Tutorial:
Software Cost Estimation with COCOMO 2.0

Barry Boehm, USC
COCOMO/SCM Forum 1996
October 9, 1996
Outline

- Steps in Software Cost Estimation
  - 10 years of TRW experience
- Integrated Estimation, Measurement, and Management
  - Getting to CMM levels 4 and 5
  - Role of COCOMO 2.0
- COCOMO 2.0 Model Overview
- COCOMO 2.0 Status and Plans
- Information Sources
EXAMPLE SOFTWARE PROJECT COST HISTORY

DEVENNY, 1976

THOUSANDS OF DOLLARS, S.K.

ESTIMATED COST

ACTUAL COST

NUMBER OF MONTHS AFTER CONTRACT AWARD
Steps in Software Cost Estimation

1. Establish Objectives
   - Rough sizing
   - Make-or-Buy
   - Detailed Planning

2. Allocate Enough Time, Dollars, Talent

3. Pin Down Software Requirements
   - Documents Definition, Assumptions
   - Work out as much detail as Objectives permit
   - Use several independent techniques + sources
     - Top-Down vs. Bottom-Up
     - Algorithm vs. Expert Judgement
   - Compare and iterates estimates
     - Pin down and resolve inconsistencies
     - Be Conservative

4. Follow Up
Cost Estimating Objectives

- Relative vs. Absolute estimates
  - Selection vs. Resource Allocation Decisions
- Balanced Estimates
  - Degree of uncertainty vs. Size
- Generous vs. Conservative estimates

*Key objects to decision needs
*Re-example and iterate objectives as appropriate
SOFTWARE ESTIMATION ACCURACY VS. PHASE

- SIZE (USI)
- COST ($)

FEASIBILITY PLANS AND ROTS PRODUCT DESIGN DETAIL DESIGN DEVELOP. AND TEST

CONCEPT OF OPERATIONS DERIVED SPEC. PRODUCT SPEC. DETAIL SPEC. ACCEPTED SOFTWARE

PHASES AND MILESTONES

BB 04/22/96
Table 21-1 Software Cost-Estimating Miniproject Plan

1. Purpose. Why is the estimate being made?

2. Products and Schedules. What is going to be furnished when?

3. Responsibilities. Who is responsible for each product? Where are they going to do the job (organizationally? geographically?).

4. Procedures. How is the job going to be done? Which cost-estimation tools and techniques will be used (see Chapter 22)?

5. Required Resources. How much data, time, money, effort, etc. is needed to do the job?

6. Assumptions. Under what conditions are we promising to deliver the above estimates, given the above resources (availability of key personnel, computer time, user data)?
Software Cost Estimation Plan: Zenith Rapid Transit System

- **Purpose**: To help determine the feasibility of a computer controlled rapid-transit system for the Zenith metropolitan area.

- **Products and Schedules**:
  - 2/1/97  --  Cost Estimation Plan
  - 2/15/97 --  First cost model run
  - 2/22/97 --  Definitive cost model run
    - Expert estimates complete

- **Responsibilities**:
  - Cost estimation study: Z.B. Zimmerman
  - Cost model support: Applications Software Department
  - Expert estimators (2): Systems Analysis Department
Software Cost Estimation Plan:
Zenith Rapid Transit System (Contd.)

- **Procedures.** Project will use SOFCOST model, with sensitivity analysis on high-leverage cost driver attributes. Experts will contact BART personnel in San Francisco and Metro personnel in Washington, D.C. for comparative data.

- **Required Resources.**
  - Z.B. Zimmerman: 2 man-weeks
  - Expert estimators: 3 man-days each
  - Computer: $200

- **Assumptions.**
  - No major changes to system specification dated 15 January 1997. Authors of specification available to answer sizing questions.
WHAT DOES A SOFTWARE PRODUCT DO?

SUMMARY OF TWO PRODUCTS DEVELOPED TO SAME BASIC SPECIFICATION (SMALL, INTERACTIVE SOFTWARE COST MODEL)
WHAT DOES A SOFTWARE PROJECT DO?

DISTRIBUTION OF PROJECT TIME BY ACTIVITY

SUMMARY OF TWO TEAMS DEVELOPING PRODUCT TO SAME BASIC SPECIFICATION (SMALL, INACTIVE SOFTWARE COST MODEL)
WHAT DO PROGRAMMERS DO?

BELL LABS TIME AND MOTION STUDY

- BARDAIN, 1964
- 70 PROGRAMMERS

- WRITE PROGRAMS: 13
- READ PROGRAMS, MANUALS: 16
- TRAINING: 6
- MAIL, MISC DOC: 5
- MISC (WALKING, OFFSITE,...): 15
- JOB COMMUNICATION: 32
- PERSONAL: 13
## Strengths and Weaknesses of Software Cost-Estimation Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithmic model</td>
<td>Objective, repeatable, analyzable formula</td>
<td>Subjective inputs</td>
</tr>
<tr>
<td></td>
<td>Efficient, good for sensitivity</td>
<td>Assortment of exceptional circumstances</td>
</tr>
<tr>
<td></td>
<td>Objectively calibrated to experience</td>
<td>Calibrated to past, no future</td>
</tr>
<tr>
<td>Expert judgment</td>
<td>Assessment of representatives, interactions, exceptional circumstances</td>
<td></td>
</tr>
<tr>
<td>Analogy</td>
<td>Based on representative experience</td>
<td></td>
</tr>
<tr>
<td>Parkinson</td>
<td>Correlates with some experience</td>
<td></td>
</tr>
<tr>
<td>Price to win</td>
<td>Often wins</td>
<td></td>
</tr>
<tr>
<td>Top-down</td>
<td>System level focus</td>
<td>No better than participants</td>
</tr>
<tr>
<td></td>
<td>Efficient</td>
<td>Biases, incomplete recall</td>
</tr>
<tr>
<td>Bottom-up</td>
<td>More detailed basis</td>
<td>Representativeness of experience</td>
</tr>
<tr>
<td></td>
<td>More stable</td>
<td>Reinforces poor practice</td>
</tr>
<tr>
<td></td>
<td>Fosters individual commitment</td>
<td>Generally produces large overruns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Less detailed basis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Less stable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>May overlook system level costs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Requires more effort</td>
</tr>
</tbody>
</table>
Comparing and Iterating Estimates

