Software Engineering

Software Cost Estimation
Objectives

- To introduce the fundamentals of software costing and pricing
- To explain software productivity metric
- To explain why different techniques for software estimation:
  - LOC model
  - Function points model
  - Object point model
  - COCOMO (COstructive COst MOdel): 2 algorithmic cost estimation model
  - UCP: Use Case Points
What is Software Cost Estimation

- Predicting the cost of resources required for a software development process
Software is a Risky Business

- 53% of projects cost almost 200% of original estimate.

All surveyed projects used waterfall lifecycle.
Software is a Risky Business

- British Computer Society (BCS) survey:
  - 1027 projects
  - Only 130 were successful!

- Success was defined as:
  - deliver all system requirements
  - within budget
  - within time
  - to the quality agreed on
Why early Cost Estimation?

- Cost estimation is needed early for s/w pricing
- S/W price = cost + profit
Fundamental estimation questions

- **Effort**
  - How much effort is required to complete an activity?
  - Units: man-day (person-day), man-week, man-month, ...

- **Duration**
  - How much calendar time is needed to complete an activity? Resources assigned
  - Units: hour, day, week, month, year, ...

- **Cost of an activity**
  - What is the total cost of an activity?

- Project estimation and scheduling are interleaved management activities
Software Cost Components

1. Effort costs (dominant factor in most projects)
   - salaries
   - Social and insurance & benefits

2. Tools costs: Hardware and software for development
   - Depreciation on relatively small # of years 300K US$

3. Travel and Training costs (for particular client)

4. Overheads(OH): Costs must take overheads into account
   - costs of building, air-conditioning, heating, lighting
   - costs of networking and communications (tel, fax, )
   - costs of shared facilities (e.g library, staff restaurant, etc.)
   - depreciation costs of assets
   - Activity Based Costing (ABC)
S/W Pricing Policy

S/W price is influenced by
- economic consideration
- political consideration
- and business consideration
## Software Pricing Policy/Factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market opportunity</td>
<td>A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed.</td>
</tr>
<tr>
<td>Cost estimate uncertainty</td>
<td>If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit.</td>
</tr>
<tr>
<td>Contractual terms</td>
<td>A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.</td>
</tr>
<tr>
<td>Requirements volatility</td>
<td>If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices may be charged for changes to the requirements.</td>
</tr>
<tr>
<td>Financial health</td>
<td>Developers in financial difficulty may lower their price to gain a contract. It is better to make a small profit or break even than to go out of business.</td>
</tr>
</tbody>
</table>
Programmer Productivity

- **Rate of s/w production**
  - Needs for measurements
  - Measure software produced per time unit (Ex: LOC/hr)
    - rate of s/w production
    - software produced including documentation

- Not quality-oriented: although quality assurance is a factor in productivity assessment
S/W productivity measures are based on:

- **Size related measures:**
  - Based on some output from the software process
  - Number lines of delivered source code (LOC)

- **Function-related measures**
  - Based on an estimate of the functionality of the delivered software:
    - Function-points (are the best known of this type of measure)
    - Object-points
    - UCP
Measurement problems

- Estimating the size of the measure
- Estimating the total number of programmer-months which have elapsed
- Estimating contractor productivity (e.g. documentation team) and incorporating this estimate in overall estimate
Lines Of Code (LOC)

- Program length (LOC) can be used to predict program characteristics e.g. person-month effort and ease of maintenance

- What's a line of code?
  - The measure was first proposed when programs were typed on cards with one line per card
  - How does this correspond to statements as in Java which can span several lines or where there can be several statements on one line?

- What programs should be counted as part of the system?

- Assumes linear relationship between system size and volume of documentation
Versions of LOC

- DSI: Delivered Source Instructions
- KLOC: Thousands of LOC
- DSI
  - One instruction is one LOC
  - Declarations are counted
  - Comments are not counted
LOC

- **Advantages**
  - Simple to measure

- **Disadvantages**
  - Defined on code: it cannot measure the size of specification
  - Based on one specific view of size: length. What about complexity and functionality!!
  - Bad s/w may yield more LOC
  - Language dependent

- Therefore: Other s/w size attributes must be included
LOC Productivity

- The lower level the language, the less productive the programmer
  - The same functionality takes more code to implement in a lower-level language than in a high-level language

- Measures of productivity *based on LOC* suggest that programmers who write verbose code are more productive than programmers who write compact code !!!
Function Points: FP

Function Points is used in 2 contexts:

- Past: To develop metrics from historical data
- Future: Use of available metrics to size the s/w of a new project
Function Points

- Based on a combination of program characteristics
- The number of:
  - External (user) inputs: input transactions that update internal files
  - External (user) outputs: reports, error messages
  - User interactions: inquiries
  - Logical internal files used by the system:
    - Example a purchase order logical file composed of 2 physical files/tables Purchase_Order and Purchase_Order_Item
  - External interfaces: files shared with other systems

