Contents

- Software cost model
- COCOMO and COCOMO II
- Constraint model
- Software lifecycle management (SLIM)
- Cost models: advantages and drawbacks

Software cost

Cost model
- Effort vs. cost drive (size)

Constraint model
- Effort & cost vs. time
Software Cost Estimation

- Software cost estimation is the process of predicting the amount of effort required to build a software system and time to develop it.

- Models provide mathematical algorithms to compute cost as a function of a number of variables such as size (using lines of code, function points, etc.) and/or complexity (using cyclomatic complexity, etc.).

- Models used to estimate cost can be categorized as:
  - **Cost model**; *e.g.*, COCOMO
  - **Constraint model**; *e.g.*, SLIM

- Most of the models are available as automated tools.
Cost Estimation: 7 Techniques

Cost/Schedule Estimation Techniques

- Algorithmic Models
  - Cost Models: COCOMO / COCOMO II, ESTIMACS, PRICE S, Softcost, DOTY, CheckPoint, etc.
  - Constraint Models: SLIM, Jensen Model, COPMO, etc.
- Expert Judgment
- Analogy
- Machine Learning:
  - Case-Based Reasoning
  - Collaborative Filtering
  - Classification Systems
  - etc.
- Other
  - Parkinson’s Law
  - Pricing-to-Win
  - Top-Down Estimation
  - Bottom-Up Estimation

Source: Boehm (1981)
Bottom-Up Estimation:

- Start at the lowest system level. The cost/schedule of each component is estimated. All these costs are added to produce a final cost estimate.

**Advantage:**
- Can be accurate, if the system has been designed in detail

**Disadvantage:**
- May underestimate costs of system level activities such as integration and documentation
Top-Down Estimation:

- A cost/schedule estimate is established by considering the overall functionality of the product and how that functionality is provided by interacting sub-functions. Cost estimates are made on the basis of the logical function rather than the components implementing that function.
  - **Advantage:** Takes into account costs such as integration, configuration management and documentation
  - **Disadvantage:** Can underestimate the cost of solving difficult low-level technical problems
Pricing-to-Win

- Software cost/schedule is estimated to be whatever the customer has available to spend on the project. The estimated effort depends on the customer's budget and not on the software functionality.
- **Advantage:** Good chances to win the contract!
- **Disadvantages:** The probability that costs accurately reflect the work required and the probability that the customer gets the system he or she wants are low.
Parkinson's Law

- Parkinson's Law states that work expands to fill the time available. This means that cost is determined by available resources rather than objective assessment.
- Example: If the software has to be delivered in 12 months and 5 people are available, the effort required is estimated to be 60 person-months.

- **Advantage:** No overspending
- **Disadvantage:** System is usually unfinished / or effort is wasted
Expert Judgement

- One or more experts in both software development and the application domain use their experience to predict software costs. Process iterates until some consensus is reached (e.g., using Delphi Method).

- **Advantage:** Relatively cheap estimation method. Can be accurate if experts have substantial experience with similar systems.

- **Disadvantage:** Very inaccurate if there are no experts! Difficult to trace back, in case the estimate was not accurate.
Cost/Schedule Estimation Techniques /6

Estimation by Analogy

- This technique is applicable when other projects in the same application domain have been done in the past. The cost of a new project is computed by comparing the project to a similar completed project in the same application domain.

- **Advantages:** Accurate if similar projects are available.

- **Disadvantages:** Impossible if no comparable project available. Needs systematically maintained project database (→ expensive). It’s not always clear how to define a good similarity function.
### Case-Based Reasoning (CBR) Example:

<table>
<thead>
<tr>
<th>Attributes</th>
<th>New Case</th>
<th>Retrieved Case 1</th>
<th>Retrieved Case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Category</td>
<td>Real Time</td>
<td>Real Time</td>
<td>Simulator</td>
</tr>
<tr>
<td>Language</td>
<td>C++</td>
<td>C++</td>
<td>C++</td>
</tr>
<tr>
<td>Team Size</td>
<td>10</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>System Size</td>
<td>150</td>
<td>200</td>
<td>175</td>
</tr>
<tr>
<td>Effort</td>
<td>?</td>
<td>1000</td>
<td>950</td>
</tr>
<tr>
<td>Similarity</td>
<td></td>
<td>90%</td>
<td>~50%</td>
</tr>
</tbody>
</table>

**Possible adaptation rule:**
\[
Effort = f(System\_Size) = \frac{150}{200} \times 1000 = 750
\]

**Possibilities to predict effort:**
- adapted effort based on 1 project
- average effort of 2 projects
- weighted average effort of 2 projects

`far@ucalgary.ca`
Case-Based Reasoning (CBR) Example:

<table>
<thead>
<tr>
<th>Attributes</th>
<th>New Case</th>
<th>Retrieved Case 1</th>
<th>Retrieved Case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Category</td>
<td>Real Time</td>
<td>Real Time</td>
<td>Simulator</td>
</tr>
<tr>
<td>Language</td>
<td>C++</td>
<td>C++</td>
<td>C++</td>
</tr>
<tr>
<td>Team Size</td>
<td>10</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>System Size</td>
<td>150</td>
<td>200</td>
<td>175</td>
</tr>
<tr>
<td>Effort</td>
<td>?</td>
<td>1000</td>
<td>950</td>
</tr>
<tr>
<td>Similarity</td>
<td></td>
<td>90%</td>
<td>~50%</td>
</tr>
</tbody>
</table>

Possible adaptations rule:

\[
\text{Effort} = f(\text{System Size})
\]

Possibilities to predict effort:
- adapted effort based on 1 project
- average effort of 2 projects
- weighted average effort of 2 projects

\[
\text{Predicted Effort} = \frac{1}{2} \left( \frac{150}{200} \times 1000 + \frac{150}{175} \times 950 \right) \approx 782
\]
### Case-Based Reasoning (CBR) Example:

<table>
<thead>
<tr>
<th>Attributes</th>
<th>New Case</th>
<th>Retrieved Case 1</th>
<th>Retrieved Case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Category</td>
<td>Real Time</td>
<td>Real Time</td>
<td>Simulator</td>
</tr>
<tr>
<td>Language</td>
<td>C++</td>
<td>C++</td>
<td>C++</td>
</tr>
<tr>
<td>Team Size</td>
<td>10</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>System Size</td>
<td>150</td>
<td>200</td>
<td>175</td>
</tr>
<tr>
<td>Effort</td>
<td>?</td>
<td>1000</td>
<td>950</td>
</tr>
<tr>
<td>Similarity</td>
<td></td>
<td>90%</td>
<td>(~50%)</td>
</tr>
</tbody>
</table>

**Possibilities to predict effort:**
- adapted effort based on 1 project
- average effort of 2 projects
- weighted average effort of 2 projects

**Possible adaptation rule:**

\[
\text{Predicted \_ Effort} = \frac{150}{200} \times 1000 \times \frac{9}{14} + \frac{150}{175} \times 950 \times \frac{5}{14} \approx 773
\]
Algorithmic Models

- Uses historical cost information and relates it to some software project measure(s).

- Two main categories (Kitchenham, 1991):
  - **Empirical Factor Models (Cost Models):** Provide estimates of project cost (effort) and schedule based on size; derived from empirical data.
  - **Constraint Models:** Assume a more complex relationship between cost (effort), schedule and size, i.e., assume that productivity is not only a function of effort (input) and size (output) but that it is “constrained” by the (anticipated) schedule and/or complexity.
Cost Models

- Cost models provide direct estimates of **effort**.
- Cost models typically are based on:
  - a primary cost factor such as **size**
  - a number of secondary adjustment factors or **cost drivers**
- Cost drivers are characteristics of the project, process, products, or resources that influence effort.
- Cost drivers are used to adjust the preliminary estimate provided by the primary cost factor.
Cost Models /2

Effort is plotted against the primary cost factor (usually LOC or FP) for a series of projects.

