# Chapter 33

- ## **Estimation for Software Projects**

*Slide Set to accompany*

*Software Engineering: A Practitioner's Approach*

**by Roger S. Pressman**

# In this course …

- Learning is more important than the final product.
  - Your product will be probably thrown away after evaluation but you take knowledge and lessons learned to the real world.
- Real world does not suffer from lack of knowledge but from the inability to cope with its unique situation.
- For detailed knowledge, you can look up books and papers.
- For the ability to cope with the unique situation, experience and "sound" creativity as opposed to wild creativity is important !
  - **Creativity** rather than experience helps find solution
  - **Experience** helps avoid dangers.

# In the following classes

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

# Table of Contents

# 33.1 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.

# 33.2 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

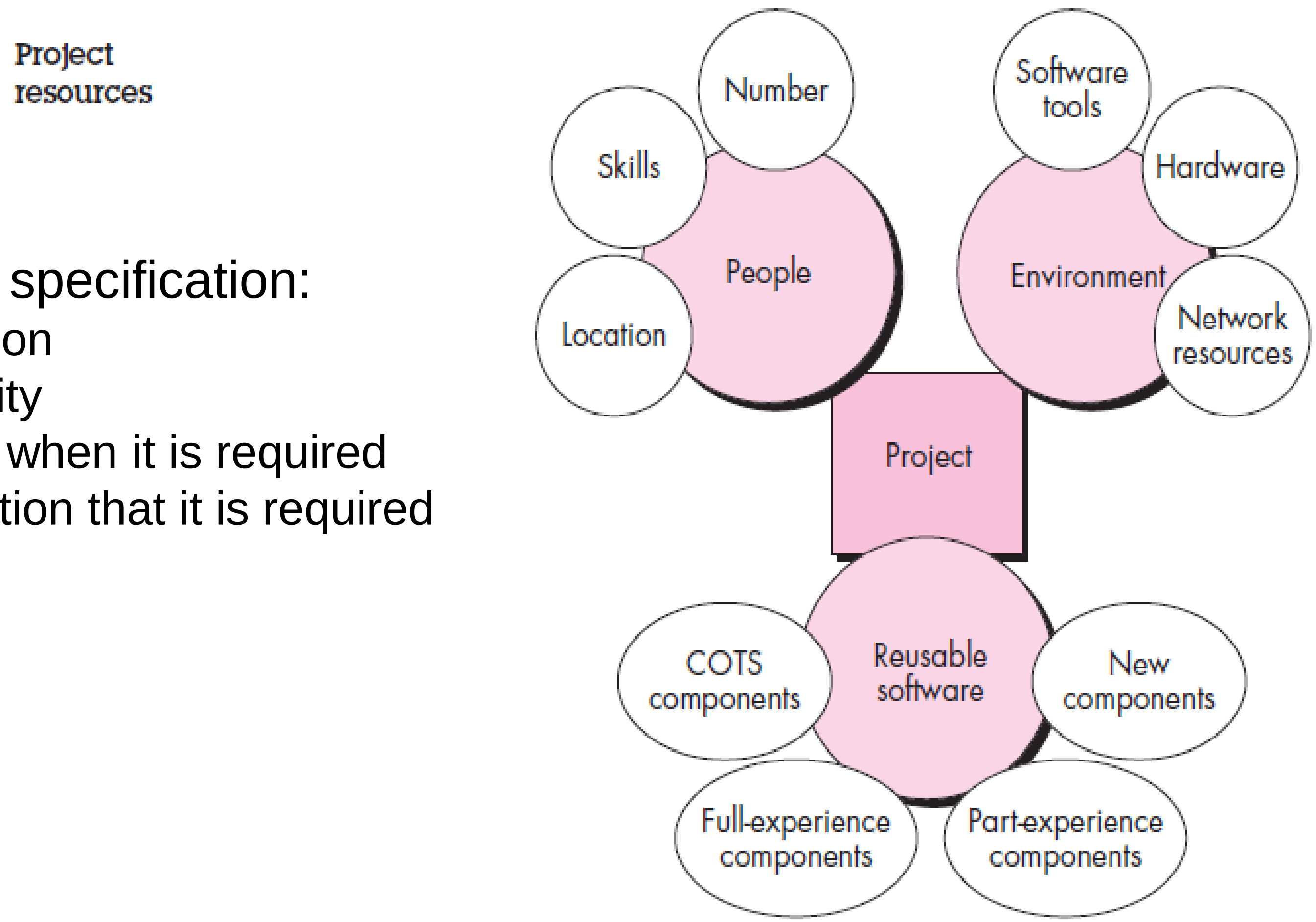
# Estimation

- 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, which leads to uncertainty

