Chapter 23
Software Cost Estimation
Predicting the resources required for a software development process

Objectives
- To introduce the fundamentals of software costing and pricing
- To describe three metrics for software productivity assessment
- To explain why different techniques should be used for software estimation
- To describe the COCOMO II algorithmic cost estimation model

Topics covered
- Productivity
- Estimation techniques
- Algorithmic cost modelling
- Project duration and staffing

Fundamental estimation questions
- How much effort is required to complete an activity?
- How much calendar time is needed to complete an activity?
- What is the total cost of an activity?
- Project estimation and scheduling and interleaved management activities

Software cost components
- Hardware and software costs
- Travel and training costs
- Effort costs (the dominant factor in most projects)
  - salaries of engineers involved in the project
  - Social and insurance costs
- Effort costs must take overheads into account
  - costs of building, heating, lighting
  - costs of networking and communications
  - costs of shared facilities (e.g., library, staff restaurant, etc.)

Programmer productivity
- A measure of the rate at which individual engineers involved in software development produce software and associated documentation.
- Not quality-oriented although quality assurance is a factor in productivity assessment.
- Essentially, we want to measure useful functionality produced per time unit.
Productivity measures

- Size-related measures based on some output from the software process. This may be lines of delivered source code, object code instructions, etc.
- 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.

Measurement problems

- Estimating the size of the product
- 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

- 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

Productivity comparisons

- The lower level the language, the more productive the programmer
  - The same functionality takes more code to implement in a lower-level language than in a high-level language
- The more verbose the programmer, the higher the productivity
  - Measures of productivity based on lines of code suggest that programmers who write verbose code are more productive than programmers who write compact code

High and low level languages

<table>
<thead>
<tr>
<th>Low-level language</th>
<th>Analysis</th>
<th>Design</th>
<th>Coding</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-level language</td>
<td>Analysis</td>
<td>Design</td>
<td>Coding</td>
<td>Validation</td>
</tr>
</tbody>
</table>

System development times

<table>
<thead>
<tr>
<th></th>
<th>Analysis</th>
<th>Design</th>
<th>Coding</th>
<th>Testing</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly code</td>
<td>3 weeks</td>
<td>5 weeks</td>
<td>8 weeks</td>
<td>10 weeks</td>
<td>2 weeks</td>
</tr>
<tr>
<td>High-level language</td>
<td>3 weeks</td>
<td>5 weeks</td>
<td>8 weeks</td>
<td>4 weeks</td>
<td>2 weeks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>Effort</th>
<th>Productivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly code</td>
<td>5000 lines</td>
<td>28 weeks</td>
<td>714 lines/month</td>
</tr>
<tr>
<td>High-level language</td>
<td>1500 lines</td>
<td>20 weeks</td>
<td>300 lines/month</td>
</tr>
</tbody>
</table>
Function points

- Based on a combination of program characteristics
  - external inputs and outputs
  - user interactions
  - external interfaces
  - files used by the system
- A weight is associated with each of these
- The function point count is computed by multiplying each raw count by the weight and summing all values

Function points

- Function point count modified by complexity of the project
- FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language
  - LOC = AVC * number of function points
  - AVC is a language-dependent factor varying from 200-300 for assemble language to 2-40 for a 4GL
- FPs are very subjective. They depend on the estimator.
  - Automatic function-point counting is impossible

Object points

- 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

Object point estimation

- Object points are easier to estimate from a specification than function points as they are simply concerned with screens, reports and 3GL modules
- They can therefore be estimated at an early point in the development process. At this stage, it is very difficult to estimate the number of lines of code in a system

Productivity estimates

- Real-time embedded systems, 40-160 LOC/P-month
- Systems programs, 150-400 LOC/P-month
- Commercial applications, 200-800 LOC/P-month
- In object points, productivity has been measured between 4 and 50 object points/month depending on tool support and developer capability

Factors affecting productivity

- Individual ability - most important factor
- Application domain experience
- Process quality
- Project size
- Technology support
- Working environment
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
- It is not clear how productivity/quality metrics are related
- If change is constant then an approach based on counting lines of code 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 are 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 formulaic approach based on historical cost information and which is generally based on the size of the software
- Discussed later in this chapter