The line of best fit is calculated for the data points. If the primary cost factor were a perfect predictor of effort, then every point on the graph would lie on the line of best fit.

In reality, there is usually significant residual error. Therefore necessary to identify the factors that cause variation between predicted and actual effort. These parameters are added to the model as cost drivers.
Example
The regression-based cost models take the form:

\[ E = a + b \times S^c \]

where

- \( E \) is the effort in person months (PM)
- \( a, b, \) and \( c \) are empirically derived constants
- \( S \) is the primary cost factor (typically, either LOC or FP)
The following are some examples of cost models using LOC as a primary cost factor (Pressman, 1997):

\[
E = 5.2 \times (KLOC)^{0.91} \quad \text{Walston-Felix Model}
\]
\[
E = 5.5 + 0.73 \times (KLOC)^{1.16} \quad \text{Bailey-Basili Model}
\]
\[
E = 2.4 \times (KLOC)^{1.05} \quad \text{COCOMO Basic Model}
\]
\[
E = 5.288 \times (KLOC)^{1.047} \quad \text{Doty Model for KLOC > 9}
\]
Cost Models /5

- The following are some examples of cost models using FP as a primary cost factor (Pressman, 1997):

\[ E = -13.39 + 0.0545 \times FP \]
\[ E = 60.62 \times 7.728 \times 10^{-8} \times FP^3 \]
\[ E = 585.7 + 15.12 \times FP \]

Albrecht and Gaffney Model
Kremerer Model
Matson, Barnett, & Mellichamp Model
Constraint Models

- Constraint models demonstrate the relationship over time between two or more parameters of *effort, duration, or staffing level* (Fenton, 1997).

- Putnam’s SLIM model is an example of constraint model: it adds time dimension to effort vs. size graph.

Constructive Cost Model

COCOMO

Prof. Barry Boehm
COCOMO (Constructive Cost Model) (Boehm, 1981)

COCOMO is a model based on inputs relating to the size of the system and a number of cost drivers that affect productivity.

Updated version, called COCOMO II, accounts for recent changes in software engineering technology, including object-oriented software, software created via spiral or evolutionary development models, software reuse and building new systems using off-the-shelf software components.
The original COCOMO is a collection of three models:

- **Basic model** that is applied early in the project
- **Intermediate model** that is applied after requirements acquisition
- **Advanced model** that is applied after design is complete

All three models take the form:

\[ E = aS^b \times EAF \]

\[ T_{dev} = cE^d \]

where

- \( E \) is effort in person months
- \( T_{dev} \) is the development time
- \( S \) is size measured in KLOC
- \( EAF \) is an effort adjustment factor (1 in the Basic model)
- Factors \( a, b, c \) and \( d \) depend on the **development mode**.
COCOMO has three development modes:

- **Organic mode**: small teams, familiar environment, well-understood applications, no difficult non-functional requirements (EASY)

- **Semi-detached mode**: project team may have experience mixture, system may have more significant non-functional constraints, organization may have less familiarity with application (HARDER)

- **Embedded mode**: hardware/software systems, tight constraints, unusual for team to have deep application experience (HARD)
COCOMO: Basic

The Basic COCOMO model computes effort (person-month) as a function of program size. The Basic COCOMO equation is:

\[ E_{\text{nom}} = a \left( KLOC \right)^b \]

\[ EAF = 1 \]

The factors \( a \) and \( b \) for the Basic model are shown in the table.

\( T_{\text{dev}} \) is not defined

<table>
<thead>
<tr>
<th>Mode</th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organic</td>
<td>2.4</td>
<td>1.05</td>
</tr>
<tr>
<td>Semi-detached</td>
<td>3.0</td>
<td>1.12</td>
</tr>
<tr>
<td>Embedded</td>
<td>3.6</td>
<td>1.20</td>
</tr>
</tbody>
</table>

**Organic:**
PM = 2.4 \((\text{KDSI})^{1.05}\)

**Semi-detached:**
PM = 3.0 \((\text{KDSI})^{1.12}\)

**Embedded:**
PM = 3.6 \((\text{KDSI})^{1.20}\)
Basic COCOMO: Examples

- **Organic mode project:** 32 KLOC
  - \( PM = 2.4 \times (32)^{1.05} = 91 \) person-months
  - Development time: don’t know

- **Embedded mode project:** 128 KLOC
  - \( PM = 3.6 \times (128)^{1.2} = 1216 \) person-months
  - Development time: don’t know
The Intermediate COCOMO model computes effort as a function of program size and a set of cost drivers. The intermediate COCOMO equation is:

\[ E = a \left( KLOC \right)^b \times EAF \]

\[ T_{dev} = c E_{nom}^d \]

The factors \( a, b, c \) and \( d \) for the intermediate model are shown in the table.

<table>
<thead>
<tr>
<th>Mode</th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organic</td>
<td>3.2</td>
<td>1.05</td>
</tr>
<tr>
<td>Semi-detached</td>
<td>3.0</td>
<td>1.12</td>
</tr>
<tr>
<td>Embedded</td>
<td>2.8</td>
<td>1.20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mode</th>
<th>( c )</th>
<th>( d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organic</td>
<td>2.5</td>
<td>0.38</td>
</tr>
<tr>
<td>Semi-detached</td>
<td>2.5</td>
<td>0.35</td>
</tr>
<tr>
<td>Embedded</td>
<td>2.5</td>
<td>0.32</td>
</tr>
</tbody>
</table>

The effort adjustment factor (\( EAF \)) is calculated using 15 cost drivers.
The effort adjustment factor (EAF) is calculated using 15 cost drivers. Each cost driver is rated on a 6 point ordinal scale ranging from very low to extra high importance. Based on the rating, the effort multiplier (EM) is determined (Boehm, 1981). The product of all effort multipliers is the EAF.

\[ E = E_{nom} \times EAF \quad EAF = \prod_{i=1}^{15} EM_i \]
COCOMO Intermediate /3

15 Cost Drivers

- **Product Factors**
  - Reliability (RELY)
  - Data (DATA)
  - Complexity (CPLX)

- **Platform Factors**
  - Time constraint (TIME)
  - Storage constraint (STOR)
  - Platform volatility (PVOL)

- **Personnel factors**
  - Analyst capability (ACAP)
  - Program capability (PCAP)
  - Applications experience (APEX)
  - Platform experience (PLEX)
  - Language and tool experience (LTEX)
  - Personnel continuity (PCON)

- **Project Factors**
  - Software tools (TOOL)
  - Multisite development (SITE)
  - Required schedule (SCED)
## Cost Driver Rating /1

### 1. Product

<table>
<thead>
<tr>
<th>Description</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>RELY Required software reliability</td>
<td>0.75</td>
<td>0.88</td>
<td>1.00</td>
<td>1.15</td>
<td>1.40</td>
<td>-</td>
</tr>
<tr>
<td>DATA Database size</td>
<td>-</td>
<td>0.94</td>
<td>1.00</td>
<td>1.08</td>
<td>1.16</td>
<td>-</td>
</tr>
<tr>
<td>CPLX Product complexity</td>
<td>0.70</td>
<td>0.85</td>
<td>1.00</td>
<td>1.15</td>
<td>1.30</td>
<td>1.65</td>
</tr>
</tbody>
</table>
## Cost Driver Rating /2