# 33.3 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?

# 33.4 Resources




Resource specification:

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

# 33.5 Software Project Estimation

- For estimation of resources, cost, and schedule
  - 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 estimation

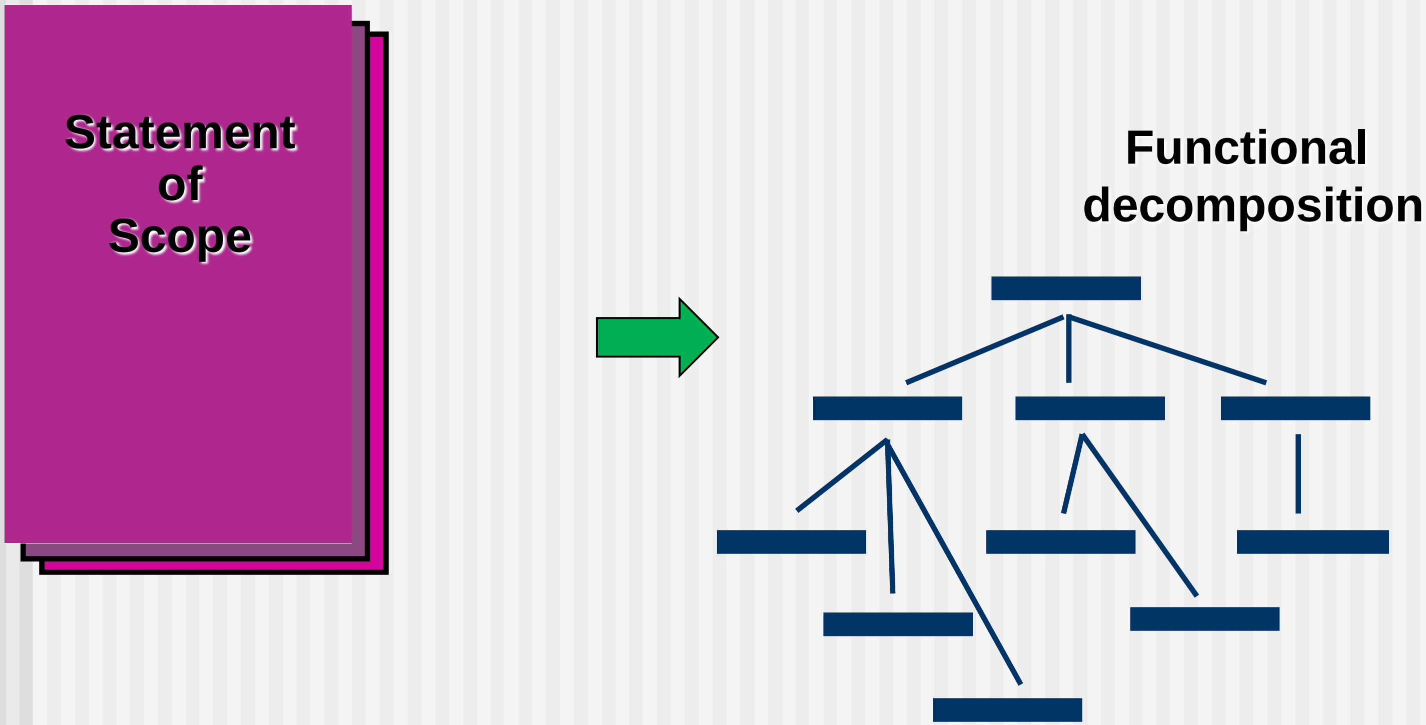
# Estimation Techniques

- 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 (a function of the availability of reliable software metrics from past projects)
  - the abilities of the software team
  - the stability of product requirements
  - the environment that supports the development effort

