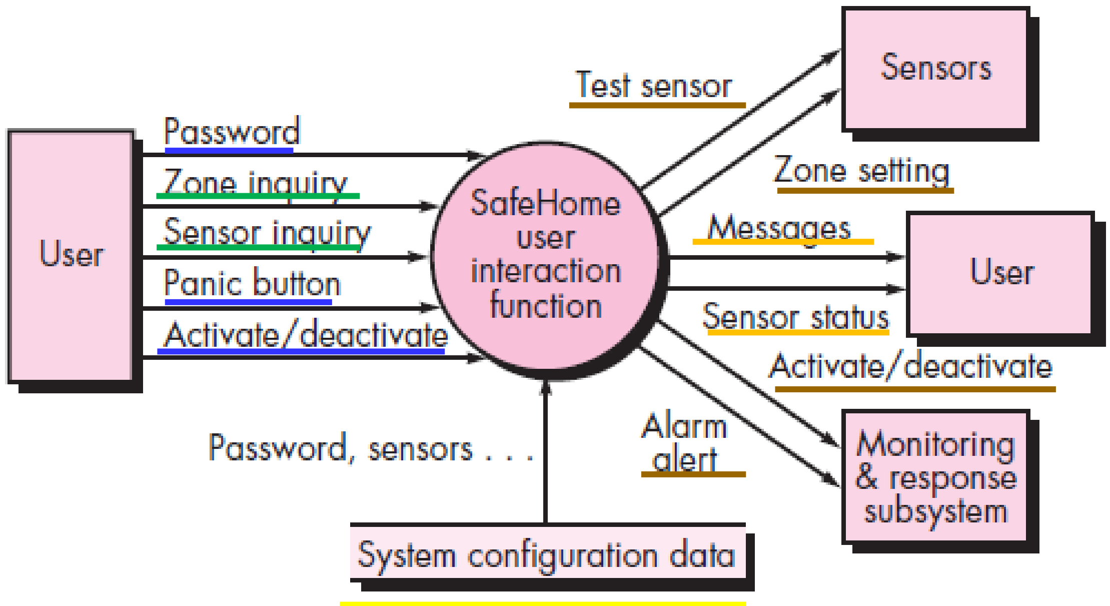
Organizational average productivity = 6.5 FP/pm.

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

- The total estimated project cost = \$461,000
- The estimated effort = 58 pm

Example 2

A data flow model for *SafeHome* software



Computing function points

Information Domain Value

External Inputs (EIs)

External Outputs (EOs)

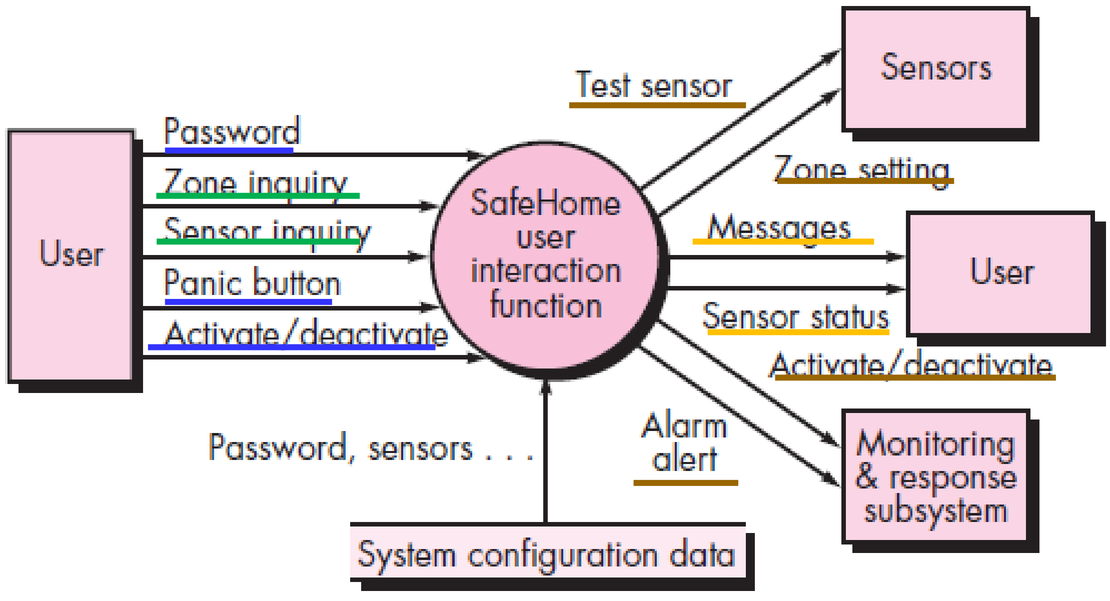
External Inquiries (EQs)