### 2. Platform

<table>
<thead>
<tr>
<th>Description</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>TIME</strong> Execution time constraint</td>
<td>-</td>
<td>-</td>
<td>1.00</td>
<td>1.11</td>
<td>1.30</td>
<td>1.66</td>
</tr>
<tr>
<td><strong>STOR</strong> Main storage constraint</td>
<td>-</td>
<td>-</td>
<td>1.00</td>
<td>1.06</td>
<td>1.21</td>
<td>1.56</td>
</tr>
<tr>
<td><strong>VIRT</strong> Virtual machine volatility</td>
<td>-</td>
<td>0.87</td>
<td>1.00</td>
<td>1.15</td>
<td>1.30</td>
<td>-</td>
</tr>
<tr>
<td><strong>TURN</strong> Computer turnaround time</td>
<td>-</td>
<td>0.87</td>
<td>1.00</td>
<td>1.07</td>
<td>1.15</td>
<td>-</td>
</tr>
</tbody>
</table>
## Cost Driver Rating /3

### 3. Personnel

<table>
<thead>
<tr>
<th>Description</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>ACAP Analyst capability</td>
<td>1.46</td>
<td>1.19</td>
<td>1.00</td>
<td>0.86</td>
<td>0.71</td>
<td>-</td>
</tr>
<tr>
<td>AEXP Applications experience</td>
<td>1.29</td>
<td>1.13</td>
<td>1.00</td>
<td>0.91</td>
<td>0.82</td>
<td>-</td>
</tr>
<tr>
<td>PCAP Programmer capability</td>
<td>1.42</td>
<td>1.17</td>
<td>1.00</td>
<td>0.86</td>
<td>0.70</td>
<td>-</td>
</tr>
<tr>
<td>VEXP Virtual machine experience</td>
<td>1.21</td>
<td>1.10</td>
<td>1.00</td>
<td>0.90</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LEXP Language experience</td>
<td>1.14</td>
<td>1.07</td>
<td>1.00</td>
<td>0.95</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
# Cost Driver Rating /4

## 4. Project

<table>
<thead>
<tr>
<th>Description</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>MODP - Modern programming practices</td>
<td>1.24</td>
<td>1.10</td>
<td>1.00</td>
<td>0.91</td>
<td>0.82</td>
<td>-</td>
</tr>
<tr>
<td>TOOL - Software Tools</td>
<td>1.24</td>
<td>1.10</td>
<td>1.00</td>
<td>0.91</td>
<td>0.83</td>
<td>-</td>
</tr>
<tr>
<td>SCED - Development Schedule</td>
<td>1.23</td>
<td>1.08</td>
<td>1.00</td>
<td>1.04</td>
<td>1.10</td>
<td>-</td>
</tr>
</tbody>
</table>
Cost Driver Rating: Example

- **Example: Required Software Reliability (RELY)**
  - Measures the extent to which the software must perform its intended function over a period of time.
  - Ask: What is the *effect of a software failure*?

<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>RELY</td>
<td>slight</td>
<td>low, easily</td>
<td>moderate,</td>
<td>high financial</td>
<td>risk to human life</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inconvenience</td>
<td>recoverable losses</td>
<td>easily recoverable losses</td>
<td>loss</td>
<td>life</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.75</td>
<td>0.88</td>
<td>1.00</td>
<td>1.15</td>
<td>1.39</td>
<td></td>
</tr>
</tbody>
</table>
Cost Driver Rating: Example

- Effort Multiplier values for Reliability

- E.g., a highly reliable system costs 39% more than a nominally reliable system \((1.39/1.0 = 1.39)\)

- A highly reliable system costs 85% more than a very low reliability system \((1.39/0.75 = 1.85)\)

From [Boe81]
Cost Driver Rating /5

Cost Driver Impact [Boe81]
COCOMO provides a framework to identify high leverage productivity improvement factors and estimate their payoffs.

From [Boe81]

RELY

far@ucalgary.ca
Advanced COCOMO model computes effort as a function of program size and a set of cost drivers *weighted according to each phase of software lifecycle*. Advanced model applies the intermediate model at the component level, and then a phase-based approach is used to consolidate the estimate.
The 4 phases used in the detailed COCOMO model are: requirements planning and product design (RPD), detailed design (DD), code and unit test (CUT), and integration and test (IT).

Each cost driver is broken down by phase as in the table below.

Estimates made for each module are combined into subsystems and eventually an overall project estimate.

<table>
<thead>
<tr>
<th>Cost Driver</th>
<th>Rating</th>
<th>RPD</th>
<th>DD</th>
<th>CUT</th>
<th>IT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Analyst capability)</td>
<td>Very Low</td>
<td>1.80</td>
<td>1.35</td>
<td>1.35</td>
<td>1.50</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>0.85</td>
<td>0.85</td>
<td>0.85</td>
<td>1.20</td>
</tr>
<tr>
<td></td>
<td>Nominal</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>High</td>
<td>0.75</td>
<td>0.90</td>
<td>0.90</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>Very High</td>
<td>0.55</td>
<td>0.75</td>
<td>0.75</td>
<td>0.70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACAP Intermediate</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.46 – Very Low</td>
</tr>
<tr>
<td></td>
<td>1.19 – Low</td>
</tr>
<tr>
<td></td>
<td>1.00 – Nominal</td>
</tr>
<tr>
<td></td>
<td>0.86 – High</td>
</tr>
<tr>
<td></td>
<td>0.71 – Very High</td>
</tr>
</tbody>
</table>
COCOMO II

Constructive Cost Model II
COCOMO II includes three-stage series of models:

- The earliest phases will generally involve prototyping, using the *Application Composition* model capabilities. (replaces Basic model in COCOMO)

- The next phases will generally involve exploration of architectural alternatives or incremental development strategies. To support these activities, COCOMO II provides an early estimation model called the *Early Design model*. (replaces Intermediate model in COCOMO)

- Once the project is ready to develop and sustain a fielded system, it should have a life-cycle architecture, which provides more accurate information on cost driver inputs, and enables more accurate cost estimates. To support this stage, COCOMO II provides the *Post-Architecture model*. (replaces Advanced model in COCOMO)
1. Application Composition Model

- The Application Composition model is used in early development stages and prototyping to resolve potential high-risk issues such as user interfaces, software/system interaction, performance, or technology maturity.

- **Object points** are used for sizing rather than the traditional LOC metric.

\[
E = \frac{OP}{PROD}
\]

where

- \(OP\) is the object point
- \(PROD\) is the productivity rate

<table>
<thead>
<tr>
<th>Developers’ 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>PROD</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

far@ucalgary.ca
2. Early Design Model

- The Early Design model is used to evaluate alternative software/system architectures and concepts of operation. Unadjusted function point count (UFC) is used for sizing. This value is converted to KLOC.

- The Early Design model’s equation is:
  \[ E = 2.45 \times KLOC \times EAF \]

- The effort adjustment factor (EAF) is calculated as in the original COCOMO model using 7 cost drivers.
The effort adjustment factor (EAF) for the Early Design model (obtained by combining the 17 Post-Architecture cost drivers)

<table>
<thead>
<tr>
<th>Cost Driver</th>
<th>Description</th>
<th>Combined Post-Architecture Cost Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 RCPX</td>
<td>Product reliability and complexity</td>
<td>RELY, DATA, CPLX, DOCU</td>
</tr>
<tr>
<td>2 RUSE</td>
<td>Required reuse</td>
<td>RUSE</td>
</tr>
<tr>
<td>3 PDIF</td>
<td>Platform difficulty</td>
<td>TIME, STOR, PVOL</td>
</tr>
<tr>
<td>4 PERS</td>
<td>Personnel capability</td>
<td>ACAP, PCAP, PCON</td>
</tr>
<tr>
<td>5 PREX</td>
<td>Personnel experience</td>
<td>AEXP, PEXP, LTEX</td>
</tr>
<tr>
<td>6 FCIL</td>
<td>Facilities</td>
<td>TOOL, SITE</td>
</tr>
<tr>
<td>7 SCED</td>
<td>Schedule</td>
<td>SCED</td>
</tr>
</tbody>
</table>
3. Post-Architecture Model