- Complementarity of techniques
- Missing items
- The “tall pole in the tent”
  - Pareto (80-20) principle
- The optimist/pessimist phenomenon
SOFTWARE COST ESTIMATION

ACCURACY VS. PHASE

EXAMPLE SOURCES OF UNCERTAINTY:
MAN-MACHINE INTERFACE SOFTWARE

- CONCEPT OF OPERATION
- PRODUCT DESIGN SPEC.
- DETAIL DESIGN SPEC.
- ACCEPTED SOFTWARE

CLASSES OF PEOPLE, DATA SOURCES TO SUPPORT
- QUERY TYPES, DATA LOADS, SMART-TERMINAL TRADEOFFS, RESPONSE TIMES
- INTERNAL DATA STRUCTURE
- BUFFER HANDLING TECHNIQUES
- DETAILED SCHEDULING ALGORITHMS, ERROR HANDLING

0.25x
0.5x
1x
1.25x
2x
4x

RELATIVE COST RANGE

FEASIBILITY
PLANS AND ROTS
PRODUCT DESIGN PHASES AND MILESTONES
DEVEL. AND TEST

BB 04/22/96

A A A

A A A

A A A

A A A

A A A

A A A

A A A

A A A

A A A

A A A

A A A

A A A

A A A
Judging the Soundness of Cost Estimates: Some Review Questions

- How would the estimated cost change if:
  - There was a big change in the ________ interface?
  - We had to supply a more extensive ________ capability?
  - Our workload estimates were off by 50%?
  - We had to do the job on a ________ hardware configuration?

- Suppose our budget was 20% (less, greater). How would this affect the product?

- How does the cost of this item break down by component?
Judging the Optimism of Cost Estimates: Some Review Questions

- How do the following compare with previous experience:
  - Sizing of subsystems?
  - Cost driver ratings?
  - Cost per instruction?
  - Assumptions about hardware? GFE?
  - Subcontractors? Vendor software?

- Have you done a cost-risk analysis?
  - What are the big-risk items?

- Suppose we had enough money to really do this job right
  - How much would we need?
Software Cost Estimation: Need for Follow-up

- Estimating inputs are imperfect
- Estimating techniques are imperfect
- Project may not fit model
  - Incremental development
- Software projects are volatile
- Software is an evolving field
  - COTS, objects, cyberspace, new processes
Management-Relevant, Hypothesis Driven Data: TRW Data Collection Policy

- Identify major project milestone

- At each milestone:
  - Re-do cost model inputs
  - Re-run cost model
  - Collect actuals corresponding to cost model outputs
  - Compare results
Outline

- Steps in Software Cost Estimation
  - 10 years of TRW experience
- Integrated Estimation, Measurement, and Management
  -- Getting to CMM levels 4 and 5
  -- Role of COCOMO 2.0
- COCOMO 2.0 Model Overview
- COCOMO 2.0 Status and Plans
- Information Sources
Key Process Areas by Level
Level 4 (Managed)

- Process Measurement and Analysis
  - Statistical process control
  - Cost/schedule/quality tradeoffs understood
  - Causes of variation identified and controlled

- Quality Management
  - Measurable quality goals and priorities
  - Quality goals drive product and process plans
  - Measurements used to manage project
Key Process Areas by Level

Level 5 (Optimizing)

- **Defect Prevention**
  - Defect sources identified and eliminated

- **Technology Innovation**
  - Processes for technology tracking, evaluation, assimilation
  - Assimilation tied to process quality and productivity improvements

- **Process Change Management**
  - Commitment to process improvement goals
  - Evidence of measured improvements
  - Improvements propagate across the organization
Role of COCOMO 2.0

- Supports modern application of TRW data collection policy
  - Objects, GUI builders, reuse, new processes
  - COTS integration, quality estimation underway
- Use data to recalibrate model
- Use model to optimize new technology investments
COCOMO 2.0 Long Term Vision

- System objectives: function, performance, quality
- Rescope
  - OK?
  - Yes
  - No
  - Milestone plans, resources
  - M/S Results
  - OK?
  - Yes
  - Revised expectations
  - No
  - Done?
  - Yes
  - End
- Execute project to next milestone
- Revise Milestones, Plans, Resources
- Corporate parameters: tools, processes, reuse
- Cost, Sched, Risks
- Milestone expectations
- Resources
- Cost, Sched, Quality drivers
- Revised Corporate SW Improvement Strategies
- Recalibrate COCOMO 2.0
- Evaluate Corporate SW Improvement Strategies
- COCOMO 2.0
- Yes
- Revised
- Accumulate COCOMO 2.0 calibration data
- Revisions
- Yes
- L---
Outline

- Steps in Software Cost Estimation
  - 10 years of TRW experience
- Integrated Estimation, Measurement, and Management
  - Getting to CMM levels 4 and 5
  - Role of COCOMO 2.0

- COCOMO 2.0 Model Overview
- COCOMO 2.0 Status and Plans
- Information Sources
COCOMO 2.0 Model Overview

- COCOMO Baseline Overview
- COCOMO 2.0 Objectives
- Coverage of Future Market Sectors
- Hierarchical Sizing Model
- Modes Replaced by Exponent Drivers
- Stronger Reuse/Reengineering Model
- Other Model Improvements
- Activity Sequence
COCOMO Baseline Overview I

- Software product size estimate
- Software product, process, computer and personnel attributes
- Software reuse, maintenance, and increment parameters
- Software organization’s project data

COCOMO

- Software development, maintenance cost and schedule estimates
- Cost, schedule distribution by phase, activity, increment
- COCOMO recalibrated to organization’s data
COCOMO Baseline Overview II

- Open interfaces and internals
  - Published in *Software Engineering Economics*, Boehm, 1981
- Numerous implementations, calibrations, extensions
  - Incremental development, Ada, new environment technology
  - Arguably the most frequently-used software cost model worldwide.
Partial List of COCOMO Packages
-From STSC Report, Mar. 1993

- CB COCOMO
- COCOMO1D
- COCOMO1
- CoCoPro
- COSTAR
- COSTMODL
- GECOMO Plus
- GHL COCOMO
- REVIC
- SECOMO
- SWAN
COCOMO 2.0 Objectives