- A weight (ranging from 3 for simple to 15 for complex features) is associated with each of these above
- The function point count is computed by multiplying each raw count by the weight and summing all values
<table>
<thead>
<tr>
<th>measurement parameter</th>
<th>count</th>
<th>weighting factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of user inputs</td>
<td>X 3</td>
<td>simple 4 avg. 5 complex 6</td>
</tr>
<tr>
<td>number of user outputs</td>
<td>X 4</td>
<td>simple 4 avg. 5 complex 7</td>
</tr>
<tr>
<td>number of user inquiries</td>
<td>X 3</td>
<td>simple 4 avg. 5 complex 6</td>
</tr>
<tr>
<td>number of files</td>
<td>X 7</td>
<td>simple 4 avg. 5 complex 10</td>
</tr>
<tr>
<td>number of ext. interfaces</td>
<td>X 5</td>
<td>simple 4 avg. 5 complex 10</td>
</tr>
<tr>
<td>count-total</td>
<td></td>
<td></td>
</tr>
<tr>
<td>complexity multiplier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>function points</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Function Points – Taking Complexity into Account - 14 Factors F_i

Each factor is rated on a scale of:

- **Zero**: not important or not applicable
- **Five**: absolutely essential

1. Backup and recovery
2. Data communication
3. Distributed processing functions
4. Is performance critical?
5. Existing operating environment
6. On-line data entry
7. Input transaction built over multiple screens
Function Points – Taking Complexity into Account -14 Factors $F_i$ (cont.)

8. Master files updated on-line
9. Complexity of inputs, outputs, files, inquiries
10. Complexity of processing
11. Code design for re-use
12. Are conversion/installation included in design?
13. Multiple installations
14. Application designed to facilitate change by the user
Function Points – Taking Complexity into Account -14 Factors $F_i$ (cont.)

\[
FP = UFC \times \left[ 0.65 + 0.01 \times \sum_{i=1}^{14} F_i \right]
\]

UFC: Unadjusted function point count

\[0 \leq F_i \leq 5\]
FP: Advantages & Disadvantages

- **Advantages**
  - Available early .. We need only a detailed specification
  - Not restricted to code
  - Language independent
  - More accurate than LOC

- **Disadvantages**
  - Ignores quality issues of output
  - Subjective counting .. depend on the estimator
  - Hard to automate.. Automatic function-point counting is impossible
Function points and LOC

- FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language

  \[ \text{LOC} = \text{AVC} \times \text{number of function points} \]

  - AVC is a language-dependent factor varying from approximately 300 for assemble language to 12-40 for a 4GL
## Relation Between FP & LOC

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>LOC/FP (average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly language</td>
<td>320</td>
</tr>
<tr>
<td>C</td>
<td>128</td>
</tr>
<tr>
<td>COBOL</td>
<td>106</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>106</td>
</tr>
<tr>
<td>Pascal</td>
<td>90</td>
</tr>
<tr>
<td>C++</td>
<td>64</td>
</tr>
<tr>
<td>Ada</td>
<td>53</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>32</td>
</tr>
<tr>
<td>Smalltalk</td>
<td>22</td>
</tr>
<tr>
<td>Power Builder (code generator)</td>
<td>16</td>
</tr>
<tr>
<td>SQL</td>
<td>12</td>
</tr>
</tbody>
</table>
Function Points & Normalisation

- Function points are used to normalise measures (same as for LOC) for:
  - S/w productivity
  - Quality

- Error (bugs) per FP (discovered at programming)
- Defects per FP (discovered after programming)
- $ per FP
- Pages of documentation per FP
- FP per person-month
Expected Software Size

- Based on three-point
- Compute Expected Software Size (S) as weighted average of:
  - Optimistic estimate: S(opt)
  - Most likely estimate: S(ml)
  - Pessimistic estimate: S(pess)

\[ S = \frac{S(\text{opt}) + 4S(\text{ml}) + S(\text{pess})}{6} \]

- Beta probability distribution
Example 1: LOC Approach

- A system is composed of 7 subsystems as below.
- Given for each subsystem the size in LOC and the 2 metrics: productivity LOC/pm (pm: person month), Cost $/LOC
- Calculate the system total cost in $ and effort in months.

<table>
<thead>
<tr>
<th>Functions</th>
<th>estimated LOC</th>
<th>LOC/pm</th>
<th>$/LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>UICF</td>
<td>2340</td>
<td>315</td>
<td>14</td>
</tr>
<tr>
<td>2DGA</td>
<td>5380</td>
<td>220</td>
<td>20</td>
</tr>
<tr>
<td>3DGA</td>
<td>6800</td>
<td>220</td>
<td>20</td>
</tr>
<tr>
<td>DSM</td>
<td>3350</td>
<td>240</td>
<td>18</td>
</tr>
<tr>
<td>CGDF</td>
<td>4950</td>
<td>200</td>
<td>22</td>
</tr>
<tr>
<td>PCF</td>
<td>2140</td>
<td>140</td>
<td>28</td>
</tr>
<tr>
<td>DAM</td>
<td>8400</td>
<td>300</td>
<td>18</td>
</tr>
</tbody>
</table>
# Example 1: LOC Approach