- The Post-Architecture model is used during the actual development and maintenance of a product.
- Function point (FP) or LOC can be used for sizing, with modifiers for reuse and software breakage.
- The Post-Architecture model includes a set of 17 cost drivers ($EAF$) and a set of 5 scale factors ($SF$) determining the projects scaling component.
- The Post-Architecture model equation is:

\[ E = 2.45 \times \left( KLOC \right)^b \times EAF \]

\[ b = 0.91 + 0.01 \sum_{j=1}^{5} SF_i \quad 0.91 \leq b \leq 1.23 \]
## Scale Factors

<table>
<thead>
<tr>
<th>Scale Factor</th>
<th>SF</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>Precedentedness</td>
<td>PREC</td>
<td>6.20</td>
<td>4.96</td>
<td>3.72</td>
<td>2.48</td>
<td>1.24</td>
<td>0.00</td>
</tr>
<tr>
<td>Degree to which system is new and past experience applies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development/Flexibility</td>
<td>FLEX</td>
<td>5.07</td>
<td>4.05</td>
<td>3.04</td>
<td>2.03</td>
<td>1.01</td>
<td>0.00</td>
</tr>
<tr>
<td>Need to conform to specified requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture/Risk Resolution</td>
<td>RESL</td>
<td>7.07</td>
<td>5.65</td>
<td>4.24</td>
<td>2.83</td>
<td>1.41</td>
<td>0.00</td>
</tr>
<tr>
<td>Degree of design thoroughness and risk elimination</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team Cohesion</td>
<td>TEAM</td>
<td>5.48</td>
<td>4.38</td>
<td>3.29</td>
<td>2.19</td>
<td>1.10</td>
<td>0.00</td>
</tr>
<tr>
<td>Need to synchronize stakeholders and minimize conflict</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Maturity</td>
<td>PMAT</td>
<td>7.80</td>
<td>6.24</td>
<td>4.68</td>
<td>3.12</td>
<td>1.56</td>
<td>0.00</td>
</tr>
<tr>
<td>SEI CMM process maturity rating</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Scale Factors /2

\[ E = 2.45 \times (KLOC)^b \times EAF \]

\[ b = 0.91 + 0.01 \sum_{j=1}^{5} SF_i \]

\[ 0 \leq PREC \leq 6.20 \]
\[ 0 \leq FLEX \leq 5.07 \]
\[ 0 \leq RESL \leq 7.07 \]
\[ 0 \leq TEAM \leq 5.48 \]
\[ 0 \leq PMAT \leq 7.80 \]

far@ucalgary.ca
Elaboration of **PREC** and **FLEX** rating scales:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Very Low</th>
<th>Nominal / High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precededness</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organizational understanding of product objectives</td>
<td>General</td>
<td>Considerable</td>
<td>Thorough</td>
</tr>
<tr>
<td>Experience in working with related software systems</td>
<td>Moderate</td>
<td>Considerable</td>
<td>Extensive</td>
</tr>
<tr>
<td>Concurrent development of associated new hardware and operational procedures</td>
<td>Extensive</td>
<td>Moderate</td>
<td>Some</td>
</tr>
<tr>
<td>Need for innovative data processing architectures, algorithms</td>
<td>Considerable</td>
<td>Some</td>
<td>Minimal</td>
</tr>
<tr>
<td><strong>Development Flexibility</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Need for software conformance with pre-established requirements</td>
<td>Full</td>
<td>Considerable</td>
<td>Basic</td>
</tr>
<tr>
<td>Need for software conformance with external interface specifications</td>
<td>Full</td>
<td>Considerable</td>
<td>Basic</td>
</tr>
<tr>
<td>Premium on early completion</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
</tbody>
</table>

far@ucalgary.ca
Elaboration of **RESL** rating scale: Use a subjective weighted average of the characteristics

<table>
<thead>
<tr>
<th>Characteristic</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>Risk Management Plan identifies all critical risk items, establishes milestones for resolving them by PDR.</td>
<td>None</td>
<td>Little</td>
<td>Some</td>
<td>Generally</td>
<td>Mostly</td>
<td>Fully</td>
</tr>
<tr>
<td>Schedule, budget, and internal milestones through PDR compatible with Risk Management Plan</td>
<td>None</td>
<td>Little</td>
<td>Some</td>
<td>Generally</td>
<td>Mostly</td>
<td>Fully</td>
</tr>
<tr>
<td>Percent of development schedule devoted to establishing architecture, given general product objectives</td>
<td>5</td>
<td>10</td>
<td>17</td>
<td>25</td>
<td>33</td>
<td>40</td>
</tr>
<tr>
<td>Percent of required top software architects available to project</td>
<td>20</td>
<td>40</td>
<td>60</td>
<td>80</td>
<td>100</td>
<td>120</td>
</tr>
<tr>
<td>Tool support available for resolving risk items, developing and verifying architectural specs</td>
<td>None</td>
<td>Little</td>
<td>Some</td>
<td>Good</td>
<td>Strong</td>
<td>Full</td>
</tr>
<tr>
<td>Level of uncertainty in Key architecture drivers: mission, userinterface, COTS, hardware, technology, performance.</td>
<td>Extreme</td>
<td>Significant</td>
<td>Considerable</td>
<td>Some</td>
<td>Little</td>
<td>Very Little</td>
</tr>
<tr>
<td>Number and criticality of risk items</td>
<td>&gt; 10 Critical</td>
<td>5-10 Critical</td>
<td>2-4 Critical</td>
<td>1 Critical</td>
<td>&gt; 5 Non-Critical</td>
<td>&lt; 5 Non-Critical</td>
</tr>
</tbody>
</table>

**PDR** = Product Design Review

**COTS** = Component Off-The-Shelf

far@ucalgary.ca
Elaboration of the **TEAM** rating scale:

- Use a subjective weighted average of the characteristics to account for project turbulence and entropy due to difficulties in synchronizing the project's stakeholders.
- Stakeholders include users, customers, developers, maintainers, etc.

<table>
<thead>
<tr>
<th>Characteristic</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>Consistency of stakeholder objectives and cultures</td>
<td>Little</td>
<td>Some</td>
<td>Basic</td>
<td>Considerable</td>
<td>Strong</td>
<td>Full</td>
</tr>
<tr>
<td>Ability, willingness of stakeholders to accommodate other stakeholders' objectives</td>
<td>Little</td>
<td>Some</td>
<td>Basic</td>
<td>Considerable</td>
<td>Strong</td>
<td>Full</td>
</tr>
<tr>
<td>Experience of stakeholders in operating as a team</td>
<td>None</td>
<td>Little</td>
<td>Little</td>
<td>Basic</td>
<td>Considerable</td>
<td>Extensive</td>
</tr>
<tr>
<td>Stakeholder teambuilding to achieve shared vision and commitments</td>
<td>None</td>
<td>Little</td>
<td>Little</td>
<td>Basic</td>
<td>Considerable</td>
<td>Extensive</td>
</tr>
</tbody>
</table>
Elaboration of the **PMAT** rating scale:

- Two methods based on the Software Engineering Institute’s Capability Maturity Model (CMM)
  - Method 1: Overall Maturity Level (CMM Level 1 through 5)
  - Method 2: Key Process Areas (see next slide)
Elaboration of the PMAT rating scale (Method 2):

- Decide the percentage of compliance for each of the KPAs as determined by a judgement-based averaging across the goals for all 18 Key Process Areas.

\[
PMAT = 5 - \left[ \sum_{i=1}^{18} \left( \frac{KPA\%_i}{100} \times \frac{5}{18} \right) \right]
\]

