Estimation for Software Projects
The overall goal of project planning is to establish a pragmatic strategy for controlling, tracking, and monitoring a complex technical project.

Why?

So the end result gets done on time, with quality!
Project Planning Task Set-I

- Establish project scope
- Determine feasibility
- Analyze risks
- Define required resources
  - Determine required human resources
  - Define reusable software resources
  - Identify environmental resources
Project Planning Task Set-II

• Estimate cost and effort
  – Decompose the problem
  – Develop two or more estimates using size, function points, process tasks or use-cases
  – Reconcile the estimates

• Develop a project schedule
  – Scheduling is considered in detail in Chapter 27.
    • Establish a meaningful task set
    • Define a task network
    • Use scheduling tools to develop a timeline chart
    • Define schedule tracking mechanisms
Write it Down!

- Project Scope
- Estimates
- Risks
- Schedule
- Control Strategy

Software Project Plan
To Understand Scope ...

- Understand the customers needs
- Understand the business context
- Understand the project boundaries
- Understand the customer’s motivation
- Understand the likely paths for change
- Understand that ...

Even when you understand, nothing is guaranteed!
What is Scope?

- **Software scope** describes
  - the functions and features that are to be delivered to end-users
  - the data that are input and output
  - the “content” that is presented to users as a consequence of using the software
  - the performance, constraints, interfaces, and reliability that bound the system.

- Scope is defined using one of two techniques:
  - “A narrative description of software scope is developed after communication with all stakeholders”.
  - “A set of use-cases is developed by end-users”.

Resources

people
  - number
  - skills
  - location

project
  - reusable software
    - OTS components
    - full-experience components
    - part.-experience components
  - new components

environment
  - software tools
  - hardware
  - network resources
Software Project Estimation

- Project scope must be understood
- Elaboration (decomposition) is necessary
- Historical metrics are very helpful
- At least two different techniques should be used
- Uncertainty is inherent in the process
Software Estimation Techniques

- Model-Based:
  - SLIM, Jensen, Bailey-Basili's, COCOMO, Gulezian's, Checkpoint, Price-S, Softcost, ESTIMACS, SERR-SEM

- Expert-Based:
  - Delphi, Rule-based systems, WBS

- Learning-Oriented:
  - Neural Networks, Case-based reasoning

- Dynamics-Based:
  - Detailed COCOMO, Price-S, System Dynamics Approach

- Regression-Based:
  - OLS, Robust Regression

- Composite:
  - Bayesian Approach
Estimation Accuracy

• Predicated on …
  – the degree to which the planner has properly estimated the size of the product to be built
  – the ability to translate the size estimate into human effort, calendar time, and dollars (a function of the availability of reliable software metrics from past projects)
  – the degree to which the project plan reflects the abilities of the software team
  – the stability of product requirements and the environment that supports the software engineering effort.
• Estimation carries inherent risk and this risk leads to uncertainty
Perform a Grammatical “parse”

Statement of Scope

Functional Decomposition
Sizing Methods

- Source Lines of Code (SLOC)
  - SEI Definition Check List
- Unadjusted Function Points (UFP)
  - IFPUG
Source Lines of Code

• Best Source: Historical data from previous projects
• Expert-Judged Lines of Code
• Expressed in thousands of source lines of code (KSLOC)
• Difficult Definition – Different Languages
• Logical Source Statement
  – SEI Source Lines of Code Check List
  – Excludes COTS, GFS, other products, language support libraries and operating systems, or other commercial libraries
## SEI Source Lines of Code Checklist

### Definition Checklist for Source Statements Counts

**Definition name:** Logical Source Statements  
**Date:**  
**Originator:** COCOMO II

<table>
<thead>
<tr>
<th>Measurement unit:</th>
<th>Physical source lines</th>
<th>Logical source statements</th>
<th>(\checkmark)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Statement type</th>
<th>Definition</th>
<th>(\checkmark)</th>
<th>Data Array</th>
<th>Includes</th>
<th>Excludes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable</td>
<td>1</td>
<td>(\checkmark)</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonexecutable</td>
<td>2</td>
<td>(\checkmark)</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Declarations</td>
<td>3</td>
<td>(\checkmark)</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler directives</td>
<td>4</td>
<td>(\checkmark)</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>5</td>
<td>(\checkmark)</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>On their own lines</td>
<td>6</td>
<td>(\checkmark)</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>On lines with source code</td>
<td>7</td>
<td>(\checkmark)</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Banners and non-blank spacers</td>
<td>8</td>
<td>(\checkmark)</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blank (empty) comments</td>
<td>9</td>
<td>(\checkmark)</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blank lines</td>
<td>10</td>
<td>(\checkmark)</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### How Produced