# 33.6 Decomposition Techniques



# Problem Decomposition

- Sometimes called partitioning or problem elaboration

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

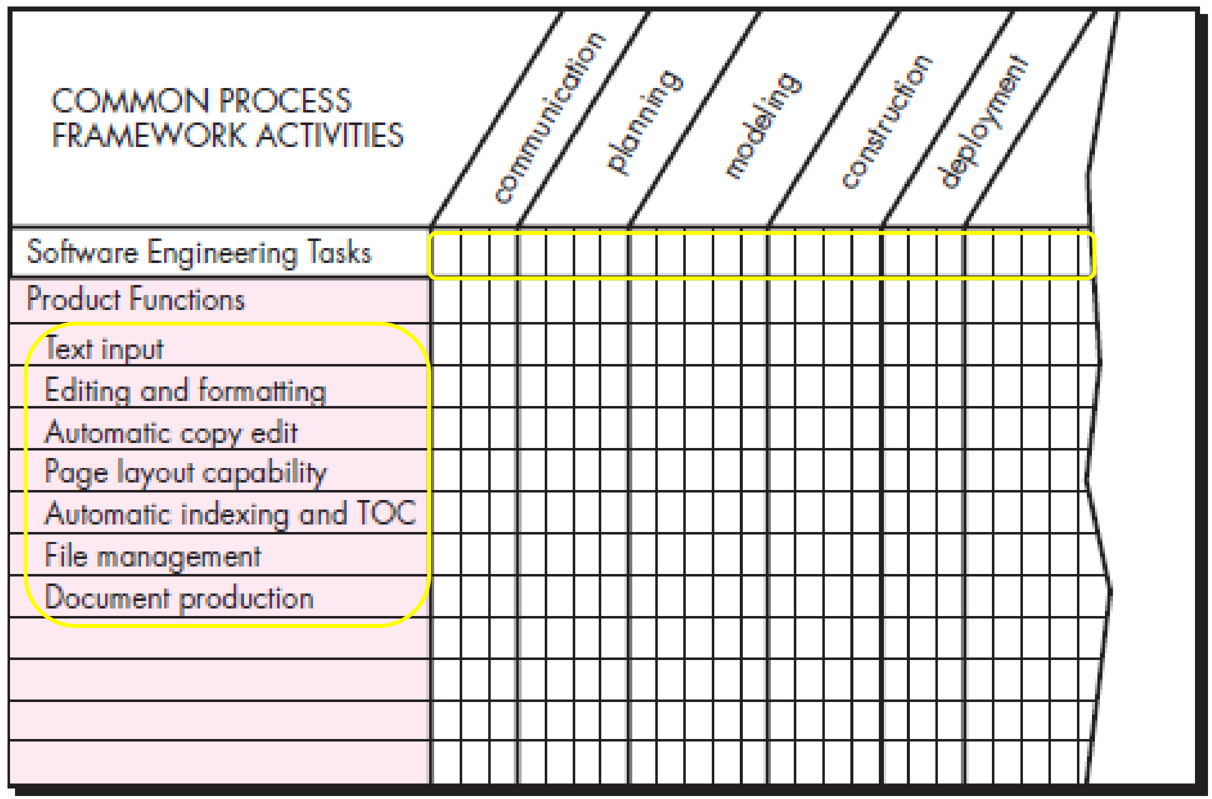
  *or*
  - It is decomposed into a set of problem classes

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

# Process Decomposition

- 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



# 33.6.1 Software Sizing

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

# 33.6.2 Problem-based Estimation

- LOC/FP
  - 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

# 33.6.3 An Example of 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

- The cost per LOC ≈ $13.

- The total estimated project cost = $431,000.
- The estimated effort = 54 person-months.

# 33.6.4 An Example of FP-Based Estimation

- The Function Point (FP) metric
  - Proposed by [Albrecht 79]
  - An effective means for measuring the functionality delivered by a system.
  - Derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity
- Information domain values consists of the counts of :
  - External inputs (EIs)
  - External outputs (EOs)
  - External inquiries (EQs)
  - Internal logical files (ILFs)
  - External interface files (EIFs)

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 5 | 6 | = | |
| Internal Logical Files (ILFs) | | × | 7 | 10 | 15 | = | |
| External Interface Files (EIFs) | | × | 5 | 7 | 10 | = | |
| Count total | | | | | | → | |

Unadjusted FP

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

(See the next slide)

# 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 adjustment factor** | **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 | 5 | 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* |

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

<u>Historical productivity data</u>

Organizational average productivity = 6.5 FP/pm.

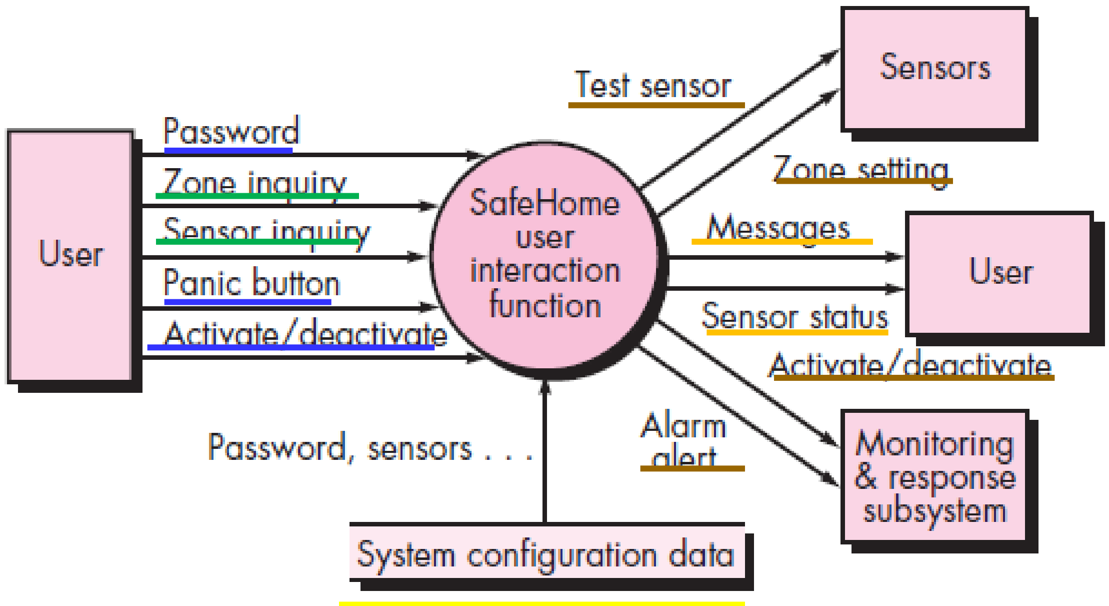
Burdened labor rate = \$8000 per month, approximately \$1230/FP.