<table>
<thead>
<tr>
<th>Key Process Areas</th>
<th>Almost Always (&gt;90%)</th>
<th>Frequently (60-90%)</th>
<th>About Half (40-60%)</th>
<th>Occasionally (10-40%)</th>
<th>Rarely If Ever (&lt;10%)</th>
<th>Does Not Apply</th>
<th>Don't Know</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Requirements Management</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>2 Software Project Planning</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>3 Software Project Tracking and Oversight</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>4 Software Subcontract Management</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

(See COCOMO II Model Definition Manual for remaining details)
Cost Drivers /1

17 Cost Drivers

- **Product Factors**
  - Required Software Reliability (RELY)
  - Data Base Size (DATA)
  - Product Complexity (CPLX)
  - Required Reusability (RUSE)
  - Documentation (DOCU)

- **Platform Factors**
  - Execution Time Constraint (TIME)
  - Main Storage Constraint (STOR)
  - Platform Volatility (PVOL)

- **Personnel factors**
  - Analyst Capability (ACAP)
  - Application Experience (APEX)
  - Programmer Capability (PCAP)
  - Platform Experience (PLEX)
  - Language and Tool Experience (LTEX)
  - Personnel Continuity (PCON)

- **Project Factors**
  - Software Tools (TOOL)
  - Required Schedule (SCED)
  - Multi-Site Development (SITE)
## Cost Drivers /2

- **Post-Architecture** cost drivers: **Product**

<table>
<thead>
<tr>
<th>Description</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>RELY</strong> Required software reliability</td>
<td>0.82</td>
<td>0.92</td>
<td>1.00</td>
<td>1.10</td>
<td>1.26</td>
<td>-</td>
</tr>
<tr>
<td><strong>DATA</strong> Database size</td>
<td>-</td>
<td>0.90</td>
<td>1.00</td>
<td>1.14</td>
<td>1.28</td>
<td>-</td>
</tr>
<tr>
<td><strong>CPLX</strong> Product complexity</td>
<td>0.73</td>
<td>0.87</td>
<td>1.00</td>
<td>1.17</td>
<td>1.34</td>
<td>1.74</td>
</tr>
<tr>
<td><strong>RUSE</strong> Required reusability</td>
<td>-</td>
<td>0.95</td>
<td>1.00</td>
<td>1.07</td>
<td>1.15</td>
<td>1.24</td>
</tr>
<tr>
<td><strong>DOCU</strong> Documentation</td>
<td>0.81</td>
<td>0.91</td>
<td>1.00</td>
<td>1.11</td>
<td>1.23</td>
<td>-</td>
</tr>
</tbody>
</table>
Cost Drivers /3

- Post-Architecture cost drivers: **Platform**

<table>
<thead>
<tr>
<th>Description</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>TIME</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution time constraint</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>STOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main storage constraint</td>
<td></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>
<tr>
<td>Platform volatility</td>
<td>0.87</td>
<td>1.00</td>
<td>1.00</td>
<td>1.15</td>
<td>1.30</td>
<td></td>
</tr>
</tbody>
</table>
## Cost Drivers /4

- **Post-Architecture cost drivers: Personnel**

<table>
<thead>
<tr>
<th>Description</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>ACAP Analyst capability</td>
<td>1.42</td>
<td>1.19</td>
<td>1.00</td>
<td>0.85</td>
<td>0.71</td>
<td>-</td>
</tr>
<tr>
<td>AEXP Applications experience</td>
<td>1.22</td>
<td>1.10</td>
<td>1.00</td>
<td>0.88</td>
<td>0.81</td>
<td>-</td>
</tr>
<tr>
<td>PCAP Programmer capability</td>
<td>1.34</td>
<td>1.15</td>
<td>1.00</td>
<td>0.88</td>
<td>0.76</td>
<td>-</td>
</tr>
<tr>
<td>PEXP Platform experience</td>
<td>1.19</td>
<td>1.09</td>
<td>1.00</td>
<td>0.91</td>
<td>0.85</td>
<td>-</td>
</tr>
<tr>
<td>LTEX Language &amp; Tool experience</td>
<td>1.20</td>
<td>1.09</td>
<td>1.00</td>
<td>0.91</td>
<td>0.84</td>
<td>-</td>
</tr>
<tr>
<td>PCON Personnel continuity</td>
<td>1.29</td>
<td>1.12</td>
<td>1.00</td>
<td>0.90</td>
<td>0.81</td>
<td>-</td>
</tr>
</tbody>
</table>
Cost Drivers /5

- Post-Architecture cost drivers: Project

<table>
<thead>
<tr>
<th>Description</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>TOOL</strong> Software Tools</td>
<td>1.17</td>
<td>1.09</td>
<td>1.00</td>
<td>0.90</td>
<td>0.78</td>
<td>-</td>
</tr>
<tr>
<td><strong>SCED</strong> Development Schedule</td>
<td>1.43</td>
<td>1.14</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>-</td>
</tr>
<tr>
<td><strong>SITE</strong> Multi-site development</td>
<td>1.22</td>
<td>1.09</td>
<td>1.00</td>
<td>0.93</td>
<td>0.86</td>
<td>0.80</td>
</tr>
</tbody>
</table>
Sensitivity to Cost Drivers

COCOMO provides a framework to identify high leverage productivity improvement factors and estimate their payoffs.

For detailed elaboration of Cost Driver consult COCOMO II Model Description or visit: http://sunset.usc.edu/research/COCOMOII/expert_cocomo/drivers.html

far@ucalgary.ca
COCOMO II

Example & Tool
COCOMO II: The Tool
Example: Electronic Bookshop

- Objective: sell books online

- Specifications for the Application Composition Model (→ Object Points):
  - Number and complexity of screens and reports
  - Number of acquired components

- Specifications for the Early Design Model and Post Architecture Model (→ Function Points):
  - External inputs (EI), External outputs (EO), External inquiries (EQ), External interface files (EIF), Internal logical files (ILF)
COCOMO II - Example

Application

Composition Model
High-Level Specification

Screens

- Login Screen
- Search Screen
- Results Screen
- Purchase Screen
- Confirmation Screen

Electronic Bookshop System

Validation Component

Reports

- Transaction Report
- Inventory Update
- Credit Card Validation

far@ucalgary.ca
Application Composition Model

- **Effort**

\[ E = \frac{OP}{PROD} \]

where

- \( OP \) is the object point
- \( PROD \) is the productivity rate

To determine \( OP \), we need to estimate the number of *screens* and *reports* and data inputs for each of them.

Also we need to find the number of acquired components.

<table>
<thead>
<tr>
<th>Developers’ 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>( PROD )</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>
### ACM Example: Screens

<table>
<thead>
<tr>
<th>Screens</th>
<th>Data Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title Screen</td>
<td>Enter</td>
</tr>
<tr>
<td>Search Screen</td>
<td>Title, Author, Publisher, ISBN</td>
</tr>
<tr>
<td>Results Screen</td>
<td>Purchase</td>
</tr>
<tr>
<td>Purchase Screen</td>
<td>First Name, Last Name, Address, Phone Number, City, Province, Country, Postal Code, E-mail Address, Credit Card Number, Expiration Date</td>
</tr>
<tr>
<td>Confirmation Screen</td>
<td>New Search, Print</td>
</tr>
</tbody>
</table>
# ACM Example: Reports

## Reports

<table>
<thead>
<tr>
<th>Reports</th>
<th>Data Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Report</td>
<td>First Name, Last Name, Address, Phone Number, City, Province, Country, Postal Code, E-mail Address, Credit Card Number, Expiration Date, Title, Author, Publisher, ISBN, price</td>
</tr>
<tr>
<td>Inventory Update</td>
<td>Title, Author, Publisher, ISBN, price</td>
</tr>
<tr>
<td>Credit Card Validation</td>
<td>Credit Card Number, Expiration Date</td>
</tr>
</tbody>
</table>

*Assume this is handled by an acquired component*
Complexity of Screens/Reports

Based on Object Point (OP) complexity level definitions for Screens & Reports:

- **Simple**: Login Screen (1), Search Screen (1), Results Screen (1), Confirmation Screen (1).
- **Medium**: Purchase Screen (2), Transaction Report (5), Inventory Update (5).
- Credit Card Validation is an acquired component (10).