- Retain COCOMO internal, external openness
- Develop a 1990-2000’s software cost model
  - Addressing full range of market sectors
  - Addressing new processes and practices
- Develop database, tool support for continuous model improvement
- Develop analytic framework for economic analysis
  - E.g., software product line return-on-investment analysis
- Integrate with process, architecture research
The future of the software practices marketplace

User programming
(55M performers in US in year 2005)

Application generators (0.6M)
Application composition (0.7M)
System integration (0.7M)

Infrastructure (0.75M)
COCOMO 2.0 Coverage of Future SW Practices Sectors

- User Programming: No need for cost model
- Applications Composition: Use object counts or object points
  - Count (weight) screens, reports, 3GL routines
- System Integration; development of applications generators and infrastructure software
  - Prototyping: Applications composition model
  - Early design: Function Points and 7 cost drivers
  - Post-architecture: Source Statements or Function Points and 17 cost drivers
- Stronger reuse/reengineering model
Applications Composition

- Challenge:
  - Modeling rapid applications composition with graphical user interface (GUI) builders, client-server support, object-libraries, etc.

- Response:
  - Object-Point sizing and costing model
  - Reporting estimate ranges rather than point estimates
Step 1: Assess Object-Counts: estimate the number of screens, reports, and 3GL components that will comprise this application. Assume the standard definitions of these objects in your iCASE environment.

Step 2: Classify each object instance into simple, medium and difficult complexity levels depending on values of characteristic dimensions. Use the following scheme:

<table>
<thead>
<tr>
<th>For Screens</th>
<th>For Reports</th>
</tr>
</thead>
<tbody>
<tr>
<td># and source of data tables</td>
<td># and source of data tables</td>
</tr>
<tr>
<td>Number of Views Contained</td>
<td>Total &lt; 4 (&lt;2 srvr, c3 clnt)</td>
</tr>
<tr>
<td>&lt;3</td>
<td>simple</td>
</tr>
<tr>
<td>3-7</td>
<td>simple</td>
</tr>
<tr>
<td>8+</td>
<td>medium</td>
</tr>
</tbody>
</table>

Step 3: Weigh the number in each cell using the following scheme. The weights reflect the relative effort required to implement an instance of that complexity level.

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Complexity-Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen</td>
<td>Simple</td>
</tr>
<tr>
<td>Report</td>
<td>2</td>
</tr>
<tr>
<td>3GL Component</td>
<td>1</td>
</tr>
</tbody>
</table>

Step 4: Determine Object-Points: add all the weighted object instances to get one number, the Object-Point count.

Step 5: Estimate percentage of reuse you expect to be achieved in this project. Compute the New Object Points to be developed, \( NOP = (\text{Object-Points} \times (100 - \%\text{reuse}) / 100. \)

Step 6: Determine a productivity rate, \( \text{PROD} = \frac{\text{NOP}}{\text{person-month}} \), from the following scheme:

<table>
<thead>
<tr>
<th>Developer's experience and capability</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICASE maturity and capability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROD</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

Step 7: Compute the estimated person-months: \( \text{PM} = \frac{\text{NOP}}{\text{PROD}}. \)
**SW Costing and Sizing Accuracy vs. Phase**

- **Completed Programs**
- **USAF/ESD Proposals**
- **Size (DSI)**
- **+ Cost ($)**

**Relative Size Range**

- **0.25X**
- **0.5X**
- **1X**
- **1.25X**
- **2X**
- **4X**

**Phases and Milestones**

- **Feasability**
- **Plans and Rqts.**
- **Product Design Spec.**
- **Detail Design Spec.**
- **Accepted Software**

**BB 04/22/96**
New Processes

- Challenges
  - Cost and schedule estimation of composable mixes of prototyping, spiral, evolutionary, incremental development, other models

- Responses
  - New milestone definitions
    » Basic requirements consensus, life-cycle architecture
  - Replace development modes by exponent drivers
  - Post-Initial Operational Capability (IOC) evolution
    » Drop maintenance/reuse distinction
Process/Anchor Point Examples


Spiral-type  Waterfall, Spiral-type  W’fall, IncDev, EvDev, Spiral, Design-to-Cost, etc.

LCO  LCA  IOC
## Life-cycle Architecture as Critical Milestone

<table>
<thead>
<tr>
<th>Candidate Critical Milestones</th>
<th>Concerns</th>
</tr>
</thead>
</table>
| Complete Requirements Specifications | - Premature decisions, e.g. user interface features  
- Inflexible point solutions  
- Gold plating |
| Beta-Test Code | - High-risk downstream functions  
- Inflexible point solutions  
- Capabilities too far from user needs |
| Life-Cycle Architecture | - Provides flexibility to address concerns above  
Further concerns:  
- Predicting directions of growth, evolution  
- High-risk architectural elements |
Early Design and Post-Arch Model:

- **Effort:**
  \[ PM_{\text{estimated}} = A \times (\text{Size})^\text{(SF)} \times \prod_{i} EM_i \]

- **Size**
  - KSLOC (Thousands of Source Lines of Code)
  - UFP (Unadjusted Function Points)
  - EKSLOC (Equivalent KSLOC) used for adaptation

- **SF:** Scale Factors (5)
- **EM:** Effort Multipliers (7 for ED, 17 for PA)
IMPORTANCE OF CLEAR DEFINITIONS

SCOPE
- Include Conversion? Training? Documentation?

ENDPOINTS
- Include requirements analysis? Full validation?

INSTRUCTIONS

PRODUCT
- Application? Support S/W? Test Drivers?

"DEVELOPMENT"
- Include reused software? Furnished Utility S/W?

"MAN MONTHS"
- Include Clerical and Support Personnel? Vacation?

These can bias results by factors of 2-10
Unadjusted Function Points

Step 1: **Determine function counts by type.** The unadjusted function counts should be counted by a lead technical person based on information in the software requirements and design documents. The number of each of the five user function types should be counted (Internal Logical File (ILF), External Interface File (EIF), External Input (EI), External Output (EO), and External Inquiry (EQ)).