Internal Logical Files (ILFs)

External Interface Files (EIFs) <= data sent with file read/write

Example 2

A data flow model for *SafeHome* software



Computing function points

Information Domain Value	Count		Weighting factor: Simple	Average	Complex		
External Inputs (EIs)	3	×	3	4	6	=	9
External Outputs (EOs)	2	×	4	5	7	=	8
External Inquiries (EQs)	2	×	3	4	6	=	6
Internal Logical Files (ILFs)	1	×	7	10	15	=	7
External Interface Files (EIFs)	4	×	5	7	10	=	20
Count total							50

Assume $\Sigma(F_i)$ is 46

Then $FP = 50 \times [0.65 + (0.01 \times 46)] = 56$

1.11

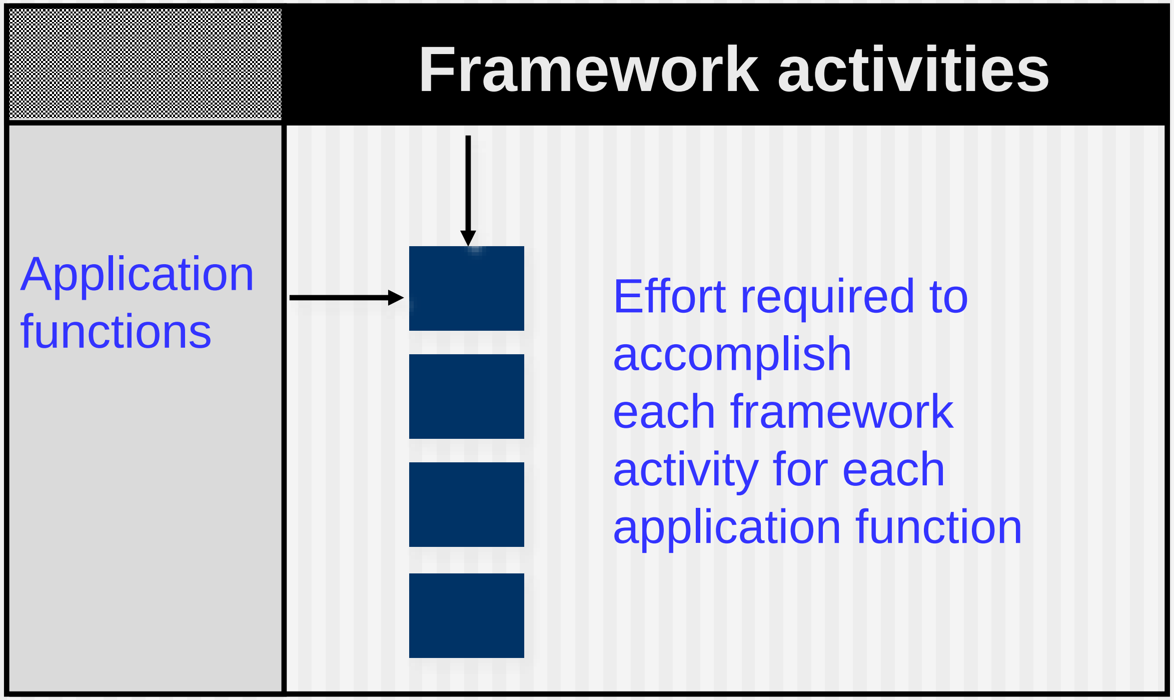
Converting FPs to LOC

- FPs can be used to estimate LOC depending on the Average number of LOC per FP for a given language
  - LOC = Average number of LOC/FP $\times$ number of FP
  - Average number of LOC per FP in reality:
    - 200 ~ 300 for assemble languages
    - . . . for 3GLs (3rd Generation Languages)
    - 2 ~ 40 for 4GLs (4th Generation Languages)

How many debugged LOC can you write per day?