**Definition:**  
**Data array:** \(\checkmark\)  
**Includes:**  
**Excludes:**

<table>
<thead>
<tr>
<th>How Produced</th>
<th>Definition</th>
<th>(\checkmark)</th>
<th>Data array</th>
<th>Includes</th>
<th>Excludes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmed</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generated with source code generators</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Converted with automated translators</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copied or reused without change</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modified</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Removed</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Origin

**Definition:**  
**Data array:** \(\checkmark\)  
**Includes:**  
**Excludes:**

<table>
<thead>
<tr>
<th>Origin</th>
<th>Definition</th>
<th>(\checkmark)</th>
<th>Data array</th>
<th>Includes</th>
<th>Excludes</th>
</tr>
</thead>
<tbody>
<tr>
<td>New work: no prior existence</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prior work: taken or adapted from</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A previous version, build, or release</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commercial, off-the-shelf software (COTS), other than libraries</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Government furnished software (GFS), other than reuse libraries</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Another product</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A vendor-supplied language support library (unmodified)</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A vendor-supplied operating system or utility (unmodified)</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A local or modified language support library or operating system</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other commercial library</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A reuse library (software designed for reuse)</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other software component or library</td>
<td>(\checkmark)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Usage

**Definition:**  
**Data array:** \(\checkmark\)  
**Includes:**  
**Excludes:**
Unadjusted Function Points - I

• Based on the amount of functionality in a software project and a set of individual project factors.

• Useful since they are based on information that is available early in the project life-cycle.

• Measure a software project by quantifying the information processing functionality associated with major external data or control input, output, or file types.
Step 1. Determine function counts by type. The unadjusted function point counts should be counted by a lead technical person based on information in the software requirements and design documents. The number of each the five user function types should be counted (Internal Logical File (ILF), External Interface File (EIF), External Input (EI), External Output (EO), and External Inquiry (EQ)).

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

<table>
<thead>
<tr>
<th>For ILF and EIF</th>
<th>For EO and EQ</th>
<th>For EI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record Elements</td>
<td>Data Elements</td>
<td>File Types</td>
</tr>
<tr>
<td>1-19</td>
<td>1-5</td>
<td>0 or 1</td>
</tr>
<tr>
<td>2-5</td>
<td>2-3</td>
<td>2-3</td>
</tr>
<tr>
<td>6+</td>
<td>4+</td>
<td>4+</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Complexity Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal Logical File (ILF)</td>
<td>7 10 15</td>
</tr>
<tr>
<td>External Interface Files (EIF)</td>
<td>5 7 10</td>
</tr>
<tr>
<td>External Inputs (EI)</td>
<td>3 4 6</td>
</tr>
<tr>
<td>External Outputs</td>
<td>4 5 7</td>
</tr>
<tr>
<td>External Inquiries</td>
<td>3 4 6</td>
</tr>
</tbody>
</table>

Step 4. Compute Unadjusted Function Points. Add all the weight functions counts to get one number, the Unadjusted Function Points.
Sizing Methods: LOC/FP Approach

• compute LOC/FP using estimates of information domain values
• use historical data to build estimates for the project
Example: LOC Approach

Average productivity for systems of this type = 620 LOC/pm.

Burdened labor rate = $8000 per month, the cost per line of code is approximately $13.

Based on the LOC estimate and the historical productivity data, the total estimated project cost is $431,000 and the estimated effort is 54 person-months.
Example: FP Approach

The estimated number of FP is derived:

\[ FP_{estimated} = count_{total} \times [0.65 + 0.01 \times \sum (F_i)] = 375 \]

organizational average productivity = 6.5 FP/pm.
burdened labor rate = $8000 per month, approximately $1230/FP.
Based on the FP estimate and the historical productivity data, total estimated project cost is $461,000 and estimated effort is 58 person-months.
Process-Based Estimation

Obtained from “process framework”