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

Assuming
- Estimated project LOC = 33200
- Organisational productivity (similar project type) = 620 LOC/p-m
- Burdened labour rate = 8000 $/p-m

Then
- Effort = 33200/620 = (53.6) = 54 p-m
- Cost per LOC = 8000/620 = (12.9) = 13 $/LOC
- Project total Cost = 8000 * 54 = 432000 $
## Example 3: FP Approach

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info Domain</td>
<td>Optimistic</td>
<td>Likely</td>
<td>Pessim.</td>
<td>Est Count</td>
<td>Weight</td>
<td>FP count</td>
<td></td>
</tr>
<tr>
<td># of inputs</td>
<td>22</td>
<td>26</td>
<td>30</td>
<td>26</td>
<td>4</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td># of outputs</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td>18</td>
<td>5</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td># of inquiries</td>
<td>16</td>
<td>21</td>
<td>26</td>
<td>21</td>
<td>4</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td># of files</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>10</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td># of external inq.</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td><strong>UFC: Unadjusted Function Count</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>342</strong></td>
<td></td>
</tr>
<tr>
<td>Complexity adjustment factor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>1.17</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>400</strong></td>
<td></td>
</tr>
</tbody>
</table>
## Example 3: FP Approach (cont.)

### Complexity Factor

<table>
<thead>
<tr>
<th>Complexity factor</th>
<th>value=0</th>
<th>value=1</th>
<th>value=2</th>
<th>value=3</th>
<th>value=4</th>
<th>value=5</th>
<th>Fi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backup and recovery</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Data communication</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Distributed processing functions</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Is performance critical?</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Existing operating environment</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>On-line data entry</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Input transaction built over multiple screens</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Master files updated on-line</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Complexity of inputs, outputs, files, inquiries</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Complexity of processing</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Code design for re-use</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Are conversion/installation included in design?</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Multiple installations</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Application designed to facilitate change by the user</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

**Sigma (F)** = 52

**Complexity adjustment factor**

\[
0.65 + 0.01 \times \text{Sigma (F)} = 1.17
\]
Example 3: FP Approach (cont.)

Assuming \( \sum_i F_i = 52 \)

\[
FP = UFC \times \left[ 0.65 + 0.01 \times \sum_i F_i \right]
\]

\[
FP = 342 \times 1.17 = 400
\]

Complexity adjustment factor = 1.17
Example 4: FP Approach (cont.)

Assuming
- Estimated FP = 401
- Organisation average productivity (similar project type) = 6.5 FP/p-m (person-month)
- Burdened labour rate = 8000 $/p-m

Then
- Estimated effort = 401/6.5 = (61.65) = 62 p-m
- Cost per FP = 8000/6.5 = 1231 $/FP
- Project cost = 8000 * 62 = 496000 $
Object Points (for 4GLs)

- Object points are an alternative function-related measure to function points when 4GLs or similar languages are used for development.
- Object points are NOT the same as object classes.
- The number of object points in a program is a weighted estimate of:
  - The number of separate screens that are displayed;
  - The number of reports that are produced by the system;
  - The number of 3GL modules that must be developed to supplement the 4GL code.

C:\Software_Eng\Cocomo\Software Measurement Page, COCOMO II, object points.htm
## Object Points – Weighting

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Simple</th>
<th>Medium</th>
<th>Difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Report</td>
<td>2</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Each 3GL module</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
**Object Points – Weighting (cont.)**

- **svr**: number of server data tables used with screen/report
- **clnt**: number of client data tables used with screen/report

<table>
<thead>
<tr>
<th>Number of Views contained</th>
<th>For Screens</th>
<th>For Reports</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># and source of data tables</td>
<td># and source of data tables</td>
</tr>
<tr>
<td></td>
<td>Total &lt; 4 (≤2 svr ≤3 clnt)</td>
<td>Total &lt; 4 (≤2 svr ≤3 clnt)</td>
</tr>
<tr>
<td>&lt; 3</td>
<td>simple</td>
<td>simple</td>
</tr>
<tr>
<td>3 - 7</td>
<td>simple</td>
<td>medium</td>
</tr>
<tr>
<td>&gt; 8</td>
<td>medium</td>
<td>difficult</td>
</tr>
<tr>
<td></td>
<td>Over 1</td>
<td>simple</td>
</tr>
<tr>
<td></td>
<td>simple</td>
<td>medium</td>
</tr>
<tr>
<td></td>
<td>difficult</td>
<td>difficult</td>
</tr>
<tr>
<td></td>
<td>2 or 3</td>
<td>simple</td>
</tr>
<tr>
<td></td>
<td>medium</td>
<td>difficult</td>
</tr>
<tr>
<td></td>
<td>4 or 5</td>
<td>medium</td>
</tr>
<tr>
<td></td>
<td>difficult</td>
<td>difficult</td>
</tr>
</tbody>
</table>
Object Point Estimation