B) Process-Based Estimation

Recall the matrix obtained from
Problem Decomposition and Process Decomposition



Actually a problem-based & process-base estimation !

How are these numbers obtained?

Example: Process-Based Estimation

Process-based estimation table
(In Person-Month)

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

<u>Historical productivity data</u>
The average burdened labor rate = $8,000 per month

- The estimated effort = 46 pm
- The total estimated project cost = $368,000

C) Estimation with Use-Cases

UCP = (UUCW + UAW) × TCF × ECF

UCP: Use Case Point
UUCW: Unadjusted Use Case Weight
UAW: Unadjusted Actor Weight
TCF: Technical Complexity Factor
ECF: Environmental Complexity Factor

Use Case Complexity	Factor
Simple	5
Average	10
Complex	15

Actor Complexity	Factor
Simple	1
Average	2
Complex	3

What would be a problem with the use case based estimation?

Example: Use-Case-Based Estimation

Function
User interface and control facilities (UICF)
Two-dimensional geometric analysis (2DGA)
Three-dimensional geometric analysis (3DGA)
Database management (DBM)
Computer graphics display facilities (CGDF)
Peripheral control function (PCF)
Design analysis modules (DAM)
Estimated lines of code

Subsystem	# Use Cases	Complexity
UICF	16	15
2DGA, 3DGA	14	10
DAM	8	5
CGDF, PCF	10	5

Actor Complexity	#Actors	Factor
Simple	8	1
Average	12	2
Complex	4	3

Assume: 8 simple, 12 average, 4 complex actors

TCF = 1.04

ECF = 0.96

UUCW = $16 \times 15 + [14 \times 10 + 8 \times 5] + 10 \times 5 = 470$

UAW = $8 \times 1 + 12 \times 2 + 4 \times 3 = 44$

UCP = (UUCW + UAW) $\times$ TCF $\times$ ECF = $(470 + 44) \times 1.04 \times 0.96 = \mathbf{513}$

UUCW: Unadjusted Use Case Weight

UAW: Unadjusted Actor Weight

TCF: Technical Complexity Factor

ECF: Environmental Complexity Factor

Converting Use-Case Points to LOC (Pressman)

CAD system

	use cases		scenarios	pages	scenarios	pages	Average LOC		LOC estimate
User interface subsystem	6	x	10	6	12	5	560	=	3,366
Engineering subsystem group	10	x	20	8	16	8	3100	=	31,233
Infrastructure subsystem group	5	x	6	5	10	6	1650	=	7,970
Total LOC estimate									42,568

Historical Data

Using the relationship in the next slide with $n = 30\%$, the table above is obtained.

$$\text{LOC estimate} = \boxed{N \times \text{LOC}_{avg}} + \boxed{[(S_a/S_h - 1) + (P_a/P_h - 1)] \times \text{LOC}_{adjust}}$$

Example User interface subsystem

$$[(10/12 - 1) + (6/5 - 1)] = (0.83 - 1) + (1.2 - 1) = 0.033$$

$$0.033 \times \text{LOC}_{adjust} = 0.033 \times 560 \times 30\% = 5.6 \approx 6$$

$$N \times \text{LOC}_{avg} = 6 \times 560 = 3,360$$

Converting Use-Case Points to LOC (Pressman)

$$\text{LOC estimate} = N \times \text{LOC}_{avg} + [(S_a/S_h - 1) + (P_a/P_h - 1)] \times \text{LOC}_{adjust}$$

N	= actual number of use cases
LOC_{avg}	= historical average LOC per use case for this type of subsystem
LOC_{adjust}	= represents an adjustment based on n percent of LOC_{avg} where n is defined locally and represents the difference between this project and "average" projects
S_a	= actual scenarios per use case
S_h	= average scenarios per use case for this type of subsystem
P_a	= actual pages per use case
P_h	= average pages per use case for this type of subsystem

- Like other metrics, need to be validated "locally" !

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill 2014). Slides copyright 2014 by Roger Pressman.

Converting Use-Case Points to LOC (Pressman)

	use cases	scenarios	pages	scenarios	pages	LOC	LOC estimate
User interface subsystem	6	10	6	12	5	560	3,366
Engineering subsystem group	10	20	8	16	8	3100	31,233
Infrastructure subsystem group	5	6	5	10	6	1650	7,970
Total LOC estimate							42,568

Historical productivity data

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

Burdened labor rate =$8000 per month

The cost per LOC ≈ $13

- Total estimated project cost = $552,000
- Estimated effort = 68 pm

Reconciling Estimates

- The estimation techniques result in multiple estimates that must be reconciled to produce a single estimate of effort, project duration, or cost.