☛The total estimated project cost = \$461,000

☛ The estimated effort = 58 person-months.

# Example 2

A data flow model for *SafeHome* software




Computing function points

**Information Domain Value**

External Inputs (EIs)

External Outputs (EOs)

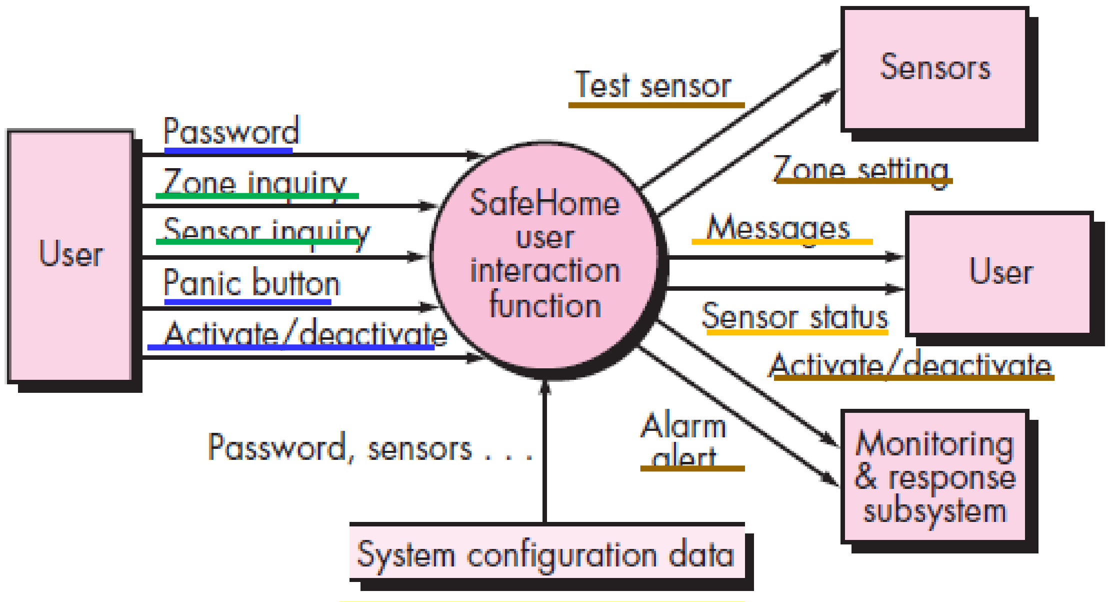
External Inquiries (EQs)

Internal Logical Files (ILFs)

External Interface Files (EIFs)