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

- 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
Top-down and bottom-up estimation

- Any of these approaches may be used top-down or bottom-up
- Top-down
  - Start at the system level and assess the overall system functionality and how this is delivered through sub-systems
- 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

- Each method has strengths and weaknesses
- 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

Experience-based estimates

- Estimating is primarily experience-based
- 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

Algorithmic cost modelling

- Cost is estimated as a mathematical function of estimated value of product, project and process attributes.
  - \[ \text{Effort} = A \times \text{Size}^2 \times M \]
  - \( A \) is an organisation-dependent constant, \( B \) reflects the disproportionate effort for large projects and \( M \) is a multiplier reflecting product, process and people attributes
- Most commonly used product attribute for cost estimation is code size
- 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, the size estimate becomes more accurate.

Estimate uncertainty

The COCOMO model

- An empirical model based on project experience
- Well-documented, ‘independent’ model which is not tied to a specific software vendor
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO II
- COCOMO II 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</td>
<td>$PM = \frac{2.4 \times (KDSI)^{1.05} \times M}{1 - \frac{\text{%reuse}}{100}}$</td>
<td>Well-understood applications developed by small teams, more complex projects where team members may have limited experience of related systems.</td>
</tr>
<tr>
<td>Moderate</td>
<td>$PM = \frac{3.0 \times (KDSI)^{1.12} \times M}{1 - \frac{\text{%reuse}}{100}}$</td>
<td>More complex projects where team members may have limited experience of related systems.</td>
</tr>
<tr>
<td>Embedded</td>
<td>$PM = \frac{3.6 \times (KDSI)^{1.20} \times M}{1 - \frac{\text{%reuse}}{100}}$</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>

COCOMO II levels

- 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
- Post-architecture level
  - Estimates based on lines of source code

Early prototyping level

- 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
  - $PM = \frac{\text{NOP} \times \left(1 - \frac{\text{%reuse}}{100}\right)}{\text{PROD}}$
  - $PM$ is the effort in person-months, NOP is the number of object points and PROD is the productivity
Object point productivity

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

Early design level

- Estimates can be made after the requirements have been agreed
- Based on standard formula for algorithmic models
  - \( PM = A \times Size^B \times M + PM_m \) where
  - \( M = PERS \times RCPX \times RUSE \times PDF \times PREX \times FCIL \times SCED \)
  - \( PM_m = (ASLOC \times (AT/100)) / ATPROD \)
  - \( A = 2.5 \) in initial calibration, Size in KLOC, B varies from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity

Multipliers

- 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
  - PDF - platform difficulty
  - PREX - personnel experience
  - PERS - personnel capability
  - SCED - required schedule
  - FCIL - the team support facilities
- PM reflects the amount of automatically generated code

Post-architecture level

- 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
  - \( ESLOC = ASLOC \times (AA + SU + 0.4DM + 0.3CM + 0.3IM)/100 \)
    - \( ESLOC \) 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.

The effects of various factors

The values of the three constants in the formula

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

are influenced by various factors as explained in the following.

B, the exponent

- In COCOMO II this depends on 5 scale factors (see next slide). Their sum/100 is added to 1.01
- Example
  - Precedenteness - new project - 4
  - Development flexibility - no client involvement - Very high - 1
  - Architecture/risk resolution - No risk analysis - V. Low - 5
  - Team cohesion - new team - nominal - 3
  - Process maturity - some control - nominal - 3
- Scale factor is therefore 1.17
Exponent scale factors

- Precedenteness: reflects previous experience
- Development flexibility
- Architecture/risk resolution: reflects the extent of risk analysis carried out
- Team cohesion: reflects how well the team members know each other and work together
- Process maturity

M, the multiplier

- Product attributes: concerned with required characteristics of the software product being developed
- Computer attributes: constraints imposed on the software by the hardware platform
- Personnel attributes: multipliers that take the experience and capabilities of the people working on the project into account.
- Project attributes: concerned with the particular characteristics of the software development project

