Before starting to build software, it is essential to plan the entire development effort in detail. Planning continues during development and then postdelivery maintenance. Initial planning is not enough; planning must proceed throughout the project. The earliest possible time that detailed planning can take place is after the specifications are complete.

The accuracy of estimation increases as the process proceeds.

Example:
- Cost estimate of $1 million during the requirements workflow:
  - Likely actual cost is in the range ($0.25M, $4M)
- Cost estimate of $1 million at the end of the requirements workflow:
  - Likely actual cost is in the range ($0.5M, $2M)
- Cost estimate of $1 million at the end of the analysis workflow (earliest appropriate time):
  - Likely actual cost is in the range ($0.67M, $1.5M)

These four points are shown in the cone of uncertainty.

This model is old (1976). Estimating techniques have improved, but the shape of the curve is likely to be similar.

Accurate duration estimation is critical.
Accurate cost estimation is critical:
- Internal cost – cost of development
- External cost – price client pays
There are too many variables for an accurate estimate of cost or duration.
Human Factors

- Sackman (1968) measured differences of up to 28 to 1 between pairs of programmers.
  - He compared matched pairs of programmers with respect to:
    - Product size
    - Product execution time
    - Development time
    - Coding time
    - Debugging time
- Critical staff members may resign during the project.

Metrics for the Size of a Product

- Lines of code (LOC, KDSI, KLOC)
- FFP (Files, Flows, and Processes)
- Function Points
- COCOMO (COnstructive COst MOdel)

Lines of Code (LOC)

- Alternate metric
  - Thousand delivered source instructions (KDSI)
- Source code is only a small part of the total software effort
- Different languages lead to different lengths of code
- LOC is not defined for nonprocedural languages (like LISP)

Lines of Code (2)

- It is not clear how to count lines of code:
  - Executable lines of code?
  - Data definitions?
  - Comments?
  - JCL statements?
  - Changed/deleted lines?
- Not everything written is delivered to the client
- A report, screen, or GUI generator can generate thousands of lines of code in minutes

Lines of Code (3)

- LOC is accurately known only when the product finished
- Estimation based on LOC is therefore doubly dangerous:
  - To start the estimation process, LOC in the finished product must be estimated
  - The LOC estimate is then used to estimate the cost of the product — an uncertain input to an uncertain cost estimator

Metrics for the Size of a Product (2)

- Metrics based on measurable quantities that can be determined early in the software life cycle
  - FFP
  - Function points
FFP Metric

- For cost estimation of medium-scale data processing products
- The three basic structural elements of data processing products
  - Files – permanently resident collections of related records
  - Flows – data interfaces between product and environment
  - Processes – functionally defined manipulation of data

FFP Metric (2)

- Given the number of files ($F_i$), flows ($F_l$), and processes ($P_r$)
  - The size ($S$), cost ($C$) are given by
    \[ S = F_i + F_l + P_r \]
    \[ C = d \times S \]
  - The constant $d$ (efficiency or productivity) varies from organization to organization based on cost data from previously developed products

FFP Metric (3)

- The validity and reliability of the FFP metric were demonstrated using a purposive sample
  - However, the metric was never extended to include databases

Function Points

- Based on the number of inputs ($I_{np}$), outputs ($O_{ut}$), inquiries ($I_{nq}$), master files ($Maf$), interfaces ($Inf$)
  - For any product, the size in “function points” is given by
    \[ FP = 4 \times I_{np} + 5 \times O_{ut} + 4 \times I_{nq} + 10 \times Maf + 7 \times Inf \]
  - This is an oversimplification of a 3-step process

Function Points (2)

- Step 1. Classify each component of the product ($I_{np}$, $O_{ut}$, $I_{nq}$, $Maf$, $Inf$) as simple, average, or complex (Figure 9.3)
  - Assign the appropriate number of function points
  - The sum gives UFP (unadjusted function points)

<table>
<thead>
<tr>
<th>Component</th>
<th>Simple</th>
<th>Average</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input item</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Output item</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Inquiry</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Master file</td>
<td>7</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>Interface</td>
<td>5</td>
<td>7</td>
<td>10</td>
</tr>
</tbody>
</table>

Function Points (3)

- Step 2. Compute the technical complexity factor (TCF)
  - Assign a value from 0 (“not present”) to 5 (“strong influence throughout”) to each of 14 factors such as transaction rates, portability (Figure 9.4)
Function Points (4)

- Sum the 14 numbers
  - This gives the total degree of influence ($DI$)
  
  \[ TCF = 0.65 + 0.01 \times DI \]

- The technical complexity factor ($TCF$) lies between 0.65 and 1.35

Function Points (5)

- Step 3. The number of function points ($FP$) is then given by
  
  \[ FP = UFP \times TCF \]

Analysis of Function Points

- Function points are usually better than KDSI — but there are some problems
- "Errors in excess of 800% counting KDSI, but only 200% in counting function points" [Jones, 1987]

Analysis of Function Points (2)

- We obtain nonsensical results from metrics (Figure 9.5)
  - KDSI per person-month and Cost per source statement
  - Cost per function point is meaningful

<table>
<thead>
<tr>
<th>Source code size</th>
<th>Assembler Version</th>
<th>Ada Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDSI per person-month</td>
<td>$1.043M$</td>
<td>$0.59M$</td>
</tr>
<tr>
<td>Cost per source statement</td>
<td>$14.90$</td>
<td>$23.60$</td>
</tr>
<tr>
<td>Function pts. per person-month</td>
<td>$1.65$</td>
<td>$2.92$</td>
</tr>
<tr>
<td>Cost per function point</td>
<td>$3023$</td>
<td>$1170$</td>
</tr>
</tbody>
</table>