### Screen Complexity

<table>
<thead>
<tr>
<th>Number of views contained</th>
<th>Number and source of data tables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total &lt;4</td>
</tr>
<tr>
<td></td>
<td>&lt;2 servers</td>
</tr>
<tr>
<td></td>
<td>&lt;2 clients</td>
</tr>
<tr>
<td>&lt; 3</td>
<td>Simple 1</td>
</tr>
<tr>
<td>3 - 7</td>
<td>Simple 1</td>
</tr>
<tr>
<td>8+</td>
<td>Medium 2</td>
</tr>
</tbody>
</table>

### Report Complexity

<table>
<thead>
<tr>
<th>Number of sections contained</th>
<th>Number and source of data tables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total &lt;4</td>
</tr>
<tr>
<td></td>
<td>&lt;2 servers</td>
</tr>
<tr>
<td></td>
<td>&lt;2 clients</td>
</tr>
<tr>
<td>0 - 1</td>
<td>Simple 2</td>
</tr>
<tr>
<td>2 - 3</td>
<td>Simple 2</td>
</tr>
<tr>
<td>4+</td>
<td>Medium 5</td>
</tr>
</tbody>
</table>
Effort Estimation

- **Object Points (OP):**
  \[
  OP = (4 \times 1) + (1 \times 2) + (2 \times 5) + (1 \times 10) = 36
  \]

- **Development team consists of:**
  - 4 entry-level programmers who have 2+ years of experience in C++ and web programming
  - However, they have no experience working together
  - Thus, we expect team productivity to be low in the beginning, but improve throughout the project (→ nominal)

- For the Application Composition Model, we estimate the average productivity rate \( \text{PROD} \) to be nominal. Thus the project effort is:
  \[
  \text{Effort} = \frac{36}{13} = 2.77 \text{ [PM]}
  \]
COCOMO II - Example

Early Design Model
High-Level Specification

Electronic Bookshop System

Validation Component

Credit Card System

User

ILF
EIF
EI
EO
EQ
High-Level Specification

- The *Early Design Model* uses function points (FP) and a set of seven cost drivers.
- Assume, we have divided this project into four modules: *Search, User Input*, *Validation*, and *Purchase*.
- Use COCOMO II tool to estimate FP for each module.
COCOMO II: Define Modules

Define 4 modules
1. Search Module

Using LOC

Function Point entries

Function Point

Code reuse

Equivalent LOC

Function Point entries

Function Type

<table>
<thead>
<tr>
<th>Function Type</th>
<th># of Function Points</th>
<th>SubTotal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>Average</td>
</tr>
<tr>
<td>Inputs</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Outputs</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Files</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Interfaces</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Queries</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Total Unadjusted Function Points</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>Equivalent Total in SLOC</td>
<td>8960</td>
<td></td>
</tr>
</tbody>
</table>
SLOC and Code Adaptation

SLOC Input Dialog - <sample>

- Sizing Method
  - SLOC
  - Function Points
  - Adaptation

- Module Size in SLOC
  - SLOC: 0

- Breakage
  - % of code thrown away due to requirements volatility
  - BRAK: 0.00

- Adaptation
  - Initial SLOC
  - % Design Modified (DM): 0%
  - % Code Modified (CM): 0%
  - % Integration Modified (IM): 0%
  - Software Understanding (SU): 30%
  - Assessment & Assimilation (AA): 4%
  - Unfamiliarity with Software: 0.4 UNFM
  - % Components Automatically Translated (AT): 0%
  - Automatic Translation Productivity (ATPROD): 2400

- Computed Adaptation Adjustment Factor: 100
- Computed ASLOC: 0

OK | Cancel | Help
2. User Input Module

![SLOC Input Dialog - User Input](image)

Function Point

Equivalent LOC
3. Validation Module

![SLOC Input Dialog - Validation](image)

Function Point:

<table>
<thead>
<tr>
<th>Function Type</th>
<th># of Function Points</th>
<th>SubTotal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>Average</td>
</tr>
<tr>
<td>Inputs</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Outputs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Files</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Interfaces</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Queries</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Total Unadjusted Function Points = 20
Equivalent Total in SLOC = 2560
4. Purchase Module

Function Point

Equivalent LOC
Effort Adjustment Factor (EAF)

- The effort adjustment factor (EAF) for the Early Design model: Product Reliability and Complexity (RCPX); Required Reuse (RUSE); Platform Difficulty (PDIF); Personnel Capability (PERS); Personnel Experience (PREX); Facilities (FCIL); Schedule (SCED).
- Adjust the values for EAF factor for each module.
EAF of The Modules

**EAF - Search**

- **Base + incr % = rating**
  - Name: RCPX, USE, PDIF, PERS, PREX, FCIL, USR1, USR2
- **Base**:
  - HI, LO, NOM, NOM
- **Incr%**:
  - 0%, 0%, 0%, 0%, 0%, 0%, 0%

**EAF is also affected by Schedule**

- **EAF**: 1.95

**EAF - User Input**

- **Base + incr % = rating**
  - Name: RCPX, USE, PDIF, PERS, PREX, FCIL, USR1, USR2
- **Base**:
  - NOM, HI, LO, HI, HI, NOM, NOM
- **Incr%**:
  - 0%, 0%, 0%, 0%, 0%, 0%, 0%

**EAF is also affected by Schedule**

- **EAF**: 0.63

**EAF - Validation**

- **Base + incr % = rating**
  - Name: RCPX, USE, PDIF, PERS, PREX, FCIL, USR1, USR2
- **Base**:
  - HI, NOM, HI, NOM, HI, NOM, NOM
- **Incr%**:
  - 0%, 0%, 0%, 0%, 0%, 0%, 0%

**EAF is also affected by Schedule**

- **EAF**: 1.63

**EAF - Purchase**

- **Base + incr % = rating**
  - Name: RCPX, USE, PDIF, PERS, PREX, FCIL, USR1, USR2
- **Base**:
  - HI, NOM, HI, HI, HI, LO, NOM, NOM
- **Incr%**:
  - 0%, 0%, 0%, 0%, 0%, 0%, 0%, 0%

**EAF is also affected by Schedule**

- **EAF**: 1.33
Early Design Results

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Module Size</th>
<th>LABOR Rate ($/month)</th>
<th>ERF</th>
<th>NOM Effort DEV</th>
<th>EST Effort DEV</th>
<th>PROD</th>
<th>COST</th>
<th>INST COST</th>
<th>Staff</th>
<th>RISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>$:8960</td>
<td>5000.00</td>
<td>1.95</td>
<td>37.7</td>
<td>73.4</td>
<td>122.1</td>
<td>368831.03</td>
<td>40.9</td>
<td>3.7</td>
<td>0.0</td>
</tr>
<tr>
<td>User Input</td>
<td>$:10752</td>
<td>5000.00</td>
<td>0.63</td>
<td>45.2</td>
<td>28.4</td>
<td>378.7</td>
<td>141971.72</td>
<td>13.2</td>
<td>1.4</td>
<td>0.0</td>
</tr>
<tr>
<td>Validation</td>
<td>$:2560</td>
<td>5000.00</td>
<td>1.63</td>
<td>10.8</td>
<td>17.6</td>
<td>145.6</td>
<td>87923.77</td>
<td>34.3</td>
<td>0.9</td>
<td>0.0</td>
</tr>
<tr>
<td>Purchase</td>
<td>$:13324</td>
<td>5000.00</td>
<td>1.33</td>
<td>58.1</td>
<td>77.4</td>
<td>178.6</td>
<td>387037.30</td>
<td>28.0</td>
<td>3.9</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Productivity [LOC/ Person-Months]