# 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

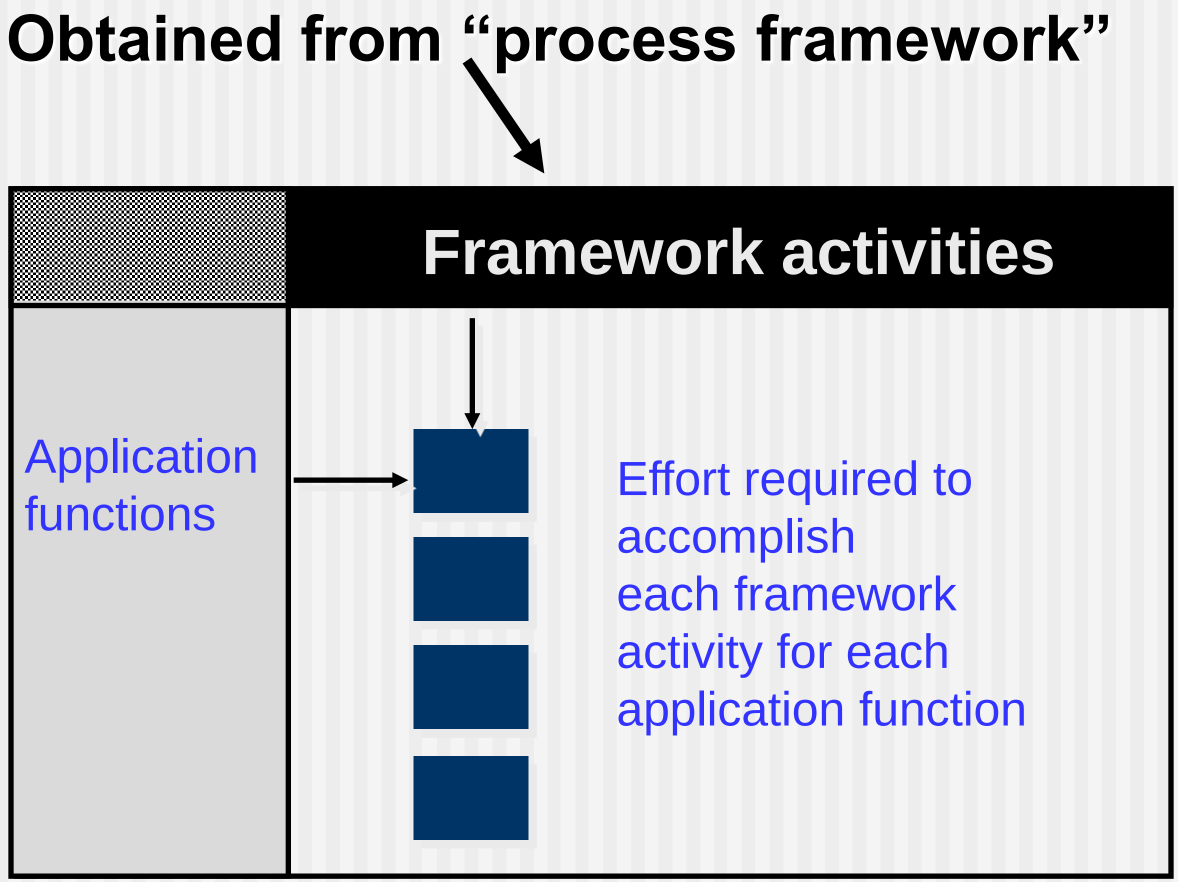
Then $FP = 50 \times [\underbrace{0.65 + (0.01 \times 46)}_{1.11}] = 56$

- FPs can be used to estimate LOC depending on the AVerage number of LOC (AVC) per FP for a given language
  - LOC = AVC * number of function points
  - AVC: 200 ~ 300 for assemble language

    2 ~ 40 for a 4GL

How many debugged LOC can you write per day?

# 33.6.5 Process-Based Estimation



# 33.6.6 An Example of 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 person-months
- The total estimated project cost = $368,000

# 33.6.7 Estimation with Use-Cases

**UCP = (UUCW + UAW) x TCF x 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 |

# 33.6.8 An Example of 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 | Factor |
| --- | --- |
| Simple | 1 |
| Average | 2 |
| Complex | 3 |

Assume: 8 simple actors, 12 average actors, 4 complex actors

TCF = 1.04

ECF = 0.96

UUCW = 16 × 15 + [14 × 10 + 8 × 5] + 10 × 5 = 470

UAW = 8 × 1 + 12 × 2 + 4 × 3 = 44

UCP = (UUCW + UAW) × TCF × ECF = (470+44) × 1.04 × 0.96 = 51.3

# 33.6.9 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 PMs (derived using a process-based estimation approach) to
  (2) a high of 68 PMs (derived with use-case estimation).

  (3) The average estimate (using all four approaches) is 56 PMs.

  (LOC, FP, Process-based, Use case-based)

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

# 33.7 Empirical Estimation Models
## 33.7.1 The Structure of Estimation Models

***General form:***

$$\text{Effort} = \text{Tuning coefficient} * \text{Size}^{\text{Exponent}}$$

usually derived as person-months of effort required

either a constant or a number derived based on complexity of project

usually LOC but may also be function point

empirically derived

# 33.7.2 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.

# The COCOMO Model


B. Boehm (1935 ~ )

- 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 the initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO II

- COCOMO-81 assumes
  (1) Waterfall process
  (2) Software developed from scratch

- COCOMO II takes into account different approaches to software development, reuse, etc.
  http://sunset.usc.edu/research/COCOMOII/index.html