Analysis of Function Points (3)

- Like FFP, maintenance can be inaccurately measured
- It is possible to make major changes without changing
  - The number of files, flows, and processes; or
  - The number of inputs, outputs, inquiries, master files, and interfaces
- In theory, it is possible to change every line of code without changing the number of lines of code

Mk II Function Points

- This metric was put forward to compute $UFP$ more accurately
- Software is decomposed into component transactions, each consisting of input, process, and output
- Mk II function points are widely used all over the world
Techniques of Cost Estimation

- Cost estimation is hard
  - Skill levels, project complexity, project size (not linear), application area, hardware, CASE tools
  - Deadline effect – more effort, hence more cost, when closer to deadline
- Various approaches
  - Expert judgment by analogy
  - Bottom-up approach
  - Algorithmic cost estimation models

Expert Judgment by Analogy

- Experts compare the target product to completed products
  - Guesses can lead to hopelessly incorrect cost estimates
  - Experts may recollect completed products inaccurately
  - Human experts have biases
  - However, the results of estimation by a broad group of experts may be accurate
- The Delphi technique is sometimes needed to achieve consensus

Bottom-up Approach

- Break the product into smaller components
  - The smaller components may be no easier to estimate
  - However, there are process-level costs
- When using the object-oriented paradigm
  - The independence of the classes assists here
  - However, the interactions among the classes complicate the estimation process

Algorithmic Cost Estimation Models

- A metric is used as an input to a model to compute cost and duration
  - An algorithmic model is unbiased, and therefore superior to expert opinion
  - However, estimates are only as good as the underlying assumptions
- Examples
  - SLIM Model
  - Price S Model
  - CONstructive COST Model (COCOMO)

Intermediate COCOMO

- COCOMO consists of three models
  - A macro-estimation model for the product as a whole
  - Intermediate COCOMO
  - A micro-estimation model that treats the product in detail
- Examine intermediate COCOMO

Intermediate COCOMO (2)

- Step 1. Estimate the length of the product in KDSI
- Step 2. Estimate the product development mode (organic, semidetached, embedded)
- Example:
  - Straightforward product ("organic mode")
    \[ \text{Nominal effort} = 3.2 \times (\text{KDSI})^{1.05} \text{ person-months} \]
Intermediate COCOMO (3)

- Step 3. Compute the nominal effort
- Example:
  - Organic product
  - 12,000 delivered source statements (12 KDSI) (estimated)

  Nominal effort = $3.2 \times (12)^{1.05} = 43$ person-months

Intermediate COCOMO (4)

- Step 4. Multiply the nominal value by 15 software development cost multipliers (Figure 9.6)
  - Attributes of product, computation, personnel, project
  - Ratings from very low to extra high
  - Multipliers from 0.70 to 1.66

Intermediate COCOMO (5)

- Example:
  - Microprocessor-based communications processing software for electronic funds transfer network with high reliability, performance, development schedule, and interface requirements
- Step 1. Estimate the length of the product
  - 10,000 delivered source instructions (10 KDSI)
- Step 2. Estimate the product development mode
  - Complex ("embedded") mode

Intermediate COCOMO (6)

- Step 3. Compute the nominal effort
  - Nominal effort = $2.8 \times (10)^{1.20} = 44$ person-months
- Step 4. Multiply the nominal value by 15 software development cost multipliers
  - For example, required software reliability is rated high (1.15), database size is rated low (0.94), ...
  - Product of effort multipliers = 1.35
  - Estimated effort for project is therefore $1.35 \times 44 = 59$ person-months

Intermediate COCOMO (7)

- Estimated effort for project (59 person-months) is used as input for additional formulas for
  - Dollar costs
  - Development schedules
  - Phase and activity distributions
  - Computer costs
  - Annual maintenance costs
  - Related items

Intermediate COCOMO (8)

- Intermediate COCOMO has been validated with respect to a broad sample
- Actual values are within 20% of predicted values about 68% of the time
  - Intermediate COCOMO was the most accurate estimation method of its time
- Major problem
  - If the estimate of the number of lines of codes of the target product is incorrect, then everything is incorrect
COCOMO II

- 1995 extension to 1981 COCOMO that incorporates
  - Object orientation
  - Modern life-cycle models
  - Rapid prototyping
  - Fourth-generation languages
  - COTS software
- COCOMO II is far more complex than the first version

COCOMO II (2)

- There are three different models
  - Application composition model for the early phases
    - Based on feature points (similar to function points)
  - Early design model
    - Based on function points
  - Post-architecture model
    - Based on function points or KDSI

COCOMO II (3)

- The underlying COCOMO effort model is
  \[ \text{effort} = a \times \text{(size)}^b \]
  - Intermediate COCOMO
    - Three values for \((a, b)\)
    - COCOMO II
      - \(b\) varies depending on the values of certain parameters
    - COCOMO II supports reuse

COCOMO II (4)

- COCOMO II has 17 multiplicative cost drivers (was 15)
  - Seven are new
- It is too soon for results regarding
  - The accuracy of COCOMO II
  - The extent of improvement (if any) over Intermediate COCOMO

Tracking Duration and Cost Estimates

- Whatever estimation method used, careful tracking is vital
- Deviations can indicate something has gone wrong and corrective action needs to take place