<table>
<thead>
<tr>
<th>Application Functions</th>
<th>framework activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effort required to accomplish each framework activity for each application function</td>
<td></td>
</tr>
</tbody>
</table>
Based on an average burdened labor rate of $8,000 per month, the total estimated project cost is $368,000 and the estimated effort is 46 person-months.
Tool-Based Estimation

- Project Characteristics
- Calibration Factors
- LOC/FP data
### Estimation with Use-Cases

<table>
<thead>
<tr>
<th></th>
<th>use cases</th>
<th>scenarios</th>
<th>pages</th>
<th>LOC</th>
<th>LOC estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>User interface subsystem</td>
<td>6</td>
<td>10</td>
<td>6</td>
<td>560</td>
<td>3,366</td>
</tr>
<tr>
<td>Engineering subsystem group</td>
<td>10</td>
<td>20</td>
<td>8</td>
<td>3,100</td>
<td>31,233</td>
</tr>
<tr>
<td>Infrastructure subsystem group</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>1,650</td>
<td>7,970</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>42,568</td>
</tr>
</tbody>
</table>

Using 620 LOC/pm as the average productivity for systems of this type and a burdened labor rate of $8000 per month, the cost per line of code is approximately $13. Based on the use-case estimate and the historical productivity data, the total estimated project cost is $552,000 and the estimated effort is 68 person-months.
Empirical Estimation Models

General form:

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

- usually derived as person-months of effort required
- either a constant or a number derived based on complexity of project
- usually LOC but may also be function point
- empirically derived
COCOMO-II

- COCOMO II is actually a hierarchy of estimation models that address the following areas:
  - *Application composition model*. Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
  - *Early design stage model*. Used once requirements have been stabilized and basic software architecture has been established.
  - *Post-architecture-stage model*. Used during the construction of the software.
A dynamic multivariable model

\[ E = \left[ \text{LOC} \times \frac{B^{0.333}}{P} \right]^3 \times \left( \frac{1}{t^4} \right) \]

where

- \( E \) = effort in person-months or person-years
- \( t \) = project duration in months or years
- \( B \) = “special skills factor”
- \( P \) = “productivity parameter”
Estimation for OO Projects-I

- Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications.
- Using object-oriented requirements modeling (Chapter 6), develop use-cases and determine a count.
- From the analysis model, determine the number of key classes (called analysis classes in Chapter 6).
- Categorize the type of interface for the application and develop a multiplier for support classes:
  - **Interface type**                          **Multiplier**
    - No GUI                                  2.0
    - Text-based user interface               2.25
    - GUI                                     2.5
    - Complex GUI                             3.0
• Multiply the number of key classes (step 3) by the multiplier to obtain an estimate for the number of support classes.
• Multiply the total number of classes (key + support) by the average number of work-units per class. Lorenz and Kidd suggest 15 to 20 person-days per class.
• Cross check the class-based estimate by multiplying the average number of work-units per use-case
Estimation for Agile Projects

• Each user scenario (a mini-use-case) is considered separately for estimation purposes.

• The scenario is decomposed into the set of software engineering tasks that will be required to develop it.

• Each task is estimated separately. Note: estimation can be based on historical data, an empirical model, or “experience.”
  – Alternatively, the ‘volume’ of the scenario can be estimated in LOC, FP or some other volume-oriented measure (e.g., use-case count).

• Estimates for each task are summed to create an estimate for the scenario.
  – Alternatively, the volume estimate for the scenario is translated into effort using historical data.

• The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment.
The Make-Buy Decision

- Build
  - Simple (0.30)
    - $380,000
  - Difficult (0.70)
    - $450,000
- Reuse
  - Minor changes (0.40)
    - $275,000
  - Major changes (0.60)
    - Simple (0.20)
      - $310,000
    - Complex (0.80)
      - $490,000
- Buy
  - Contract
    - Minor changes (0.70)
      - $210,000
    - Major changes (0.30)
      - $400,000
    - Without changes (0.60)
      - $350,000
    - With changes (0.40)
      - $500,000
For example, the expected cost to build is:

\[
\text{expected cost}_{\text{build}} = 0.30 \times \text{(\$380K)} + 0.70 \times \text{(\$450K)}
\]

\[
= \text{\$429K}
\]

Similarly,

\[
\text{expected cost}_{\text{reuse}} = \text{\$382K}
\]

\[
\text{expected cost}_{\text{buy}} = \text{\$267K}
\]

\[
\text{expected cost}_{\text{contr}} = \text{\$410K}
\]
Q & A