Example

The total estimated effort for the CAD software ranges from

(1) a low of 46 pm (derived using a process-based estimation approach) to
(2) a high of 68 pm (derived with use-case estimation).
(3) The average estimate (using all 4 approaches: LOC, FP, Process-based, Use case-based) is 56 pm.

The variation from the average estimate is approximately 18% on the low side and 21% on the high side.

☛ May have to reevaluate information used for estimation

☛ Your way of reconciliation should be “reproducible” ! Why?

9. Empirical Estimation Models

(≈ Algorithmic Cost Estimation)

9.1 The Structure of Estimation Models

General form

Empirically derived value for complexity

$$PM = A \times Size^{B} \times M$$

Effort in person-month

Multiplier: Cost drivers

Usually LOC but may also be FP

Tuning coefficient: either a constant or a number derived based on complexity of project

9.2 The COCOMO II Model

- COCOMO II is actually a hierarchy of the following estimation models:
  - Application composition model. Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
  - 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.

You should measure not just once but multiple times!

The COCOMO Model

B. Boehm (1935 ~2022)

- COCOMO (= COnstructive COst Model):
  - An empirical model based on project experience
    => Can be viewed as a *meta-model* since each user has to define parameter values before using it
  - Well-documented ‘independent’ model which is not tied to a specific software vendor
  - Has a long history from COCOMO-81(1981) through various instantiations to COCOMO II (1995)

- COCOMO-81 (based on 61 projects) assumes
  (1) Waterfall process
  (2) Software developed from scratch

- COCOMO II (based on 163 projects) takes into account different approaches to software development, reuse, etc.
  http://softwarecost.org/tools/COCOMO/

The Basic COCOMO 81 Model

Project Complexity	Formula	Description
Simple	$PM = 2.4 \times (KDSI)^{1.05} \times M$	Well-understood applications developed by small teams
Moderate	$PM = 3.0 \times (KDSI)^{1.12} \times M$	More Complex projects where team members may have limited experience of related systems
Embedded	$PM = 3.6 \times (KDSI)^{1.20} \times M$	Complex projects where the software is part of a strongly coupled complex of hardware, software, regulations and operational procedures