Total Effort [Person-Months]

Total Schedule [Months]

Total Cost [$]

Average Staffing [Person]

Instruction Cost [$/ LOC]
Choice of Language

- Choice of development language affects the staff and total cost of the project.
- Changing from C to an OO language may reduce the cost and staff to 1/3.
COCOMO II - Example

Post Architecture Model
3. Post Architecture Model

EAF for Post Architecture

- Required Software Reliability (RELY)
- Database size (DATA)
- Documentation match to life-cycle needs (DOCU)
- Product complexity (CPLX)
- Required Reuse (RUSE)
- Execution time (TIME)
- Main storage constraint (STOR)
- Platform volatility (PVOL)
- Analyst Capability (ACAP)
- Applications Experience (AEXP)
- Programmer Capability (PCAP)
- Platform Experience (PEXP)
- Language and Tool Experience (LTEX)
- Personnel Continuity (PCON)
- Use of Software Tools (TOOL)
- Multi-Site Development (SITE)
- Required Development Schedule (SCED)
Other Parameters

- Scale factors (project specific factors not included in the EAF)
- Schedule (plays a special role in the schedule estimation)

\[ TDEV = \left[ 3.67 \times (PM)^{0.28} + 0.2 \times (B - 0.91) \right] \times \frac{SCED\%}{100} \]
Schedule & Scale Factors

- The schedule is one of the EAF factors in both Early Design and Post Architecture models.
- Scale factors also adjust the effort.
- Scale factors are not part of the 17 EAF factors.
COCOMO: Accuracy

Pred(0.3) = 0.29
Pred(1.0) = 0.60

Recall:
acceptable quality is
Pred(0.25) = 0.75

Ref [Boe81]
Constraint Models

Lawrence Putnam
The Rayleigh-Putnam Model /1

- **Productivity** is usually defined in term of program *size* divided by *effort*.
- Experimental results show that on a linear scale plot, the plots would be concave upward, indicating that the relationship between “*effort*” and “*size*” is nonlinear.

Source: QSM Web page
http://www.qsm.com/
The Rayleigh-Putnam Model /2

- Experiment shows that the defect-rate data over the entire course of the development follows the Rayleigh curve.
- Putnam assumes that the rate of expending effort is proportional to the error injection (or defect detection) rate.
- Thus, the rate of effort expenditure (=staffing) is a Rayleigh curve.

Source: QSM Web page
http://www.qsm.com/
The Norden-Rayleigh model collapses data from many DoD development programs onto one curve.

Normalization
Norden-Rayleigh and Putnam-Rayleigh Models

Rayleigh Distribution

Norden-Rayleigh Model
(Cumulative Density Function)

Putnam-Rayleigh Model
(Probability Density Function)
Experience shows that productivity is related to amount of work, development time and effort, in a nonlinear way.

Putnam’s basic productivity formula:

\[ \text{Work product} = \text{process productivity} \times \text{Effort}^a \times \text{Time}^b \]

The work product (or a representation of its functionality) is measured in some unit of size, usually LOC. Since the equation is nonlinear, effort and time terms get exponents.
Model: Software Equation

\[ \text{size} = C \times B^{1/3} \times T^{4/3} \]

\[ B = \left( \frac{1}{T^4} \right) \left( \frac{\text{size}}{C} \right)^3 \]

\[ E = 0.3935B \]

- **C**: process productivity parameter
- **B**: total effort (staff-years) \( \leftarrow \) depends on size and time!
- **T**: development time (years)
- **size**: estimate of size in LOC
- **E**: total effort at delivery time (peak of Rayleigh curve)
Putnam assumed that:

- A small number of engineers are needed at the beginning of a project (planning and specification).
- As the project progresses more detailed work is required and the number of engineers slowly increases and reaches a peak.
- The time when the Rayleigh curve reaches its maximum value corresponds to system testing and product release.
- After system testing the number of project staff falls (installation, maintenance, product retirement).

**Rayleigh curve:**

- Approximately 40% of the area under the Rayleigh curve is to the left of T and 60% to the right.

**Model: Software Equation 1/2**

\[ F(t) = B(1 - e^{-\frac{t^2}{k^2}}) \]

\[ F'(t) = B \frac{t}{k^2} e^{-\frac{t^2}{2k^2}} = \frac{t}{k^2} (B - B + Be^{-\frac{t^2}{2k^2}}) \]

\[ = \frac{t}{k^2} (B - F(t)) = p(t)(B - F(t)) \]
Model: Software Equation /3

- **Process productivity** \( (C) \), combines the effect of tools, programming language, methods, quality assurance procedures, standards, **type of software application**, etc.

- \( C \) can be determined from existing project data on size, effort, and project duration.

- Rating based on PI (Productivity Index):
  - PI = 2: Microcode
  - PI = 8: Real-time embedded, Avionics
  - PI = 11: Telecommunication

- There exist tables that relate \( PI \) to \( C \)

\[
R_t = \left( \frac{t}{k^2} \right) e^{-\frac{t^2}{2k^2}} B
\]

\[
F(t) = B(1 - e^{-\frac{t^2}{2k^2}})
\]

\[
F'(t) = B \frac{t}{k^2} e^{-\frac{t^2}{2k^2}} = \frac{t}{k^2} (B - B + Be^{-\frac{t^2}{2k^2}})
\]

\[
= \frac{t}{k^2} (B - F(t)) = p(t)(B - F(t))
\]
The **Productivity Index (PI)** is a macro measure of the total development environment. It embraces many factors, including:

- Management influence, development methods, tools, techniques, skill and experience of the development team, computer availability and application type complexity.

- PI takes values from 1 to 40.
  - Low values are associated with poor environments and tools and complex systems.
  - High values are associated with good environments, tools and management and well-understood, straightforward projects.

### Productivity Index (PI) vs. Productivity Parameter (C) Application Type

<table>
<thead>
<tr>
<th>Productivity Index (PI)</th>
<th>Productivity Parameter (C)</th>
<th>Application Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>754</td>
<td>Microcode</td>
</tr>
<tr>
<td>2</td>
<td>987</td>
<td>Firmware (ROM)</td>
</tr>
<tr>
<td>3</td>
<td>1,220</td>
<td>Real-time embedded, Avionics</td>
</tr>
<tr>
<td>4</td>
<td>1,597</td>
<td>Radar systems</td>
</tr>
<tr>
<td>5</td>
<td>1,974</td>
<td>Command and control</td>
</tr>
<tr>
<td>6</td>
<td>2,584</td>
<td>Process control</td>
</tr>
<tr>
<td>7</td>
<td>3,194</td>
<td>Telecommunications</td>
</tr>
<tr>
<td>8</td>
<td>4,181</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>5,186</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>6,765</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>8,362</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>10,946</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>13,530</td>
<td>Systems software, Scientific systems</td>
</tr>
<tr>
<td>14</td>
<td>17,711</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>21,892</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>28,657</td>
<td>Business systems</td>
</tr>
<tr>
<td>17</td>
<td>35,422</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>46,368</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>57,314</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>75,025</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>92,736</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>121,393</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>150,050</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>196,418</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>242,786</td>
<td>Highest value found so far</td>
</tr>
<tr>
<td>26</td>
<td>3,524,578</td>
<td></td>
</tr>
</tbody>
</table>

far@ucalgary.ca
Effect of Schedule Change on Effort?