Step 2: **Determine complexity-level function counts.** Classify each function count into Low, Average, and High complexity levels depending on the number of data element types contained and the number of file types reference. Use the following scheme:

<table>
<thead>
<tr>
<th>Record Elements</th>
<th>Data Elements</th>
<th>ILF Type</th>
<th>EI Type</th>
<th>EO Type</th>
<th>EQ Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-5</td>
<td>Low</td>
<td>Low</td>
<td>0 or 1</td>
<td>Low</td>
</tr>
<tr>
<td>2-5</td>
<td>6-10</td>
<td>Avg</td>
<td>Avg</td>
<td>2-3</td>
<td>Avg</td>
</tr>
<tr>
<td>6+</td>
<td>11+</td>
<td>High</td>
<td>High</td>
<td>4+</td>
<td>High</td>
</tr>
</tbody>
</table>

Step 3: **Apply complexity weights.** Weight the number in each cell using the following scheme. The weights reflect the relative value of the function to the user.

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Complexity-Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Internal Logical Files</td>
<td>7</td>
</tr>
<tr>
<td>External Interfaces Files</td>
<td>5</td>
</tr>
<tr>
<td>External Inputs</td>
<td>3</td>
</tr>
<tr>
<td>External Outputs</td>
<td>4</td>
</tr>
<tr>
<td>External Inquiries</td>
<td>3</td>
</tr>
</tbody>
</table>

Step 4: **Compute Unadjusted Function Points.** Add all the weighted functions counts to get one number, the Unadjusted Function Points.
## Source Lines of Code:

**Measurement Unit:**  
- Physical source lines  
- Logical source statements ✓

When a line or statement contains more than one type, classify it as the type with the highest precedence.

<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Definition</th>
<th>Data Array</th>
<th>Order of precedence</th>
<th>Includes</th>
<th>Excludes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Executable</td>
<td></td>
<td></td>
<td>1</td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>2. Nonexecutable</td>
<td></td>
<td></td>
<td>2</td>
<td>§</td>
<td>✓</td>
</tr>
<tr>
<td>3. Declarations</td>
<td></td>
<td></td>
<td>3</td>
<td>§</td>
<td>✓</td>
</tr>
<tr>
<td>4. Compiler directives</td>
<td></td>
<td></td>
<td>4</td>
<td>§</td>
<td>✓</td>
</tr>
<tr>
<td>5. Comments</td>
<td></td>
<td></td>
<td>5</td>
<td>§</td>
<td>✓</td>
</tr>
<tr>
<td>6. On their own lines</td>
<td></td>
<td></td>
<td>6</td>
<td>§</td>
<td>✓</td>
</tr>
<tr>
<td>7. On lines with source code</td>
<td></td>
<td></td>
<td>7</td>
<td>§</td>
<td>✓</td>
</tr>
<tr>
<td>8. Banners and nonblank spacers</td>
<td></td>
<td></td>
<td>8</td>
<td>§</td>
<td>✓</td>
</tr>
<tr>
<td>9. Blank (empty) comments</td>
<td></td>
<td></td>
<td>9</td>
<td>§</td>
<td>✓</td>
</tr>
<tr>
<td>10. Blank lines</td>
<td></td>
<td></td>
<td>10</td>
<td>§</td>
<td>✓</td>
</tr>
</tbody>
</table>

**How produced**

<table>
<thead>
<tr>
<th>Definition</th>
<th>Data Array</th>
<th>Includes</th>
<th>Excludes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Programmed</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>2. Generated with source code generators</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>3. Converted with automated translators</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>4. Copied or reused without change</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>5. Modified</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>6. Removed</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
</tbody>
</table>

**Origin**

<table>
<thead>
<tr>
<th>Definition</th>
<th>Data Array</th>
<th>Includes</th>
<th>Excludes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. New work: no prior existence</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>2. Prior work: taken or adapted from</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>3. A previous version, build, or release</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>4. Commercial, off-the-shelf software (COTS), other than libraries</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>5. Government furnished software (GFS), other than reuse libraries</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>6. Another product</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>7. A vendor-supplied language support library (unmodified)</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>8. A vendor-supplied operating system or utility (unmodified)</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>9. A local or modified language support library or operating system</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>10. Other commercial library</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>11. A reuse library (software designed for reuse)</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
<tr>
<td>12. Other software component or library</td>
<td></td>
<td>✓</td>
<td>§</td>
</tr>
</tbody>
</table>
Size Adjustment

- Breakage
- Nonlinear effects of reuse

\[ AAF = 0.4(DM) + 0.3(CM) + 0.3(IM) \]

\[ ESLOC = \frac{ASLOC \times [AA + AAF(1 + 0.02(SU) \times (UNFM))]}{100}, \ AAF \leq 0.5 \]

\[ ESLOC = \frac{ASLOC \times [AA + AAF(SU) \times (UNFM)]}{100}, \ AAF > 0.5 \]

- Automatic translation of code

\[ \frac{PM_{adjusted}}{PM_{estimated}} = \frac{AT}{100} \]

\[ \frac{ASLOC}{ATPROD} \]
COCOMO 2.0 Model Overview

- COCOMO Baseline Overview
- COCOMO 2.0 Objectives
- Coverage of Future Market Sectors
- Hierarchical Sizing Model
- Modes Replaced by Exponent Drivers
- Stronger Reuse/Reengineering Model
- Other Model Improvements
- Activity Sequence
**Project Scale Factors:**

\[ PM_{estimated} = A \times (\text{Size})^{(SF)} \times \prod_{i} EM_i \]

\[ SF = 1.01 + 0.01 \times \sum w_i \]

<table>
<thead>
<tr>
<th>Scale Factors</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREC</td>
<td>thoroughly unprecedented</td>
<td>largely unprecedented</td>
<td>somewhat unprecedented</td>
<td>general goals</td>
</tr>
<tr>
<td>FLEX</td>
<td>rigorous</td>
<td>occasional relaxation</td>
<td>some relaxation</td>
<td>general conformity</td>
</tr>
<tr>
<td>RESL</td>
<td>little (20%)</td>
<td>some (40%)</td>
<td>often (60%)</td>
<td>mostly (90%)</td>
</tr>
<tr>
<td>TEAM</td>
<td>very difficult interactions</td>
<td>some difficult interactions</td>
<td>basically cooperative interaction</td>
<td>highly cooperative</td>
</tr>
<tr>
<td>PMAT</td>
<td>weighted sum of KPA competition levels</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Nature of Life-Cycle Architecture