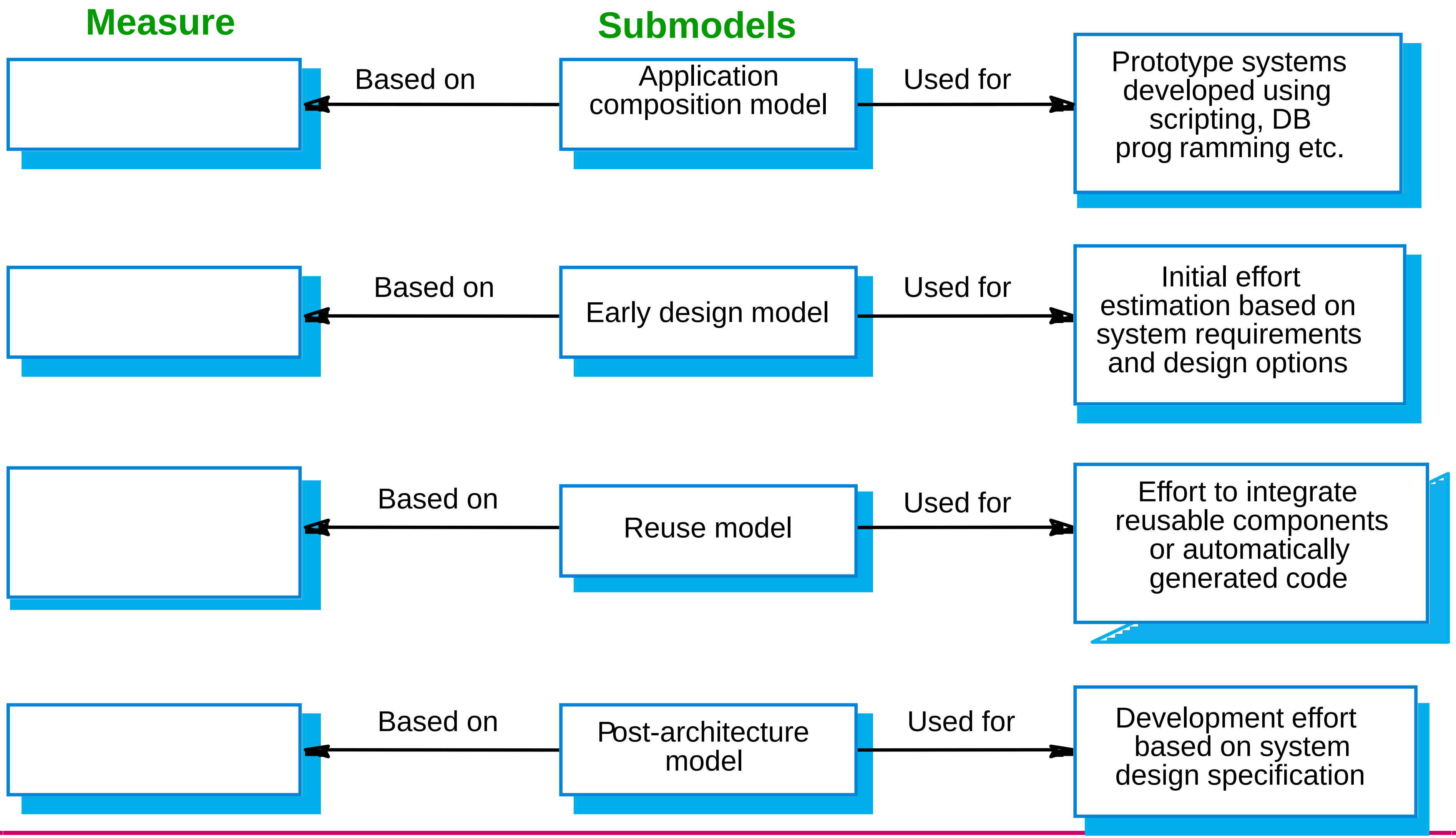
KDSI = K (Thousand) Delivered Source Instructions

COCOMO II

- COCOMO II incorporates a range of sub-models that produce increasingly detailed software estimates.

- The sub-models in COCOMO II are:
  1) **Application composition model**: Used when software is composed from existing parts.
  2) **Early design model**: Used when requirements are available but design has not yet started.
  3) **Reuse model**: Used to compute the effort of integrating reusable components.
  4) **Post-architecture model**: Used once the system architecture has been designed and more information about the system is available.

Use of COCOMO II Models



Use of COCOMO II Models

Measure		Submodels		
Number of **application points**	← Based on	Application composition model	Used for →	Prototype systems developed using scripting, DB prog ramming etc.
Number of **function points**	← Based on	Early design model	Used for →	Initial effort estimation based on system requirements and design options
Number of **lines of code reused or generated**	← Based on	Reuse model	Used for →	Effort to integrate reusable components or automatically generated code
Number of **lines of source code**	← Based on	Post-architecture model	Used for →	Development effort based on system design specification

(1) Application Composition Model

- Supports prototyping projects and projects where there is extensive reuse.

PM = (#AP × (1 – %Reuse/100)) / PROD

#AP: Number of application points

%Reuse: Percentage of reuse

PROD: Productivity

☛ Produces an approximate estimate that does not take into account the additional effort involved in reuse.

Application Points (AP)

- Also known as *Object Points* (OP)

- An alternative function-related measure to FP when 4GLs are used

- The number of APs in a program is a weighted estimate of
  - the number of separate screens that are displayed
  - the number of reports that are produced by the system
  - the number of 3GL modules that must be developed to supplement the 4GL code

AP Estimation

- APs are easier to estimate from a specification than FPs as they are simply concerned with screens, reports and 3GL modules
  => Can be estimated at an early point in the development process.

- Based on standard estimates of developer productivity in APs / month.

