Software Cost Estimation

Elsa Golden
17939-S05

Maturation milestones

1960's
Basic research
appearance of first collection of cost-related data

1976
Concept formulation
appearance of a usable system (Price S)

1978
Development & Extension
alternative systems are available (e.g. COCOMO)

1981
Enhancement & Exploration (internal)
publication of Boehm’s text (Software Engineering Economics)

1988+
Enhancement & Exploration (external)
appearance of alternative outputs of estimation (e.g. P. S., JPL)

1999
Popularization
Major conference workshops

Software Estimation Techniques

<table>
<thead>
<tr>
<th>Type</th>
<th>Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model-based</td>
<td>SLIM, COCOMO, Checkpoint, SEER-SEM, ESTIMACS, PRICE-S, SELECT Estimator</td>
</tr>
<tr>
<td>Expertise-based</td>
<td>Delphi, Rule-based</td>
</tr>
<tr>
<td>Learning-oriented</td>
<td>Neural, case-based</td>
</tr>
<tr>
<td>Dynamics-based</td>
<td>Abdel-Hamid-Madnick</td>
</tr>
<tr>
<td>Regression-based</td>
<td>OLS, Robust</td>
</tr>
<tr>
<td>Composite</td>
<td>Bayesian-COCOMO II</td>
</tr>
</tbody>
</table>

Software Life-Cycle Model (SLIM)

- Larry Putnam, Qualitative Software Measurement
- Late 1970’s
- Development lifecycle as Rayleigh distribution of project personnel vs. time
- Inputs: Function points, source instructions
- Additional factors:
  - Manpower Buildup Index (MBI)
  - Technology constant or productivity factor (PF)
- Calibration: MBI and PF derived from previous project data or set of questions (if previous data unavailable)
- Outputs: Project effort, schedule, defect rate
**Price-S**

- RCA for internal use, incl. Apollo program
- 1977- Released as proprietary model
- Currently from Price Systems
- 3 submodels
  - Acquisition submodel forecasts costs & schedules
  - Sizing submodel estimates software size in SLOC, function points, or Predictive Object Points (POPs, 1998 for sizing OO development projects)
  - Life-cycle Cost Submodel for early costing of maintenance and support phase, used together with Acquisition Submodel

**Checkpoint**

- Late 1970’s
- Originally Quick Estimation System (Quest) by Howard Rubin
- Integrated into Management and Computer Systems (MACS) in early 1980’s as ESTIMACS
- Five sequential submodels

<table>
<thead>
<tr>
<th>Submodel</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space Development Est.</td>
<td>Project organization, system structure</td>
<td>Project effort in hours, project size in function points</td>
</tr>
<tr>
<td>Staffing and Cost</td>
<td>Project effort hours, employee productivity, salaries</td>
<td>Team size, staff distribution, cost by phase and contract</td>
</tr>
<tr>
<td>Hardware Configuration</td>
<td>Project type, operating standards, estimated transaction volumes</td>
<td>Required processor power by hour, cost of design &amp; implementation</td>
</tr>
<tr>
<td>Portfolio Analyzer</td>
<td>Outputs of Staffing &amp; Cost and Risk submodels</td>
<td>Output of Staffing &amp; Cost submodels, plus any complementary outputs</td>
</tr>
</tbody>
</table>

**SEER-SEM**

- Galoreth Inc., late 1980’s
- Based on Jensen model (1983)
- Proprietary parametric estimation model
- Inputs:
  - SLOC and/or function points
- Additional factors:
  - Size, personnel, complexity, environment, constraints, platform & application, development method, standard
- Calibration:
  - User customizable knowledge base
- Outputs:
  - Capability metrics, trade-off analyses with comparison of alternatives, risk analysis

**COCOMO II**

- COCOMO published 1981, Barry Boehm
- COCOMO II published 1995
  - Basic & intermediate models replaced
  - Cost drivers replaced & updated
  - Additional development models considered
- 3 submodels
  - Applications Composition (for rapid development)
    - Inputs: Object points (count of screens, reports, inputs), weighted by 3-level complexity factor
    - Calibration: Database of 161 projects
  - Early Design (for exploration of alternative architectures)
    - Inputs: Function points or SLOC, 5 scale factors, 7 effort multipliers
    - Calibration: Effort multipliers from Post-Architecture model
  - Post-Architecture (for when top level design is complete)
    - Inputs: SLOC and/or function points, 5 scale factors, 17 effort multipliers
    - Calibration: Not yet complete
- Many commercial implementations available
ESTIMACS