- Object points are **easier** to estimate from a specification than function points
  - simply concerned with screens, reports and 3GL modules
- At an **early** point in the development process:
  - Object points can be easily estimated
  - It is very difficult to estimate the number of lines of code in a system
Productivity Estimates

- **LOC productivity**
  - Real-time embedded systems, 40-160 LOC/P-month
  - Systems programs, 150-400 LOC/P-month
  - Commercial applications, 200-800 LOC/P-month

- **Object points productivity**
  - measured 4 - 50 object points/person-month
  - depends on tool support and developer capability

<table>
<thead>
<tr>
<th>Developer's experience and Capability / ICASE maturity 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: Productivity Object-point per person-month</td>
<td></td>
<td>7</td>
<td>13</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>
Object Point Effort Estimation

- Effort in p-m = NOP / PROD
  - NOP = number of OP of the system
  - Example: An application contains 840 OP (NOP=840) & Productivity is very high (= 50)
  - Then, Effort = 840/50 = (16.8) = 17 p-m
Adjustment for % of Reuse

Adjusted NOP = NOP * (1 - % reuse / 100)

Example:

An application contains 840 OP, of which 20% can be supplied by existing components.

Adjusted NOP = 840 * (1 – 20/100) = 672 OP

Adjusted effort = 672/50 = (13.4) = 14 p-m
## Factors affecting productivity

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application domain experience</td>
<td>Knowledge of the application domain is essential for effective software development. Engineers who already understand a domain are likely to be the most productive.</td>
</tr>
<tr>
<td>Process quality</td>
<td>The development process used can have a significant effect on productivity. This is covered in Chapter 31.</td>
</tr>
<tr>
<td>Project size</td>
<td>The larger a project, the more time required for team communications. Less time is available for development so individual productivity is reduced.</td>
</tr>
<tr>
<td>Technology support</td>
<td>Good support technology such as CASE tools, supportive configuration management systems, etc. can improve productivity.</td>
</tr>
<tr>
<td>Working environment</td>
<td>As discussed in Chapter 28, a quiet working environment with private work areas contributes to improved productivity.</td>
</tr>
</tbody>
</table>
Quality and Productivity

- All metrics based on volume/unit time are flawed because they do not take quality into account.
- Productivity may generally be increased at the cost of quality.
- If change is constant, then an approach based on counting lines of code (LOC) is not meaningful.
Estimation techniques

- There is no simple way to make an accurate estimate of the effort required to develop a software system:
  - Initial estimates may be based on inadequate information in a user requirements definition
  - The software may run on unfamiliar computers or use new technology
  - The people in the project may be unknown

- Project cost estimates may be self-fulfilling
  - The estimate defines the budget and the product is adjusted to meet the budget
Estimation techniques

- Algorithmic cost modelling
- Expert judgement
- Estimation by analogy
- Parkinson's Law
- Pricing to win
Algorithmic code modelling

- A formula – empirical relation:
  - based on historical cost information and which is generally based on the size of the software
- The formulae used in a formal model arise from the analysis of historical data.
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.
- Advantages: Relatively cheap estimation method. Can be accurate if experts have direct experience of similar systems
- Disadvantages: Very inaccurate if there are no experts!
Estimation by Analogy

- Experience-based Estimates
- The cost of a project is computed by comparing the project to a similar project in the same application domain
- Advantages: Accurate if project data available
- Disadvantages: Impossible if no comparable project has been tackled. Needs systematically maintained cost database
Estimation by Analogy: Problems

- However, new methods and technologies may make estimating based on experience inaccurate:
  - Object oriented rather than function-oriented development
  - Client-server systems rather than mainframe systems
  - Off the shelf components
  - Component-based software engineering
  - CASE tools and program generators
Parkinson's Law

- “The project costs whatever resources are available”
  (Resources are defined by the software house)
- Advantages: No overspend
- Disadvantages: System is usually unfinished

   The work is contracted to fit the budget available: by reducing functionality, quality
Pricing to Win

- The project costs whatever the customer budget is.

- Advantages: You get the contract

- Disadvantages:
  - The probability that the customer gets the system he/she wants is small.
  - Costs do not accurately reflect the work required
Pricing to Win

- This approach may seem unethical and unbusiness like
- However, when detailed information is lacking it may be the only appropriate strategy
- The project cost is agreed on the basis of an outline proposal and the development is constrained by that cost
- A detailed specification may be negotiated or an evolutionary approach used for system development
Top-down and Bottom-up Estimation

- **Top-down**
  
  Start at the system level and assess the overall system functionality

- **Bottom-up**
  
  Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate
Top-down Estimation

- Usable *without* knowledge of the system architecture and the components that might be part of the system
- Takes into account costs such as integration, configuration management and documentation
- Can underestimate the cost of solving difficult low-level technical problems
Bottom-up estimation

- Usable when
  - the architecture of the system is known and components identified
- Accurate method if the system has been designed in detail
- May underestimate costs of system level activities such as integration and documentation
Estimation Methods

- S/W project estimation should be based on several methods
- If these do not return approximately the same result, there is insufficient information available
- Some action should be taken to find out more in order to make more accurate estimates
- Pricing to win is sometimes the only applicable method
Algorithmic Cost Modelling

- Most of the work in the cost estimation field has focused on algorithmic cost modelling.