# The Basic COCOMO 81 Model

| Project Complexity | Formula | Description |
|---|---|---|
| Simple | PM = 2.4 $(KDSI)^{1.05}$ M | Well-understood applications developed by small teams |
| Moderate | PM = 3.0 $(KDSI)^{1.12}$ M | More Complex projects where team members may have limited experience of related systems |
| Embedded | PM = 3.6 $(KDSI)^{1.20}$ 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

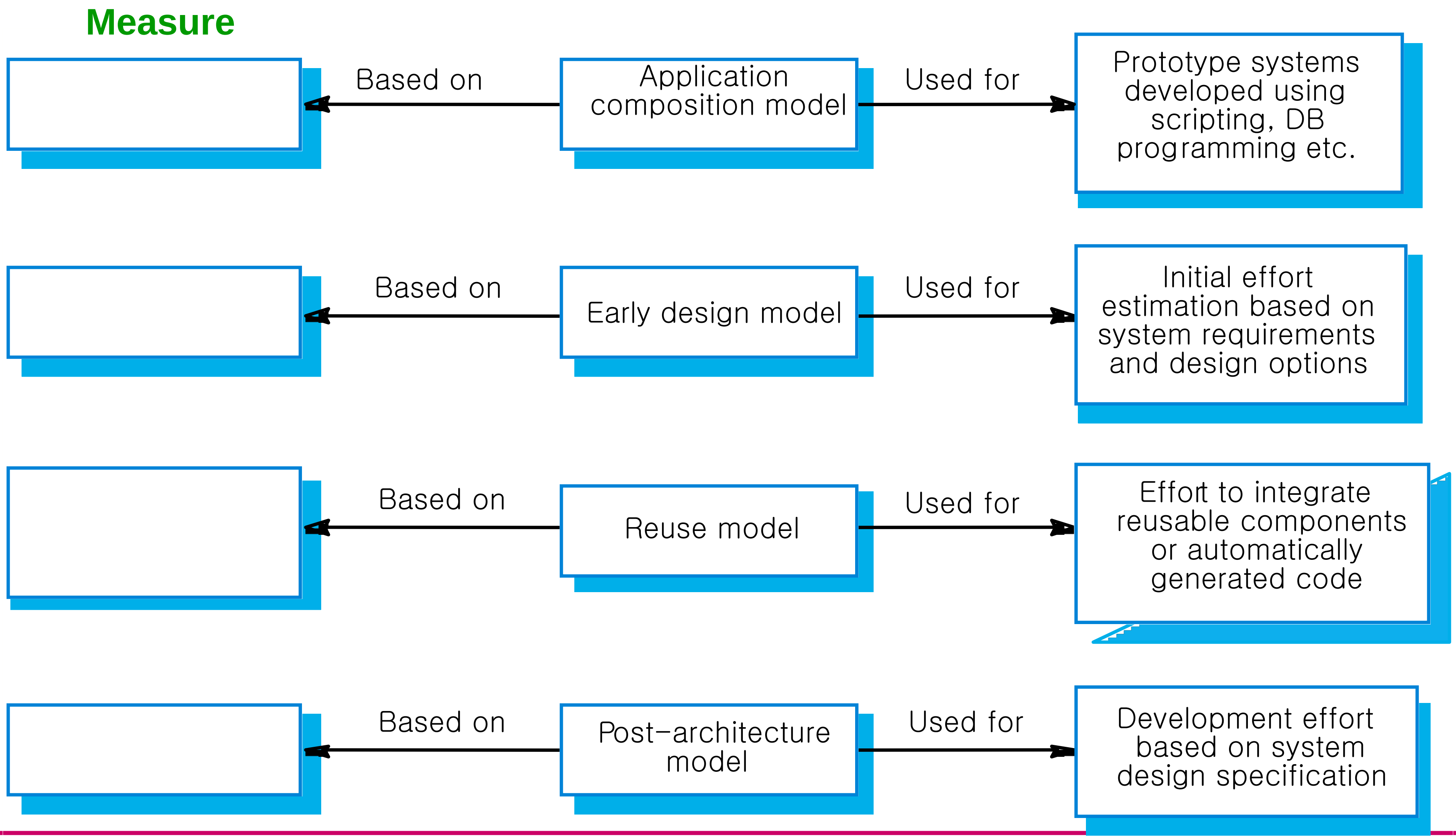
M: Multiplier

# COCOMO II

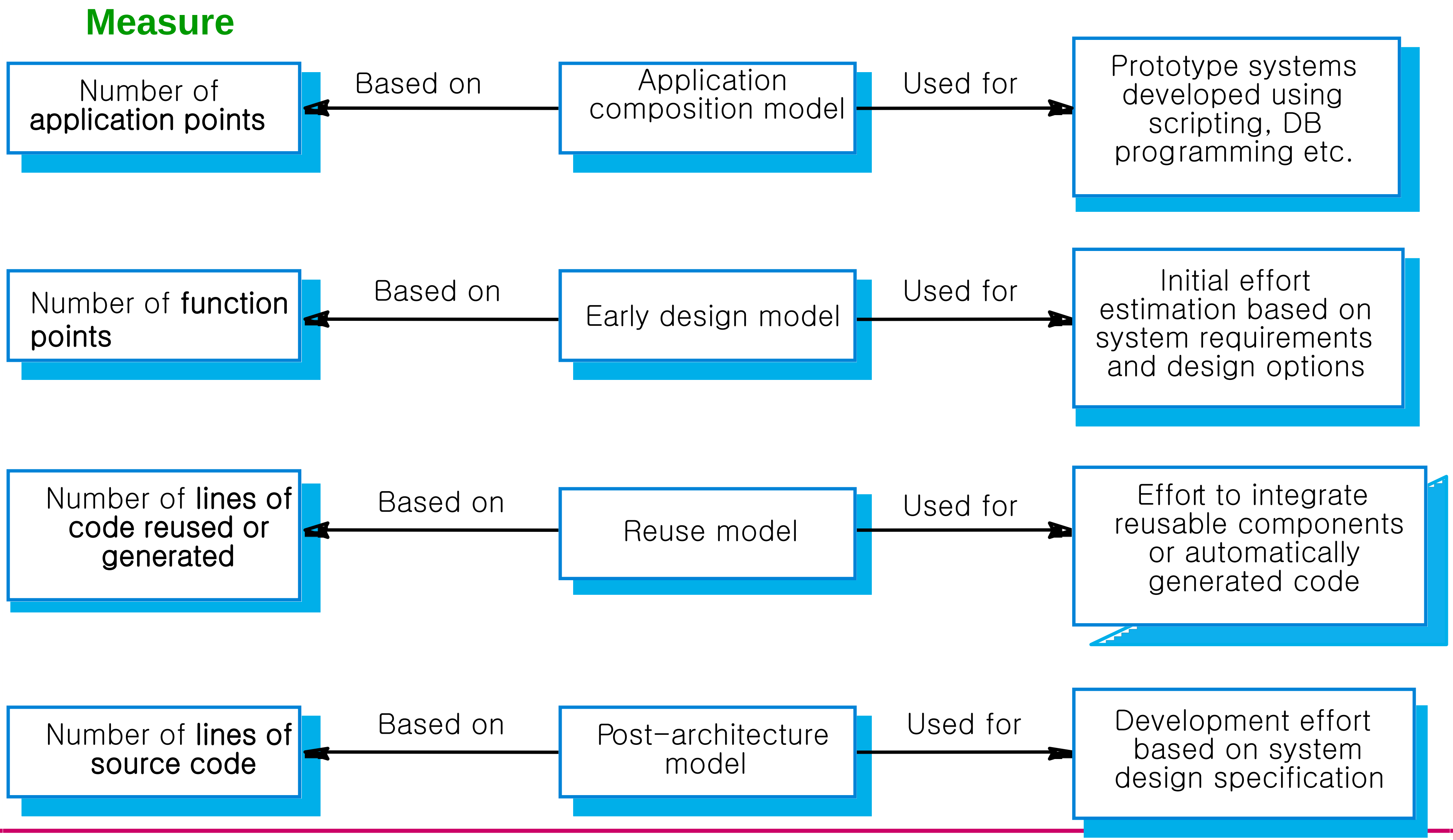
- COCOMO II incorporates a range of sub-models that produce increasingly detailed software estimates.

- 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



# (1) Application Composition Model

- Supports prototyping projects and projects where there is extensive reuse.
- Based on standard estimates of developer productivity in application (object) points/month.

**PM = ( NAP × (1 - %reuse/100 ) ) / PROD**

PM: effort in person-months

NAP: number of application points

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 function points 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.

- 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(= No. APs / month) | 4 | 7 | 13 | 25 | 50 |

# (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

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

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

We know that typically 1.05 <= M <= 1.20. (See next slides.)

# Influence of Cost Drivers

| | | |
|---|---|---|
| Exponent value | | 1.17 |
| System size (including factors for reuse and requirements volatility) | | 128, 000 DSI |
| **Initial COCOMO estimate without cost drivers** | | **730 person-months** |
| Reliability | Maximum | 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 | Minimum | 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** |

Only 5 key cost factors used here. Other factors are 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 * 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 * AT/100 => Lines of reused code

**Example**

Typically ATPROD = 2,400 SLOC/month.