- Definition of life-cycle requirements envelope
- Definition of software components and relationships
  - physical and logical
  - data, control, timing relationships
  - shared assumptions
  - developer, operator, user, manager views
- Evidence that architecture will accommodate requirements envelope
- Resolution of all life-cycle-critical COTS, reuse, and risk issues
New Scaling Exponent Approach

- Nominal person-months = $A^*(\text{size})^B$
- $B = 1.01 + 0.01 \sum \text{(exponent driver ratings)}$
  - $B$ ranges from 1.01 to 1.26
  - 5 drivers; ratings from 0 to 5
- Exponent drivers:
  - Precedentedness
  - Development flexibility
  - Architecture risk resolution
  - Team cohesion
  - Process maturity (being derived from SEI CMM)
Reuse and Product Line Management

- Challenges
  - Estimate costs of both reusing software and developing software for future reuse
  - Estimate extra effects on schedule (if any)

- Responses
  - New nonlinear reuse model
  - Cost of developing reusable software expanded from Ada COCOMO
  - Gathering schedule data
Nonlinear Reuse Effects

Data on 2954 NASA modules [Selby, 1988]

Usual Linear Assumption
Reuse and Reengineering Effects

- Add Assessment & Assimilation increment (AA)
  - Similar to conversion planning increment
- Add software understanding increment (SU)
  - To cover nonlinear software understanding effects
  - Apply only if reused software is modified

Results in revised Adaptation Adjustment Factor (AAF)

- Equivalent $DSI = (Adapted \ DSI) \times (AAF/100)$
- $AAF = AA + SU + 0.4(DM) + 0.3(CM) + 0.3(IM)$
### Prototype of Software Understanding Rating / Increment

<table>
<thead>
<tr>
<th></th>
<th>Very Low</th>
<th>Low</th>
<th>Nom</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure</strong></td>
<td>Very low cohesion, high coupling, spaghetti code.</td>
<td>Moderately low cohesion, high coupling.</td>
<td>Reasonably well-structured; some weak areas.</td>
<td>High cohesion, low coupling.</td>
<td>Strong modularity, information hiding in data/control structures.</td>
</tr>
<tr>
<td><strong>Application Clarity</strong></td>
<td>No match between program and application world views.</td>
<td>Some correlation between program and application.</td>
<td>Moderate correlation between program and application.</td>
<td>Good correlation between program and application.</td>
<td>Clear match between program and application world views.</td>
</tr>
<tr>
<td><strong>Self-Descriptiveness</strong></td>
<td>Obscure code; documentation missing, obscure or obsolete.</td>
<td>Some code commentary and headers; some useful documentation.</td>
<td>Moderate level of code commentary, headers, documentation.</td>
<td>Good code commentary and headers; useful documentation; some weak areas.</td>
<td>Self-descriptive code; documentation up-to-date, well-organized, with design rationale.</td>
</tr>
</tbody>
</table>

| SU Increment to ESLCC | 50 | 40 | 30 | 20 | 10 |
Proposed Reuse / Maintenance Model (Change <= 20%)

AAF = 0.4(DM) + 0.3(CM) + 0.3(IM)

ESLOC = \[\text{ASLOC} \times [\text{AA} + \text{AAF}(1 + 0.02\text{(SU)} \times \text{(UNFM)})] \times \frac{\text{100}}{\text{AAF}}\]

ESLOC = \[\text{ASLOC} \times [\text{AA} + \text{AAF} \times \text{(SU)} \times \text{(UNFM)}] \times \frac{\text{100}}{\text{AAF}}\]

UNFM: Unfamiliarity with Software modifier
- 0.0 - Completely familiar
- 0.2 - Mostly familiar
- 0.4 - Somewhat unfamiliar
- 0.6 - Considerably unfamiliar
- 0.8 - Mostly unfamiliar
- 1.0 - Completely unfamiliar
Maintenance Model (Change > 20%)

Obtained in one of two ways depending on what is known:

$$ESLOC = (\text{BaseCodeSize} \cdot \text{MCF}) \cdot \text{MAF}$$

or

$$ESLOC = (\text{SizeAdded} + \text{SizeModified}) \cdot \text{MAF}$$

where

$$\text{MCF} = \frac{\text{SizeAdded} + \text{SizeModified}}{\text{BaseCodeSize}}$$

$$\text{MAF} = 1 + \left( \frac{SU}{100} \cdot UNFM \right)$$
Legacy Software Reengineering

- Challenge:
  - Varying reengineering efficiencies based on tool applicability, source-target system differences

- Response:
  - Exploratory model based on NIST IRS data
NIST Reengineering Case Study
[Ruhl-Gunn, 1991]

- IRS Centralized Scheduling Program
- Mix of batch, database query/update, on-line processing
- 50 KSLOC of COBOL74, 2.4 KSLOC of assembly code
- Unisys 1100 with DMS 1100 network database
- Reengineer to SQL database, open systems
- Tools for program and data reengineering
<table>
<thead>
<tr>
<th>Program Group</th>
<th>Automated</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch</td>
<td>96%</td>
<td>4%</td>
</tr>
<tr>
<td>Batch with SORT</td>
<td>90%</td>
<td>10%</td>
</tr>
<tr>
<td>Batch with DBMS</td>
<td>88%</td>
<td>12%</td>
</tr>
<tr>
<td>Batch, SORT, DBMS</td>
<td>82%</td>
<td>18%</td>
</tr>
<tr>
<td>Interactive</td>
<td>50%</td>
<td>50%</td>
</tr>
</tbody>
</table>
COCOMO 2.0 Model Overview

- COCOMO Baseline Overview
- COCOMO 2.0 Objectives
- Coverage of Future Market Sectors
- Hierarchical Sizing Model
- Stronger Reuse/Reengineering Model
- Other Model Improvements
Other Major COCOMO 2.0 Changes

- Range versus point estimates
- Requirements Volatility replaced by Breakage %
- Multiplicative cost driver changes
  - Product CD’s
  - Platform CD’s
  - Personnel CD’s
  - Project CD’s
- Maintenance model and reuse model identical
### Post-Architecture EMs - Product:

<table>
<thead>
<tr>
<th>Required Reliability (RELY)</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>slight inconvenience</td>
<td>low, easily recoverable losses</td>
<td>moderate, easily recoverable losses</td>
<td>high financial loss</td>
<td>risk to human life</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database Size (DATA)</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB bytes/ Pgm SLOC&lt;10</td>
<td>10sD/P&lt;100</td>
<td>100sD/P&lt;1000</td>
<td>D/P&lt;1000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Documentation Match to Lifecycle (DOCU)</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many lifecycle needs uncovered</td>
<td>none</td>
<td>across project</td>
<td>across program</td>
<td>across product line</td>
<td>across multiple product lines</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Complexity (CPLX)</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>see Complexity Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Post-Architecture EMs**
  - **Product:** Required Reliability (RELY)
  - Database Size (DATA)
  - Complexity (CPLX)
  - Required Reuse (RUSE)
  - Documentation Match to Lifecycle (DOCU)

- **Table:**
  - **Required Reliability (RELY):** slight inconvenience
  - **Database Size (DATA):** DB bytes/ Pgm SLOC<10
  - **Complexity (CPLX):** see Complexity Table
  - **Required Reuse (RUSE):** none
  - **Documentation Match to Lifecycle (DOCU):** Many lifecycle needs uncovered

- **Values:**
  - **Low:** low, easily recoverable losses
  - **Nominal:** moderate, easily recoverable losses
  - **High:** high financial loss
  - **Extra High:** risk to human life
  - **Low:** 10sD/P<100
  - **Nominal:** 100sD/P<1000
  - **High:** D/P<1000
  - **Low:** none
  - **Nominal:** across project
  - **High:** across program
  - **Extra High:** across product line
  - **Low:** many lifecycle needs uncovered
  - **Nominal:** right-sized to lifecycle needs
  - **High:** excessive for lifecycle needs
  - **Extra High:** very excessive for lifecycle needs

- **Notes:**
  - **Low:** low, easily recoverable losses
  - **Nominal:** moderate, easily recoverable losses
  - **High:** high financial loss
  - **Extra High:** risk to human life
  - **Low:** DB bytes/ Pgm SLOC<10
  - **Nominal:** 10sD/P<100
  - **High:** 100sD/P<1000
  - **Extra High:** D/P<1000
  - **Low:** none
  - **Nominal:** across project
  - **High:** across program
  - **Extra High:** across product line
  - **Low:** many lifecycle needs uncovered
  - **Nominal:** right-sized to lifecycle needs
  - **High:** excessive for lifecycle needs
  - **Extra High:** very excessive for lifecycle needs

- **Details:**
  - **Recoverability:** DB bytes/ Pgm SLOC<10, low, easily recoverable losses
  - **Losses:** 10sD/P<100, moderate, easily recoverable losses
  - **Financial Loss:** 100sD/P<1000, high financial loss
  - **Risk:** D/P<1000, risk to human life
  - **Project Needs:** many lifecycle needs uncovered, low
  - **Lifecycle Needs:** across project, nominal
  - **Across Program:** across program, high
  - **Across Product Line:** across product line, extra high
  - **Documentation:** across multiple product lines, extra high

- **Additional Information:**
  - **Low:** low, easily recoverable losses
  - **Nominal:** moderate, easily recoverable losses
  - **High:** high financial loss
  - **Extra High:** risk to human life
  - **Low:** DB bytes/ Pgm SLOC<10
  - **Nominal:** 10sD/P<100
  - **High:** 100sD/P<1000
  - **Extra High:** D/P<1000
  - **Low:** none
  - **Nominal:** across project
  - **High:** across program
  - **Extra High:** across product line
  - **Low:** many lifecycle needs uncovered
  - **Nominal:** right-sized to lifecycle needs
  - **High:** excessive for lifecycle needs
  - **Extra High:** very excessive for lifecycle needs

- **Values:**
  - **Low:** low, easily recoverable losses
  - **Nominal:** moderate, easily recoverable losses
  - **High:** high financial loss
  - **Extra High:** risk to human life
  - **Low:** DB bytes/ Pgm SLOC<10
  - **Nominal:** 10sD/P<100
  - **High:** 100sD/P<1000
  - **Extra High:** D/P<1000
  - **Low:** none
  - **Nominal:** across project
  - **High:** across program
  - **Extra High:** across product line
  - **Low:** many lifecycle needs uncovered
  - **Nominal:** right-sized to lifecycle needs
  - **High:** excessive for lifecycle needs
  - **Extra High:** very excessive for lifecycle needs
## Post-Architecture Complexity:

<table>
<thead>
<tr>
<th>it0 Processing</th>
<th>Control Operations</th>
<th>Computation al Operations</th>
<th>Device-dependent Operations</th>
<th>Data Management Operations</th>
<th>User Interface Management Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Low</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
</tr>
<tr>
<td>Low</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
</tr>
<tr>
<td>Nominal</td>
<td>Mostly simple nest- ing, Some intermod- ule control, Decision tables, Simple call- backs or message passing, including middleware-supported distributed processing.</td>
<td>Use of standard math and statistical routines, Basic matrix/vector operations.</td>
<td>I/O processing includes device selection, status checking and error processing.</td>
<td>Multi-file input and single file output. Simple structural changes, simple edits. Complex COTS-DB queries, updates.</td>
<td>Simple use of widget set.</td>
</tr>
<tr>
<td>High</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
</tr>
<tr>
<td>Very High</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
</tr>
<tr>
<td>Extra High</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
<td>•••</td>
</tr>
</tbody>
</table>
### Post-Architecture EMs - Platform:

<table>
<thead>
<tr>
<th></th>
<th>Very Low</th>
<th>Low</th>
<th>Normal</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execution Time</strong></td>
<td></td>
<td></td>
<td></td>
<td>70%</td>
<td>85%</td>
<td>95%</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(TIME)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Main Storage</strong></td>
<td></td>
<td></td>
<td></td>
<td>70%</td>
<td>85%</td>
<td>95%</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(STOR)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Platform Volatility</strong></td>
<td>major change every 12 mo.; minor change every 1 mo.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PVOL</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- EMS - Platform: 150% use of available time and 70% available storage.
- Main storage constraint: 6 mo. major and 2 wk. minor.
- Volatility: 2 mo. major and 1 wk. minor.
- Platform volatility: 2 wk. major and 2 days minor.
### Post-Architecture EMs - Personnel:

<table>
<thead>
<tr>
<th></th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyst Capability (ACAP)</td>
<td>15th percentile</td>
<td>35th percentile</td>
<td>55th percentile</td>
<td>75th percentile</td>
<td>90th percentile</td>
<td></td>
</tr>
<tr>
<td>Programmer Capability (PCAP)</td>
<td>15th percentile</td>
<td>35th percentile</td>
<td>55th percentile</td>
<td>75th percentile</td>
<td>90th percentile</td>
<td></td>
</tr>
<tr>
<td>Personnel Continuity (PCON)</td>
<td>48%/year</td>
<td>24%/year</td>
<td>12%/year</td>
<td>6%/year</td>
<td>3%/year</td>
<td></td>
</tr>
<tr>
<td>Application Experience (AEXP)</td>
<td>≤ 2 months</td>
<td>6 months</td>
<td>1 year</td>
<td>3 years</td>
<td>6 years</td>
<td></td>
</tr>
<tr>
<td>Platform Experience (PEXP)</td>
<td>≤ 2 months</td>
<td>6 months</td>
<td>1 year</td>
<td>3 years</td>
<td>6 years</td>
<td></td>
</tr>
<tr>
<td>Language and Tool Experience (LTEX)</td>
<td>≤ 2 months</td>
<td>6 months</td>
<td>1 year</td>
<td>3 years</td>
<td>6 years</td>
<td></td>
</tr>
</tbody>
</table>
### Post-Architecture EMs - Project:

<table>
<thead>
<tr>
<th></th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use of Software Tools (TOOL)</strong></td>
<td>edit, code, debug</td>
<td>simple, frontend, backend CASE, little integration</td>
<td>basic lifecycle tools, moderately integrated</td>
<td>strong, mature lifecycle tools, moderately integrated</td>
<td>strong, mature, proactive lifecycle tools, well integrated with processes, methods, reuse</td>
<td></td>
</tr>
<tr>
<td><strong>Multisite Development: Collocation (SITE)</strong></td>
<td>International</td>
<td>Multi-city and Multi-company</td>
<td>Multi-city or Multi-company</td>
<td>Same city or metro. area</td>
<td>Same building or complex</td>
<td>Fully collocated</td>
</tr>
<tr>
<td><strong>Multisite Development: Communications (SITE)</strong></td>
<td>Some phone, mail</td>
<td>Individual phone, FAX</td>
<td>Narrowband email</td>
<td>Wideband electronic communication</td>
<td>Wideband electronic, comm., occasional video conf.</td>
<td>Interactive multimedia</td>
</tr>
<tr>
<td><strong>Required Development Schedule (SCED)</strong></td>
<td>75% of nominal</td>
<td>85%</td>
<td>100%</td>
<td>130%</td>
<td>160%</td>
<td></td>
</tr>
</tbody>
</table>

*Note: The table compares the use of software tools, multisite development, and required development schedules across different categories.*
# COCOMO Model Comparisons

<table>
<thead>
<tr>
<th></th>
<th>COCOMO</th>
<th>Ada COCOMO</th>
<th>COCOMO 2.0: Application Composition</th>
<th>COCOMO 2.0: Early Design</th>
<th>COCOMO 2.0: Post-Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size</strong></td>
<td>Delivered Source Instructions (DSI) or Source Lines Of Code (SLOC)</td>
<td>DSI or SLOC</td>
<td>Object Points</td>
<td>Function Points (FP) and Language</td>
<td>FP and Language or SLOC</td>
</tr>
<tr>
<td><strong>Breakage</strong></td>
<td>Requirements Volatility rating: (RVOL)</td>
<td>RVOL rating</td>
<td>Implicit in Model</td>
<td>Breakage %: BRAK</td>
<td>BRAK</td>
</tr>
<tr>
<td><strong>Maintenance</strong></td>
<td>Annual Change Traffic (ACT) = %added + %modified</td>
<td>ACT</td>
<td>ACT</td>
<td>Object Point ACT</td>
<td>Reuse model</td>
</tr>
<tr>
<td><strong>Scale (b) in MMNOM = a(Size)^b</strong></td>
<td>Organic: 1.05 Semidetached: 1.12 Embedded: 1.20</td>
<td>Embedded: 1.04 - 1.24 depending on degree of: early risk elimination solid architecture stable requirements Ada process maturity</td>
<td>1.0</td>
<td>1.02-1.26 depending on the degree of: unprecedentedness conformity early architecture, risk resolution team cohesion process maturity (SEI)</td>
<td>1.02-1.26 depending on the degree of: unprecedentedness conformity early architecture, risk resolution team cohesion process maturity (SEI)</td>
</tr>
<tr>
<td><strong>Product Cost Drivers</strong></td>
<td>RELY, DATA, CPLX</td>
<td>RELY, DATA, CPLX, RUSE</td>
<td>None</td>
<td>ACAP, AEXP, PCAP, VEXP, LEXP</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>TIME, STOR, VIRT, TURN</td>
<td>TIME, STOR, VMVII, VMVT, TURN</td>
<td>None</td>
<td>TIME, STOR, VMVII, VMVT, TURN</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>ACAP, AEXP, PCAP, VEXP, LEXP</td>
<td>ACAP, AEXP, PCAP, VEXP, LEXP</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Project Cost Drivers</td>
<td>MODP, TOOL, SCED</td>
<td>MODP, TOOL, SCED, SEDU</td>
<td>None</td>
<td>MODP, TOOL, SCED, SEDU</td>
<td>None</td>
</tr>
</tbody>
</table>

- Different multipliers
- Different rating scale
## Early Design vs. Post-Arch EMs:

<table>
<thead>
<tr>
<th>Early Design</th>
<th>Counterpart Architecture Drivers</th>
<th>Combined Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product, Reliability and Complexity</td>
<td>RELY, DATA, CPLX, DOCU</td>
<td></td>
</tr>
<tr>
<td>Required Reuse</td>
<td>RUSE</td>
<td></td>
</tr>
<tr>
<td>Platform Difficulty</td>
<td>TIME, STOR, PVOL</td>
<td></td>
</tr>
<tr>
<td>Personnel Capability</td>
<td>ACAP, FCAP, PCON</td>
<td></td>
</tr>
<tr>
<td>Personnel Experience</td>
<td>AEAP, PEXP, TEXP</td>
<td></td>
</tr>
<tr>
<td>Facilities</td>
<td>TOOL, SITE</td>
<td></td>
</tr>
<tr>
<td>Schedule</td>
<td>SCED</td>
<td></td>
</tr>
</tbody>
</table>
Other Model Refinements:
- Initial Schedule Estimation

\[ TDEV = \left[ 3.0 \times \left( \frac{\overline{PM}}{100} \right)^{0.33 + 0.2 \times (B-1.01)} \right] \times \frac{SCED\%}{100} \]

where \( \overline{PM} \) = estimated person months excluding Schedule multiplier effects