- Costs are analysed using mathematical formulas linking costs or inputs with METRICS to produce an estimated output.

- The formula is based on the analysis of historical data.

- The accuracy of the model can be improved by calibrating the model to your specific development environment, (which basically involves adjusting the weighting parameters of the metrics).
Building Metrics from measurements

- Project 1
- Project 2
- Project n

Analysis of historical data

METRICS
New Project estimation using available Metrics

- METRICS
- New Project
- Estimates for new project
Empirical Estimation Models - Algorithmic Cost Modelling

\[ \text{effort} = \text{tuning coefficient} \times \text{size} \]

- usually derived as person-months of effort required
- usually LOC but may also be function point
- empirically derived
- either an organisation-dependent constant or a number derived based on complexity of project
Algorithmic Cost Modelling

\[ \text{Effort} = A \times \text{Size}^B \times M \]

- A is an organisation-dependent constant
- B reflects the nonlinearity (disproportionate) effort for large projects
- M is a multiplier reflecting product, process and people attributes

- Most commonly used product attribute for cost estimation is code size (LOC)
- Most models are basically similar but with different values for A, B and M
Estimation Accuracy

- The size of a software system can only be known accurately when it is finished.
- Several factors influence the final size:
  - Use of COTS and components
  - Programming language
  - Distribution of system
- As the development process progresses then the size estimate becomes more accurate.
Estimate Uncertainty

Cost estimate

Feasibility  Requirements  Design  Code  Delivery

Higher uncertainty

Lower uncertainty

× 2x 4x

0.5x

0.25x

measurements
The COCOMO Cost model

Constructive Cost Model

- An empirical model based on project experience
- COCOMO'81 is derived from the analysis of 63 software projects in 1981.
- Well-documented, ‘independent’ model which is not tied to a specific software vendor

- COCOMO II (2000) takes into account different approaches to software development, reuse, etc.
## COCOMO 81

<table>
<thead>
<tr>
<th>Project complexity</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple (Organic)</td>
<td>$PM = 2.4 \times (KDSI)^{1.05} \times M$</td>
<td>Well-understood applications developed by small teams.</td>
</tr>
<tr>
<td>Moderate (Semi-detached)</td>
<td>$PM = 3.0 \times (KDSI)^{1.12} \times M$</td>
<td>More complex projects where team members may have limited experience of related systems.</td>
</tr>
<tr>
<td>Embedded</td>
<td>$PM = 3.6 \times (KDSI)^{1.20} \times M$</td>
<td>Complex projects where the software is part of a strongly coupled complex of hardware, software, regulations and operational procedures.</td>
</tr>
</tbody>
</table>

M: multiplier similar as for COCOMO II, based on 15 cost drivers
KDSI: Thousands of Delivered Source Instructions (KLOC)
Metrics: Parameters calculations

- **Least Squares method** – Curve fitting

- **Given**: n measurements of pairs \((x_i, y_i)\)
- **Required**: Best fit of measurements to get metrics parameters

- Assume: A linear relation between measured pairs:
  \[ Y = a + b \cdot x \]

  Other relations may be assumed as quadratic ‘or higher’:
  \[ Y = a + b \cdot x + c \cdot x^2 \; , \; \ldots \]

- Get metrics parameters \(a, b\) that best fit the measurements
How to get parameters $a$, $b$

Measured data $(x_i, y_i)$

Fitted data $Y_i = a + b \times x_i$

Fitting error $e_i = Y_i - y_i$

Measured pair $(x_i, y_i)$
How to get parameters $a$, $b$

- $e_i = Y_i - y_i = a + b \cdot x_i - y_i$

- For all measurements get $S$ as:
  $S$ is the sum mover $n$ measurements of squared values of $e_i$

- $S = \sum (e_i)^2 = \sum (a + b \cdot x_i - y_i)^2$
- $S = S(a, b)$
How to get parameters a, b

- Best fitting when $S$ is minimum
- $S$ is minimum when both the partial derivatives of $S$ with respect to $a$ and $b$ are zero.

- This leads to 2 equations in $a$ and $b$.