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 changing and understanding the reused code*
    - Consists of:

      1) Adaptation cost

      2) Understanding cost:

         (5 times cost to understand unstructured code)

      3) Assessment cost: whether to adapt or not

    **ESLOC**: Equivalent SLOC

# (4) Post-Architecture Model

- Uses the same formula as the early design model

  **Effort = 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

# 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.

**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 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://csse.usc.edu/tools/COCOMOII.php



**COCOMO II - Constructive Cost Model**

Model(s): COCOMO
Monte Carlo Risk: Off
Auto Calculate: Off

**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 |
| Development Flexibility | Nominal |
| Architecture / Risk Resolution | Nominal |
| Team Cohesion | Nominal |
| Process Maturity | Nominal |

**Software Cost Drivers** (17 multipliers)

**Product**

| | |
|---|---|
| Required Software Reliability | Nominal |
| Data Base Size | Nominal |
| Product Complexity | Nominal |
| Developed for Reusability | Nominal |
| Documentation Match to Lifecycle Needs | Nominal |

**Personnel**

| | |
|---|---|
| Analyst Capability | Nominal |
| Programmer Capability | Nominal |
| Personnel Continuity | Nominal |
| Application Experience | Nominal |
| Platform Experience | Nominal |
| Language and Toolset Experience | Nominal |

**Platform**

| | |
|---|---|
| Time Constraint | Nominal |
| Storage Constraint | Nominal |
| Platform Volatility | Nominal |

**Project**

| | |
|---|---|
| Use of Software Tools | Nominal |
| Multisite Development | Nominal |
| Required Development Schedule | Nominal |

# 33.7.3 The Software Equation(SKIP)

## A dynamic multivariable model

$$E = [LOC \times B^{0.333}/P]^3 \times (1/t^4)$$

where

E = effort in person-months or person-years

t = project duration in months or years

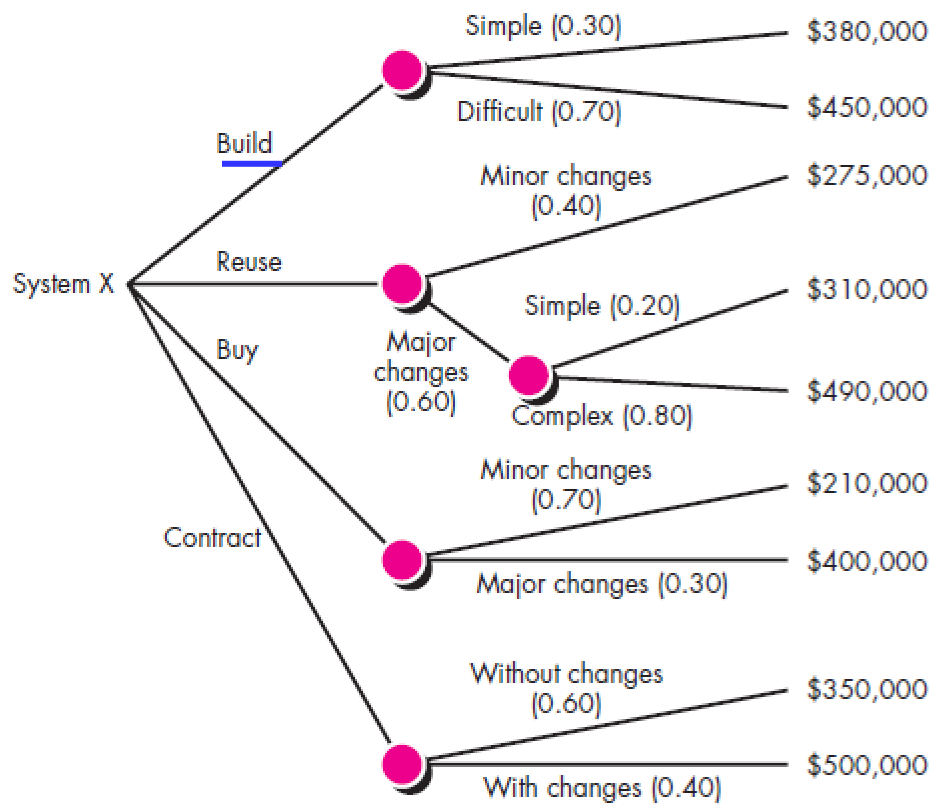
B = “special skills factor”

P = “productivity parameter”

# 33.10 The Make-Buy Decision
## 33.10.1 Creating a Decision Tree

probability

A decision tree to support the make/buy decision



# Computing Expected Cost

**Expected cost =**

$$\sum (\text{Path probability})_i \times (\text{Estimated path cost})_i$$

***For example, the expected cost to build is:***

**Expected cost$_{build}$ = 0.30 ($380K) + 0.70 ($450K)**

**= $429 K**

***similarly,***

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

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

Expected cost$_{contr}$ = $410K

# 33.10.2 Outsourcing

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