- Schedule compression results in large penalty on effort
- Example:
  - Delivery time reduction by 50% results in 16-times effort increase
  - $1 : (0.5)^4$

$$F(t) = B(1 - e^{-\frac{t^2}{2k^2}})$$

$$F'(t) = B \frac{t}{k^2} e^{-\frac{t^2}{2k^2}} = \frac{t}{k^2} (B - B + Be^{-\frac{t^2}{2k^2}})$$

$$= \frac{t}{k^2} (B - F(t)) = p(t)(B - F(t))$$

$$size = C \times B^{1/3} \times T^{4/3}$$

$$B = \left( \frac{1}{T^4} \right) \left( \frac{size}{C} \right)^3$$

$$E = 0.3935B$$
Example 1

- Software size = 200 KLOC and
- $PI = 14$ (~Systems Software) $\Rightarrow C = 17711$

\[ B = \left( \frac{1}{T^4} \right) \left( \frac{200,000}{17,711} \right)^3 \]

\[ E = 0.3935B \]

- $E$ and $B$ for various project durations:

<table>
<thead>
<tr>
<th>$T$ (years)</th>
<th>$B$ (person-years)</th>
<th>$E$ (person-years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5 (75%)</td>
<td>284</td>
<td>111.9</td>
</tr>
<tr>
<td>2.0 (100%)</td>
<td>90</td>
<td>35.4</td>
</tr>
<tr>
<td>2.6 (130%)</td>
<td>31.5</td>
<td>12.4</td>
</tr>
</tbody>
</table>
Example 2

- Assume that \( C = 4,000 \) and LOC = 200,000

\[
B = \left( \frac{1}{T^4} \right) (50)^3
\]

\[E = 0.3945B\]

- \( E \) and \( B \) for various project durations:

<table>
<thead>
<tr>
<th>( T ) (years)</th>
<th>( B ) (staff-years)</th>
<th>( E ) (staff-years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7,814</td>
<td>3,082</td>
</tr>
<tr>
<td>2.5</td>
<td>3,200</td>
<td>1,262</td>
</tr>
<tr>
<td>3</td>
<td>1,543</td>
<td>609</td>
</tr>
</tbody>
</table>
Manpower-Buildup Equation

\[ D = \frac{B}{T^3} \]

- \( D \) is a constant called manpower acceleration
- \( B \) is the total project effort in staff-years
- \( T \) is the elapsed time to delivery in years.
- The manpower acceleration (\( D \)) is:
  - 12.3 for new software with many interfaces and interactions with other systems
  - 15 for standalone systems
  - 27 for re-implementations of existing systems.
Model: Combined Equation

- Using the software and manpower-buildup equations, we can solve for effort (Fenton, 1997):

\[ B = \left( \frac{\text{size}}{C} \right)^{9/7} D^{4/7} \]

- This shows that effort is proportional to size to the power 9/7 or about 1.286, which is close to Boehm’s factor which ranges from 1.05 to 1.20 in COCOMO model.

\[
S = C \cdot B^3 \cdot T^{4/3}
\]

\[
\frac{S}{C} = B^3 \cdot T^{4/3}
\]

\[
D = B / T^3
\]

\[
T^3 = B / D
\]

\[
(T^3)^{4/9} = \left( \frac{B}{D} \right)^{4/9} = T^{4/3}
\]

\[
\frac{S}{C} = B^3 \cdot T^{4/3} = B^3 \cdot \left( \frac{B}{D} \right)^{4/9} = B^{11/9} \cdot B^{4/9} \cdot \left( \frac{1}{D} \right)^{4/9}
\]

\[
\frac{S}{C} \cdot D^{4/9} = B^{3/9} \cdot B^{4/9} = B^{7/9}
\]

\[
B = \left( \frac{S}{C} \right)^{9/7} \cdot D^{4/7}
\]
Software Lifecycle Management (SLIM)

SLIM is an automated costing system based on the Rayleigh-Putnam Model.

SLIM applies the Putnam software model, linear programming, statistical simulation, and program evaluation and review technique (PERT) techniques to derive software project estimates.
The system enables a software planner to perform the following functions in an interactive session:

- **Estimate** the parameters for new projects.
- **Calibrate** the local software development environment by interpreting historical data supplied by the planner.
- **Generate solution plans** with a specific probability of success based on the estimates.

**SLIM product family** are built upon the concept of **Productivity Index (PI)**. PI is a high-level measure of a development team’s ability to produce software.

PI is calculated from a project’s size, time, effort and application complexity. \( \Leftarrow \) reflects C factor in the formula.
SLIM Summary /3

- SLIM works reasonably well for very large systems, but seriously overestimates the effort for medium and small systems.
- Some versions of SLIM use a so-called skill factor $\beta$, which is a scaling term and is a function of project size.

$$\text{size} = C \times \left( \frac{B}{\beta} \right)^{1/3} \times T^{4/3}$$

<table>
<thead>
<tr>
<th>Size (SLOC)</th>
<th>$\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-15K</td>
<td>0.16</td>
</tr>
<tr>
<td>20K</td>
<td>0.18</td>
</tr>
<tr>
<td>30K</td>
<td>0.28</td>
</tr>
<tr>
<td>40K</td>
<td>0.34</td>
</tr>
<tr>
<td>50K</td>
<td>0.37</td>
</tr>
<tr>
<td>&gt;70K</td>
<td>0.39</td>
</tr>
</tbody>
</table>

far@ucalgary.ca
Discussion & Conclusion
Size Estimation: Critics

- Can the existing algorithmic models be applied to a wide range of projects?
- It is suggested that a model is acceptable if 75% of the predicted values fall within 25% of their actual values (Fenton, 1997).
- Unfortunately most of the models are insufficient based on this criteria.
Critics: 1. Structure

- Although most researchers and practitioners agree that size is the primary determinant of effort, the exact relationship between size and effort is unclear (Fenton, 1997).

- There is little consensus about the effect of reducing or extending duration.
  - Boehm’s schedule cost driver in COCOMO assumes that increasing or decreasing duration increases project effort.
  - Putnam’s SLIM model implies that decreasing duration increases effort and increasing duration decreases effort.
  - Other studies have shown that decreasing duration decreases effort, contradicting both models.

- Most models work well in the environments for which they were derived.
Critics: 2. Size

- Most models require an estimate of product size. However, size is difficult to predict early in the development lifecycle.
- Although function points and object points can be used earlier in the lifecycle, these measures are “extremely” subjective.
- Size estimates can also be very inaccurate. Methods of estimation and data collection must be consistent to ensure an accurate prediction of product size. Unless the size metrics used in the model are the same as those used in practice, the model will not yield accurate results (Fenton, 1997).
Critics: 3. Complexity

- Many models include adjustment factors to account for particular projects and certain development practices.
- Many of the adjustment factors, such as cost drivers affect each other, resulting in the over emphasis of certain attributes. The cost drivers are also extremely subjective (Fenton, 1997).
- Calculation of adjustment factor is also often complicated. For example, calculation of the effort adjustment factor (EAF) for the detailed COCOMO model can also be somewhat complex, as it is distributed between phases of the software lifecycle.
Conclusions

- Software cost estimation is supposed to be an integral part of the software development process.
- Cost estimation models can be used to represent the relationship between effort and a primary cost factor such as size.
- Cost drivers are used to adjust the preliminary estimate provided by the primary cost factor.
- Software cost estimation models usually suffer from some common problems:
  - Models are based on empirical results rather than theory.
  - Models are often complex and rely heavily on size estimation.
- Cost estimation models: You cannot live with them and without them!
References

IN A PERFECT WORLD THE PROJECT WOULD TAKE EIGHT MONTHS.

BUT BASED ON PAST PROJECTS IN THIS COMPANY, I APPLIED A 1.5 INCOMPETENCE MULTIPLIER.

AND THEN I APPLIED AN L.W.F. OF 6.3.

1.5 \times 8 = 12 MONTHS

L.W.F?

LYING WEASEL FACTOR.

Copyright © 2002 United Feature Syndicate, Inc.