- Solve and get $a$ and $b$. 
COCOMO II

- COCOMO II is a **3-level** model that allows **increasingly detailed estimates** to be prepared as development progresses
  - **Early prototyping level**
    - Estimates based on **object points** and a simple formula is used for effort estimation
  - **Early design level**
    - Estimates based on **function points** that are then translated to LOC
    - Includes 7 cost drivers
  - **Post-architecture level**
    - Estimates based on lines of source code or function point
    - Includes 17 cost drivers

- Five scale factors replace COCOMO 81 ratings (organic, semi-detached, and embedded)
Early prototyping level - COCOMO II

- Suitable for projects built using modern GUI-builder tools
  - Based on Object Points
- Supports prototyping projects and projects where there is extensive reuse
- Based on standard estimates of developer productivity in object points/month
- Takes CASE tool use into account
- Formula is
  \[ PM = \left( \frac{NOP \times (1 - \%\text{reuse}/100)}{PROD} \right) \]
  - PM is the effort in person-months, NOP is the number of object points and PROD is the productivity
Early Design Level: 7 cost drivers - COCOMO II

- Estimates can be made after the requirements have been agreed
- Based on standard formula for algorithmic models

\[
PM = A \times \text{Size}^B \times M + PM_m
\]

- \( M = \text{PERS} \times \text{RCPX} \times \text{RUSE} \times \text{PDIF} \times \text{PREX} \times \text{FCIL} \times \text{SCED} \)
  - \( A = 2.5 \) in initial calibration,
  - Size: manually developed code in KLOC
  - Exponent \( B \)
    - varies from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity.
    - \( B \) is calculated using a Scale Factor based on 5 exponent drivers
  - \( PM_m \): represents manual adaptation for automatically generated code
\( PM_m : \) Manual Adaptation for Automatically Generated Code

\[
PM_m = \frac{(ASLOC \times (AT/100))}{APROD}
\]

- Used when big % of code is generated automatically
- ASLOC : Size of adapted components
- ATPROD: Productivity of the engineer integrating the adapted code (app. 2400 source statements per month)
- AT: % of adapted code (that is automatically generated)
COCOMO II Early Design Stage
Effort Multipliers: 7 cost drivers

- Multipliers reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.
  - RCPX - product reliability and complexity
  - RUSE - the reuse required
  - PDIF - platform difficulty
  - PREX - personnel experience
  - PERS - personnel capability
  - SCED - required schedule
  - FCIL - the team support facilities
The Exponent B
Scale Factor (SF) - COCOMO II

- Exponent B for effort calculation
- \[ B = 1.01 + 0.01 \times \text{sum} [SF (i)] \text{, i}=1,\ldots, 5 \]
  - SF = Scale Factor
- Each SF is rated on 6-point scale (ranging from 0 to 5):
  - very low (5), low (4), nominal (3), high (2), very high (1), extra high (0)
- 5 Scale Factor (exponent drivers)
  - Precedenteness
  - Development flexibility
  - Architecture/risk resolution
  - Team cohesion
  - Process maturity

- Ex: 20 KLOC ^ 1.26 / 20 KLOC ^ 1.01 = 43.58/20.6 = 2.11
## Exponent scale factors - COCOMO II

<table>
<thead>
<tr>
<th>Scale factor</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precedentedness</td>
<td>Reflects the previous experience of the organisation with this type of project. Very low means no previous experience, Extra high means that the organisation is completely familiar with this application domain.</td>
</tr>
<tr>
<td>Development flexibility</td>
<td>Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; Extra high means that the client only sets general goals.</td>
</tr>
<tr>
<td>Architecture/risk resolution</td>
<td>Reflects the extent of risk analysis carried out. Very low means little analysis, Extra high means a complete thorough risk analysis.</td>
</tr>
<tr>
<td>Team cohesion</td>
<td>Reflects how well the development team know each other and work together. Very low means very difficult interactions, Extra high means an integrated and effective team with no communication problems.</td>
</tr>
<tr>
<td>Process maturity</td>
<td>Reflects the process maturity of the organisation. The computation of this value depends on the CMM Maturity Questionnaire but an estimate can be achieved by subtracting the CMM process maturity level from 5.</td>
</tr>
</tbody>
</table>
Example: Exponent B calculations using Scale Factor

Given:
- Precedenteness - new project – rated low  SF(1) = 4
- Development flexibility - no client involvement – rated Very high - SF(2) = 1
- Architecture/risk resolution - No risk analysis – rated Very Low - SF(3) = 5
- Team cohesion - new team - nominal - SF(4) = 3
- Process maturity - some control - nominal - SF(5) = 3

Then:
- Exponent B = 1.17
Post-architecture stage - COCOMO II

- Uses same formula as early design estimates
- Estimate of size is adjusted to take into account
  - Requirements volatility: Rework required to support change
  - Extent of possible reuse: Reuse is non-linear and has associated costs so this is not a simple reduction in LOC
  - \[ ESLOC = ASLOC \times \left( AA + SU + 0.4DM + 0.3CM + 0.3IM \right)/100 \]
    - ESLOC is equivalent number of lines of new code. ASLOC is the number of lines of reusable code which must be modified, DM is the percentage of design modified, CM is the percentage of the code that is modified, IM is the percentage of the original integration effort required for integrating the reused software.
    - SU is a factor based on the cost of software understanding, AA is a factor which reflects the initial assessment costs of deciding if software may be reused.
COCOMO II Post Architecture
Effort Multipliers (17 multipliers)

- **Product attributes (5 multipliers)**
  - concerned with required characteristics of the software product being developed

- **Computer attributes (3 multipliers)**
  - constraints imposed on the software by the hardware platform

- **Personnel attributes (6 multipliers)**
  - multipliers that take the experience and capabilities of the people working on the project into account.