AP productivity

Developer's experience and capability	very low	low	nominal	high	very high
ICASE maturity and capability	very low	low	nominal	high	very high
PROD(= #AP / month)	4	7	13	25	50

ICASE: Integrated Computer Aided Software Engineering

(2) Early Design Model

- Estimates can be made after the requirements have been agreed.

$$PM = A \times Size^{B} \times M$$

where

$M = PERS \times RCPX \times RUSE \times PDIF \times PREX \times FCIL \times SCED$

(See the next slide for multiplier definitions)

A = 2.94 in initial calibration,

Size in KLOC,

B varies from 1.1 to 1.24 depending on
novelty of the project,
development flexibility,
risk management approaches and
the process maturity.

Multipliers – M

- **Cost drivers - 4 Categories 17 Multipliers**
  1. Product attributes
     - Concerned with required characteristics of the software product being developed.
  2. Computer attributes
     - Constraints imposed on the software by the hardware platform.
  3. Personnel attributes
     - Multipliers that take the experience and capabilities of the people working on the project into account.
  4. Project attributes
     - Concerned with the particular characteristics of the software development project.

Multipliers – M

- Reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.
  - PERS: personnel capability
  - RCPX: product reliability and complexity
  - RUSE: the reuse required
  - PDIF: platform difficulty
  - PREX: personnel experience
  - FCIL: the team support facilities
  - SCED: required schedule

The value of each cost factor takes the value between 0.5 and 1.5.

We know that typically $1.05 \leq B \leq 1.20$. (See next slide.)

Influence of Cost Drivers – M

Exponent value (= B)	1.17
System size (including factors for reuse and requirements volatility)	128, 000 DSI
Initial COCOMO estimate without cost drivers	**730 person-months**
Reliability	Very high, multiplier = 1.39
Complexity	Very high, multiplier = 1.3
Memory constraint	High, multiplier = 1.21
Tool use	Low, multiplier = 1.12
Schedule	Accelerated, multiplier = 1.29
Adjusted COCOMO estimate	**2306 person-months**
Reliability	Very low, multiplier = 0.75
Complexity	Very low, multiplier = 0.75
Memory constraint	None, multiplier = 1
Tool use	Very high, multiplier = 0.72
Schedule	Normal, multiplier = 1
Adjusted COCOMO estimate	**295 person-months**

A = 2.5 assumed

$$PM = 2.5 \times \boxed{128^{1.17}} \times M$$

= 292

Maximum M = 31.59

Maximum values

Minimum M = 0.405

Minimum values

Only 5 key cost factors used here. Other factors set to 1.

(3) The Reuse Model

- Takes into account
  - code reused without change
  - code to be adapted to integrate the reused code with new code
- Reuse is either automatic translation or modification

  (3.1) Black-box reuse:

  - Code is not modified but is automatically generated.

  (3.2) White-box reuse:

  - Code is modified.

(3.1) Black-box Reuse Model

- Integration effort for automatically generated code:

  $$PM_{AUTO} = (ASLOC \times AT/100) / ATPROD$$

  ASLOC: Total number of lines of reused code (including automatically generated code)

  AT: Percentage of reused code that is automatically generated.

  ATPROD: Productivity of engineers in integrating this code

☛ Auto generated code needs manual integration

ASLOC x AT/100 => Lines of reused code

Example

Typically ATPROD = 2,400 SLOC / pm

If black-box reused code = 20,000 SLOC, automatically generated code = 30%, then what is the effort to integrate this generated code?

(20,000 x 30 /100) / 2400 = 2.5 pm

(3.2) White-box Reuse Model Estimates

- When code has to be understood and integrated:
  - ESLOC = ASLOC * (1 - AT/100) * AAM

    ASLOC: Total number of lines of reused code

    AT: Percentage of code automatically generated.

    AAM(Adaptation Adjustment Multiplier):

    - Computed from the *costs of understanding and changing the reused code*
    - Consists of:
      1) Assessment cost: whether to adapt or not
      2) Understanding cost:

         (5 times cost to understand unstructured code)
      3) Adaptation cost

    ESLOC: Equivalent SLOC

(4) Post-Architecture Model

- Uses the same formula as the early design model

$$PM = A * Size^{B} * M$$

but by determining

B more precisely (see next slide) and

M using 17 associated multipliers rather than 7.

- The code size is estimated by adding the following:
  1) Number of lines of new code to be developed(SLOC)
  2) Estimate of equivalent number of lines of new code computed using the reuse model(ESLOC)
  3) Estimate of the number of lines of code that have to be modified according to requirements changes

☛ Architecture design determines which components are to be newly developed and which components are to be reused (with or without modification)

Exponent Term - B

- This depends on 5 scale factors.