- Output Ranges

<table>
<thead>
<tr>
<th>Stage</th>
<th>Optimistic Estimate</th>
<th>Pessimistic Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Composition</td>
<td>0.50 E</td>
<td>2.0 E</td>
</tr>
<tr>
<td>Early Design</td>
<td>0.67 E</td>
<td>1.5 E</td>
</tr>
<tr>
<td>Post-Architecture</td>
<td>0.80 E</td>
<td>1.25 E</td>
</tr>
</tbody>
</table>

- One standard deviation optimistic and pessimistic estimates
- Reflect sources of uncertainty in model inputs
COCOMO 2.0 Model Overview

- COCOMO Baseline Overview
- COCOMO 2.0 Objectives
- Coverage of Future Market Sectors
- Hierarchical Sizing Model
- Modes Replaced by Exponent Drivers
- Stronger Reuse/Reengineering Model
- Other Model Improvements

Activity Sequence
COCOMO 2.0 Activity Sequence

- Cost model review
- Hypothesis formulation (prepared, iterated)
- Data definitions (prepared, iterated)
- Monthly electronic data collection
  - Amadeus data collection support
- Data archiving
- Incremental model development and refinement
  - New USC COCOMO (baseline version prepared)
- Related research
  - COTS integration cost/risk estimation
Data Collection

- Quantitative collection
  - Project sizing
  - Software metrics

- Qualitative collection
  - Scale Factors
  - Effort Multipliers
Amadeus System

- Automated system for software
  - metric collection
  - metric analysis
  - metric reporting, graphing
  - metric prediction
- Open architecture for extensibility
- Supported metrics
Preliminary Analysis

- Calibration to original COCOMO project database
  - Trend analysis is positive
  - Coefficient, A, calibrated
- Correlation between Cost Drivers
- Continuing analysis from other sources
Results of COCOMO 81 Database Calibration
Outline

- Steps in Software Cost Estimation
  - 10 years of TRW experience
- Integrated Estimation, Measurement, and Management
  - Getting to CMM levels 4 and 5
  - Role of COCOMO 2.0
- COCOMO 2.0 Model Overview
- COCOMO 2.0 Status and Plans
- Information Sources
COCOMO 2.0 Status and Plans

- Current Status
  - Affiliates’ program
- Future plans
  - Public and commercial versions
  - Model extensions
- Long term vision
COCOMO 2.0 Current Status

- Model structure iterated with Affiliates
  - Largely converged
  - Recent function point backfiring refinements being considered
- Model being calibrated to Affiliates’, other data
  - Currently around 65 solid data points
- Model being beta-tested by Affiliates
  - Reasonable results so far
Error Before Regression

Std. Dev = .81
Mean = .22
N = 65.00
Error before stratification:

![Histogram of error before stratification with mean 0 and std dev 0.40, N = 65.00]
Error after stratification:

\[ \text{Std. Dev} = .37 \]
\[ \text{Mean} = .00 \]
\[ N = 65.00 \]
Current COCOMO 2.0 Affiliates (27)

- Commercial Industry (9)
  - AT&T, Bellcore, CSC, EDS, Motorola, Rational, Sun, TI, Xerox
- Aerospace Industry (10)
  - E-Systems, Hughes, Litton, Lockheed Martin, Loral, MDAC, Northrop Grumman, Rockwell, SAIC, TRW
- Government (3)
  - AFCAA, USAF Rome Lab, US Army Research Labs
- FFRDC's and Consortia (5)
  - Aerospace, IDA, MCC, SEI, SPC
COCOMO 2.0 Affiliates' Program

- funding support
- sanitized data
- feedback on models, tools
- special project definition, support
- visiting researchers

Affiliates, Research Sponsors

Workshops, joint projects

-COCOMO 2.0 models, tools
-Amadeus metrics package
-Tutorials, reports, related research
-Special project results
-Graduating students

COCOMO 2.0 Project
-SEI, Others
-Metric, Process Definitions

-USC-CSE
-USC-IOM
-UCI-ICS

COCOMO 2.0 Database

Sponsors - USC-CSE, USC-IOM, UCI-ICS

Project - Research Workshops, joint projects

Definitions - COCOMO 2.0 models, tools

- Amadeus metrics package
- Tutorials, reports, related research
- Special project results
- Graduating students

Data
COCOMO 2.0: Public and Commercial Versions

- Available once Affiliates’ calibration and beta-testing stabilizes
  - Final beta-test version by COCOMO/SCM Forum
  - Full public version January 1997
- Public version: USC COCOMO 2.0 tool
  - will do basic estimation and calibration functions
  - won’t include extensive amenities, planning aids
- Commercial COCOMO vendors being provided with pre-release COCOMO 2.0 parameters, source code
COCOMO 2.0 Plans: Model Extensions

- COTS integration costs
- Cost/schedule/quality tradeoffs
- Activity distribution
- Domain-tailored models
- Specialized life-cycle stage estimation
  - Hardware-software integration, operational testing
- Sizing improvements
- Project dynamics modeling
- Reuse, tool, process return-on-investment models
Long Term Vision:

- Produce model covering trade-offs among software cost, schedule, functionality, performance, quality
- Enable the use of the model to scope projects and monitor project progress
- Use the data from COCOMO-assisted management to calibrate improved versions of COCOMO
- Use the data and improved COCOMO cost drivers to enable affiliates to formulate and manage software improvement investments.
Information Sources:

- E-Mail: cocomo-info@sunset.usc.edu
- Anonymous FTP: usc.edu
  in directory: /pub/soft_engineering/cocomo2
- WWW:
  - http://sunset.usc.edu/COCOMO2.0/
    Cocomo.html
  - Upcoming COCOMO events, Research Group
  - COCOMO 81 calculator
  - COCOMO 2.0 User’s Manual