- Software Productivity Research, based on work by Capers Jones
- Late 1990's
- Calibration: Database of 8000 software development projects
- Inputs: Function points or feature points
- Additional factors:
  - Captures project metrics
  - Evaluates strengths and weaknesses of software environment
  - Models process improvement recommendations
- Outputs:
  - Project effort at four levels: project, phase, activity, task
  - Resources, deliverables, costs, schedules

SELECT Estimator

- SELECT Software Tools, 1998
- Based on ObjectMetrix
- For large scale, object oriented, distributed systems
- Assumes incremental development lifecycle
- Inputs:
  - Scope of project, including applications, classes, user cases, packages, components, services
- Additional factors:
  - Complexity, reuse, technology
- Outputs:
  - Total effort in person-days, schedule in months, total development cost

Model-based techniques compared

<table>
<thead>
<tr>
<th>Group</th>
<th>Factor</th>
<th>SLAM</th>
<th>CHBASE</th>
<th>PERFECT</th>
<th>SIEVE/SMAP</th>
<th>TDEINT</th>
<th>BLURAL</th>
<th>ECOBASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Function Points</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Calibration Factors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program</td>
<td>Requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Language</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reuse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computer</td>
<td>Reliability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Performance Factors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Personnel</td>
<td>Capability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Experience</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project</td>
<td>Experience Factors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reliability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Integration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activities</td>
<td>Integration Factors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Commission</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transition/Maintenance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


- Survey paper, not suitable for Newman abstract
- State of cost estimation techniques in early 1983 by author of COCOMO and Software Engineering Economics (the book)
- Software engineering deals mainly with microeconomics (how people make decisions in resource-limited situations on a local scale)
- Economic decisions must be made at every major phase of the software development lifecycle:
  - Feasibility
  - Plans & requirements
  - Product design
  - Programming
  - Integration and testing
  - Maintenance
  - Phasesout

- Value-of-information approach (When is it worthwhile to invest in gathering more information before proceeding with a project?)
  - Condition 1: Attractive alternatives exist with payoffs that vary greatly, depending on some critical states of nature.
  - Condition 2: Appreciable probability that these critical states of nature will occur.
  - Condition 3: High probability that investigations will accurately identify occurrence of critical states of nature.
  - Condition 4: Required costs and schedule of investigations do not overly curtail their net value.
  - Condition 5: Significant side benefits will be derived from performing the investigations.

- This approach obviates 5 major pitfalls of software development
  - Always use a simulation to investigate the feasibility of complex real-time software (Cond 3 & 4)
  - Always build the software twice (Cond 1 & 2)
  - Build the software purely top-down (Cond 1 & 2)
  - Every piece of code should be proved correct (Cond 1, 2, & 3)
  - Nominal-case testing is sufficient (Cond 4)


- Major software cost estimation techniques
  - Techniques that work
    - Algorithmic models (e.g. COCOMO, SLIM)
    - Expert judgment (e.g. Delphi)
    - Analogy
    - Top-down
    - Bottom-up
    - None of these is better than the others from all aspects
  - Techniques that don’t work
    - Parkinson (work expands to fill the available volume)
    - Price to win (or schedule to win)


- Enhanced Model
  - Existing software cost estimation models are deficient in dealing with large-scale component reuse of object-oriented software development. An enhanced software cost estimation approach is described, capable of providing more accurate analyses/predictions of development cost drivers in object-oriented software designs with large-scale component reuse. The model has been tested by comparing analyses/predictions with empirically measured values of data from completed software development projects.
  - Large-scale component reuse between and across projects is not correctly costed. Lessons learned from repetitive manufacturing environments can be applied to OO software development, in the form of Activity Based Costing (ABC)

- Conditions for ABC
  - Significant indirect and overhead costs, poorly accounted for by traditional means
  - Objects for which management wants to know true cost (products, customers, projects, etc.)
  - Repetitive activities that can serve as basis for mapping costs to cost objects

Note that 20+ years later, all these areas are still full of open questions!
Recent paper: Fichman, R.G. and Kemerer, C.F.  

- Data from 15 object-oriented projects
- 5 key assumptions
  - Most of the work of reuse performed outside conventional project teams
  - Target organization performs reuse activities in systematic and measurable fashion
  - Primary cost objects of interest are software projects that consume components from shared libraries
  - Target organization wishes to assign all reuse-related costs to projects, incl. developing and maintaining reuse infrastructure, and communicating existence of components
  - Organization views production of reusable components as enabler of subsequent reuse consumption, not part-end in itself
- 27 development activities tracked
  - Systems development & maintenance
  - Program-products management & planning
  - Program supervision & general
  - Sales, general & administrative
- Results
  - Breakdown of activities, resources, cost drivers, and basis for calculating unit costs
  - Claimed support for organization efforts to understand hidden cost drivers
  - But empirical validation left for “future work”

Partial Bibliography