Project cost drivers - product

- 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

Project cost driver - Computer

- TIME Execution time constraint
- PVOL Volatility of development platform
- STOR Memory constraints

Project cost drivers - Personnel

- ACAP Capability of project analysts
- PCON Personnel continuity
- PCAP Programmer capability
- PEXP Programmer experience in problem domain
- AEXP Analyst experience in problem domain
- LTEX Language and tool experience

Project cost drivers - Project

- TOOL Use of software tools
- SCED Development schedule compression
- SITE Extent of multi-site working and quality of inter-site communications
Multipliers

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

Effects of cost drivers

<table>
<thead>
<tr>
<th>Exponent value</th>
<th>System size (including factors for reuse and requirements volatility)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.17</td>
<td>128,000 DSI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initial COCOMO estimate without cost drivers</th>
<th>750 person-months</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>Very high, multiplier = 1.39</td>
</tr>
<tr>
<td>Complexity</td>
<td>Very high, multiplier = 1.3</td>
</tr>
<tr>
<td>Memory constraint</td>
<td>High, multiplier = 1.21</td>
</tr>
<tr>
<td>Tool use</td>
<td>Low, multiplier = 1.12</td>
</tr>
<tr>
<td>Schedule</td>
<td>Accelerated, multiplier = 1.29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Adjusted COCOMO estimate</th>
<th>2500 person-months</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>Very low, multiplier = 0.73</td>
</tr>
<tr>
<td>Complexity</td>
<td>Very low, multiplier = 0.73</td>
</tr>
<tr>
<td>Memory constraint</td>
<td>None, multiplier = 1</td>
</tr>
<tr>
<td>Tool use</td>
<td>Very high, multiplier = 0.72</td>
</tr>
<tr>
<td>Schedule</td>
<td>Normal, multiplier = 1</td>
</tr>
</tbody>
</table>

Management options costs

<table>
<thead>
<tr>
<th>Option</th>
<th>RELY</th>
<th>SIOR</th>
<th>TIME</th>
<th>TOOLS</th>
<th>LEX</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>0.96</td>
<td>1</td>
<td>65</td>
<td>94453</td>
<td>100000</td>
<td>1049393</td>
</tr>
<tr>
<td>B</td>
<td>1.39</td>
<td>1</td>
<td>1</td>
<td>1.32</td>
<td>1.27</td>
<td>36</td>
<td>131550</td>
<td>100000</td>
<td>1315500</td>
</tr>
<tr>
<td>C</td>
<td>1.39</td>
<td>1</td>
<td>1</td>
<td>1.32</td>
<td>1.27</td>
<td>36</td>
<td>131550</td>
<td>100000</td>
<td>1315500</td>
</tr>
<tr>
<td>D</td>
<td>1.39</td>
<td>1</td>
<td>1</td>
<td>1.32</td>
<td>1.27</td>
<td>36</td>
<td>131550</td>
<td>100000</td>
<td>1315500</td>
</tr>
<tr>
<td>E</td>
<td>1.39</td>
<td>1</td>
<td>1</td>
<td>1.32</td>
<td>1.27</td>
<td>36</td>
<td>131550</td>
<td>100000</td>
<td>1315500</td>
</tr>
<tr>
<td>F</td>
<td>1.39</td>
<td>1</td>
<td>1</td>
<td>1.32</td>
<td>1.27</td>
<td>36</td>
<td>131550</td>
<td>100000</td>
<td>1315500</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.
**Project duration and staffing**

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

**Staffing requirements**

- Staff required can’t be computed by diving the development time by the required schedule.
- 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.

**Key points**

- Factors affecting productivity include individual aptitude, domain experience, nature of the project, the project size, tool support and the working environment.
- More than one technique of cost estimation should be used when estimating costs.
- Software may be priced to gain a contract and the functionality adjusted to the price.

**Key points (continued)**

- Cost estimation is difficult because of the need to estimate attributes of the finished product.
- The COCOMO model may take project, product, personnel and hardware attributes into account when predicting effort required.
- Algorithmic cost models support quantitative option analysis.
- The time to complete a project cannot always be reduced by increasing the number of people working on the project.