- **Project attributes (3 multipliers)**
  - concerned with the particular characteristics of the software development project
### COCOMO II Post Architecture

#### Effort Multipliers: 17 cost drivers

<table>
<thead>
<tr>
<th>Product attributes</th>
<th>Computer attributes</th>
<th>Personnel attributes</th>
<th>Project attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELY</td>
<td>DATA</td>
<td>ACAP</td>
<td>TOOL</td>
</tr>
<tr>
<td>Required system reliability</td>
<td>Size of database used</td>
<td>Capability of project analysts</td>
<td>Use of software tools</td>
</tr>
<tr>
<td>CPLX</td>
<td>RUSE</td>
<td>PCON</td>
<td>SCED</td>
</tr>
<tr>
<td>Complexity of system modules</td>
<td>Required percentage of reusable components</td>
<td>Personnel continuity</td>
<td>Development schedule compression</td>
</tr>
<tr>
<td>DOCU</td>
<td>STOR</td>
<td>PEXP</td>
<td>SITE</td>
</tr>
<tr>
<td>Extent of documentation required</td>
<td>Memory constraints</td>
<td>Programmer experience in project domain</td>
<td>Extent of multi-site working and quality of site communications</td>
</tr>
</tbody>
</table>

- **RELY**: Required system reliability
- **CPLX**: Complexity of system modules
- **DOCU**: Extent of documentation required
- **DATA**: Size of database used
- **RUSE**: Required percentage of reusable components
- **TIME**: Execution time constraints
- **PVOL**: Volatility of development platform
- **STOR**: Memory constraints
- **ACAP**: Capability of project analysts
- **PCAP**: Programmer capability
- **PCON**: Personnel continuity
- **AEXP**: Analyst experience in project domain
- **PEXP**: Programmer experience in project domain
- **LTEX**: Language and tool experience
- **TOOL**: Use of software tools
- **SITE**: Extent of multi-site working and quality of site communications
- **SCED**: Development schedule compression
Effects of cost drivers
Maximum & Minimum Data are from ref: Boehm, 1997

<table>
<thead>
<tr>
<th>Cost Driver</th>
<th>Initial COCOMO estimate without cost drivers (M=1)</th>
<th>Adjusted COCOMO estimate:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponent value</td>
<td>1.17</td>
<td></td>
</tr>
<tr>
<td>System size (including factors for reuse and requirements volatility)</td>
<td>128,000 DSI</td>
<td></td>
</tr>
<tr>
<td><strong>Initial COCOMO estimate without cost drivers (M=1)</strong></td>
<td><strong>730 person-months</strong></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>Very high, multiplier = 1.39</td>
<td></td>
</tr>
<tr>
<td>Complexity</td>
<td>Very high, multiplier = 1.3</td>
<td></td>
</tr>
<tr>
<td>Memory constraint</td>
<td>High, multiplier = 1.21</td>
<td></td>
</tr>
<tr>
<td>Tool use</td>
<td>Low, multiplier = 1.12</td>
<td></td>
</tr>
<tr>
<td>Schedule</td>
<td>Accelerated, multiplier = 1.29</td>
<td></td>
</tr>
<tr>
<td><strong>Adjusted COCOMO estimate:</strong></td>
<td><strong>2306 person-months</strong></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>Very low, multiplier = 0.75</td>
<td></td>
</tr>
<tr>
<td>Complexity</td>
<td>Very low, multiplier = 0.75</td>
<td></td>
</tr>
<tr>
<td>Memory constraint</td>
<td>None, multiplier = 1</td>
<td></td>
</tr>
<tr>
<td>Tool use</td>
<td>Very high, multiplier = 0.72</td>
<td></td>
</tr>
<tr>
<td>Schedule</td>
<td>Normal, multiplier = 1</td>
<td></td>
</tr>
<tr>
<td><strong>Adjusted COCOMO estimate:</strong></td>
<td><strong>295 person-months</strong></td>
<td></td>
</tr>
</tbody>
</table>

Maximum: 2306 person-months
Minimum: 295 person-months
## Effects of cost drivers  \((M = ?)\)

Maximum & Minimum Data are from ref: Boehm, 1997

<table>
<thead>
<tr>
<th>Exponent value</th>
<th>System size (including factors for reuse and requirements volatility)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial COCOMO estimate without cost drivers ((M=1))</td>
<td>1.17</td>
</tr>
<tr>
<td></td>
<td>128,000 DSI</td>
</tr>
<tr>
<td></td>
<td><strong>730 person-months</strong></td>
</tr>
</tbody>
</table>

| Reliability | Very high, multiplier = 1.39 |
| Complexity | Very high, multiplier = 1.3 |
| Memory constraint | High, multiplier = 1.21 |
| Tool use | Low, multiplier = 1.12 |
| Schedule | Accelerated, multiplier = 1.29 |