  Each factor ranges from 0 to 5. (0: extra high, 5: very low)

  Their sum/100 is added to 1.01.

With lower capability, more effort is needed.
With higher capability, less effort is needed.

Example A company takes on a project in a new domain.

The client has not defined the process to be used and has not allowed time for risk analysis.

The company has a CMM(Capability Maturity Model) Level 2 rating. Then

1) **Precedentedness**: E.g. new project (4)
2) **Development flexibility**: E.g. no client involvement - Very high (1)
3) **Architecture/risk resolution**: E.g. No risk analysis - Very Low (5)
4) **Team cohesion**: E.g. New team – nominal (3)
5) **Process maturity**: E.g. Some control – nominal (3)

=> Scale factor = 1.17 = 1.01 + (4 + 1 + 5 + 3 + 3) /100

http://softwarecost.org/tools/COCOMO/

Monte Carlo Risk: Off
Auto Calculate: Off

COCOMO II - Constructive Cost Model

Software Size

Sizing Method: Source Lines of Code

	SLOC	% Design Modified	% Code Modified	% Integration Required	Assessment and Assimilation (0% - 8%)	Software Understanding (0% - 50%)	Unfamiliarity (0-1)
New							
Reused		0	0				
Modified							

Software Scale Drivers **(5 scale factors)**

Precedentedness	Nominal	Architecture / Risk Resolution	Nominal	Process Maturity	Nominal
Development Flexibility	Nominal	Team Cohesion	Nominal		

Software Cost Drivers **(17 multipliers)**

Product		Personnel		Platform	
Required Software Reliability	Nominal	Analyst Capability	Nominal	Time Constraint	Nominal
Data Base Size	Nominal	Programmer Capability	Nominal	Storage Constraint	Nominal
Product Complexity	Nominal	Personnel Continuity	Nominal	Platform Volatility	Nominal
Developed for Reusability	Nominal	Application Experience	Nominal	**Project**	
Documentation Match to Lifecycle Needs	Nominal	Platform Experience	Nominal	Use of Software Tools	Nominal
		Language and Toolset Experience	Nominal	Multisite Development	Nominal
				Required Development Schedule	Nominal

Maintenance Off

Software Labor Rates

Cost per Person-Month (Dollars)

Calculate

When you do not have historical data . . .

- The following facts were discovered in the 19th century.

- **Game**: One who estimated the weight of an ox most closely wins a prize. Several hundreds people made guesses.
  Fact 1. The ‘middlemost’ estimate, known as the Median, best reflected the actual weight of the ox.
  Fact 2. The mean value produced an even more accurate estimate than any of the individual estimates.

☛ **Wisdom of Crowds (= Collective Intelligence)**

Wideband Delphi estimation method (1/2)

- A consensus-based technique for estimating
- B. Boehm et al. derived it (1970s) from the Delphi method (1950-1960s, RAND Co.)
- Adapted across many industries to estimate from statistical data collection results to sales and marketing forecasts.
- “Wideband" because it involves greater interaction and more communication among the participants

Delpi: an ancient Greek holy place where an oracle (= a female priest) was believed to be able to answer questions with advice from Apollo

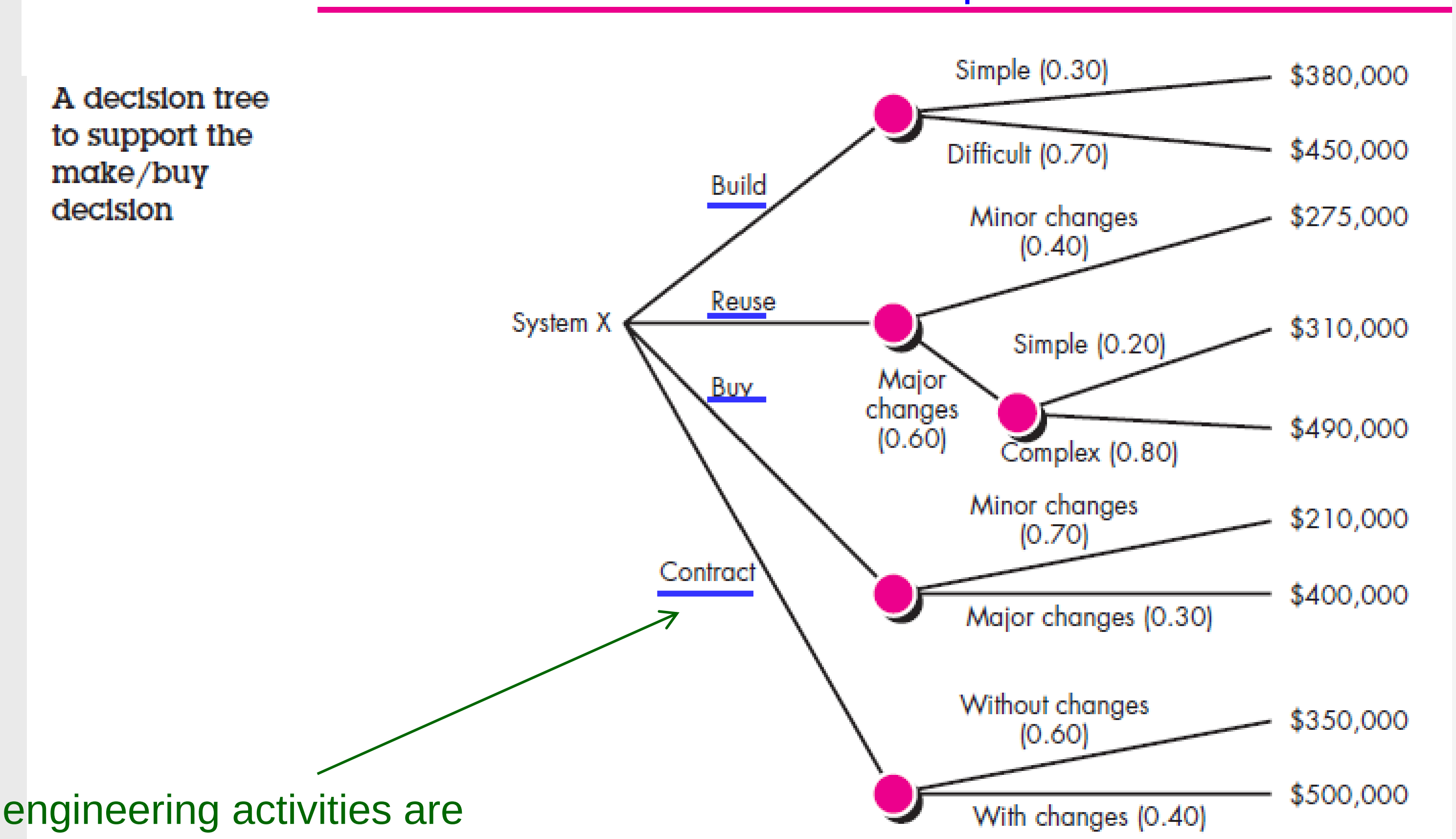
Wideband Delphi estimation method (2/2)

- Steps (Boehm, *Software Engineering Economics,* 1981).
  1. **Coordinator** presents each expert with a specification and an estimation form.
  2. **Coordinator** calls a group meeting in which the experts discuss estimation issues with the coordinator and each other.
  3. **Experts** fill out forms anonymously.
  4. **Coordinator** prepares and distributes a summary of the estimates
  5. **Coordinator** calls a group meeting, specifically focusing on having the experts discuss points where their estimates vary widely
  6. **Experts** fill out forms, again anonymously, and steps 4 to 6 are iterated for as many rounds as appropriate.

You can use a simplified version of this technique for your project !
The essence of this can be found in the Ox weight guessing game.

10. The Make-Buy Decision

probabilities



Software engineering activities are contracted to a third party who does the work at lower cost and, hopefully, higher quality.

Computing Expected Cost

Expected cost =

$$\sum \left(\text{Path\_probability}_i \times \text{Estimated\_path\_cost}_i\right)$$

For example, the expected cost to build is:

$$\textbf{Expected cost}_{\text{build}} = 0.30 \times \$380K + 0.70 \times \$450K$$

$$= \$429\text{ K}$$

Similarly,

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

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

$\text{Expected cost}_{\text{contract}} = \$410K$

This shows the idea of decision making. But to be realistic maintenance and evolution costs should be included.

Questions?