**Adjusted COCOMO estimate:** \(M = \Pi (M_i) = 3.15\)

| Reliability | Very low, multiplier = 0.75 |
| Complexity | Very low, multiplier = 0.75 |
| Memory constraint | None, multiplier = 1 |
| Tool use | Very high, multiplier = 0.72 |
| Schedule | Normal, multiplier = 1 |

**Adjusted COCOMO estimate:** \(M = \Pi (M_i) = 0.405\)

| Reliability | Very high, multiplier = 1.39 |
| Complexity | Very high, multiplier = 1.3 |
| Memory constraint | High, multiplier = 1.21 |
| Tool use | Low, multiplier = 1.12 |
| Schedule | Accelerated, multiplier = 1.29 |

**Adjusted COCOMO estimate:** \(M = \Pi (M_i) = 0.405\)

| Reliability | Very low, multiplier = 0.75 |
| Complexity | Very low, multiplier = 0.75 |
| Memory constraint | None, multiplier = 1 |
| Tool use | Very high, multiplier = 0.72 |
| Schedule | Normal, multiplier = 1 |

**Adjusted COCOMO estimate:** \(M = \Pi (M_i) = 0.405\)
Project planning

- Algorithmic cost models provide a basis for project planning as they allow alternative strategies to be compared

- Embedded spacecraft system
  - Must be reliable
  - Must minimise weight (number of chips)
  - Multipliers on reliability and computer constraints > 1

- Cost components
  - Target hardware
  - Development platform
  - Effort required
Management options

A. Use existing hardware, development system and development team

B. Processor and memory upgrade
   - Hardware cost increase
   - Experience decrease

C. Memory upgrade only
   - Hardware cost increase

D. More experienced staff

E. New development system
   - Hardware cost increase
   - Experience decrease

F. Staff with hardware experience
## Management options costs

<table>
<thead>
<tr>
<th>Option</th>
<th>RELY</th>
<th>STOR</th>
<th>TIME</th>
<th>TOOLS</th>
<th>LTEX</th>
<th>Total effort</th>
<th>Software cost</th>
<th>Hardware cost</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.39</td>
<td>1.06</td>
<td>1.11</td>
<td>0.86</td>
<td>1</td>
<td>63</td>
<td>949393</td>
<td>100000</td>
<td>1049393</td>
</tr>
<tr>
<td>B</td>
<td>1.39</td>
<td>1</td>
<td>1</td>
<td>1.12</td>
<td>1.22</td>
<td>88</td>
<td>1313550</td>
<td>120000</td>
<td>1402025</td>
</tr>
<tr>
<td>C</td>
<td>1.39</td>
<td>1</td>
<td>1.11</td>
<td>0.86</td>
<td>1</td>
<td>60</td>
<td>895653</td>
<td>105000</td>
<td>1000653</td>
</tr>
<tr>
<td>D</td>
<td>1.39</td>
<td>1.06</td>
<td>1.11</td>
<td>0.86</td>
<td>0.84</td>
<td>51</td>
<td>769008</td>
<td>100000</td>
<td>897490</td>
</tr>
<tr>
<td>E</td>
<td>1.39</td>
<td>1</td>
<td>1</td>
<td>0.72</td>
<td>1.22</td>
<td>56</td>
<td>844425</td>
<td>220000</td>
<td>1044159</td>
</tr>
<tr>
<td>F</td>
<td>1.39</td>
<td>1</td>
<td>1</td>
<td>1.12</td>
<td>0.84</td>
<td>57</td>
<td>851180</td>
<td>120000</td>
<td>1002706</td>
</tr>
</tbody>
</table>
Option choice

- Option D (use more experienced staff) appears to be the best alternative
  - However, it has a high associated risk as experienced staff may be difficult to find
- Option C (upgrade memory) has a lower cost saving but very low risk
- Overall, the model reveals the importance of staff experience in software development
As well as effort estimation, managers must estimate the calendar time required to complete a project and when staff will be required.

Calendar time can be estimated using a COCOMO II formula:

\[ TDEV = 3 \times (PM)^{(0.33+0.2\times (B-1.01))} \]

PM is the effort computation and B is the exponent computed as discussed above (B is 1 for the early prototyping model). This computation predicts the nominal schedule for the project.

The time required is independent of the number of people working on the project.
Project duration and staffing

Example

- **Given:**
  - Software development effort = 60 PM
  - Exponent B = 1.17

- **Then:**
  - Nominal schedule for the project (calendar time TDEV required to complete the project):

\[
TDEV = 3 \times (PM)^{(0.33+0.2*(1.17-1.01))} = 3 \times (PM)^{(0.36)} = 13 \text{ months}
\]
Staffing requirements

- Staff required can’t be computed by diving the development time by the required schedule – Non linear relation ship
- The number of people working on a project varies depending on the phase of the project
- The more people who work on the project, the more total effort is usually required
- A very rapid build-up of people often correlates with schedule slippage
Use Case Points UCP

- Effort: person-month based on Use Case description.

- See file: Use_Case_Points.doc