COCOMO 2.0: Addressing New Technologies

Barry Boehm
11th COCOMO/SCM Forum
October 10, 1996
Outline

- Candidate Software Silver Bullets
- Likely Future Software Practices Marketplace
- Challenges and COCOMO 2.0 Responses
  - Applications Composition
  - New Processes
  - Reuse and Product Line Management
  - Very High Level Languages (upcoming)
  - COTS Integration (upcoming)
  - New Software quality technologies (upcoming)
- COCOMO 2.0 status
- Conclusions
## Candidate Software Silver Bullets

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Payoffs</th>
<th>Issues</th>
</tr>
</thead>
</table>
| Application Generators, Applications Composition | Quick, cheap software  
User-programmable                                                   | Scope, scalability, extendability, interoperability |
| New Processes                          | Quick, cheap software (e.g. via rework reduction)                      | Complementary management techniques (e.g. cost modeling) |
| Object-Orientation                     | Good match to many domains  
Quick, cheap software via reuse of classes, patterns | Scope, scalability, interoperability  
Weak match to some domains                                      |
| Reuse and Product Line Management      | Quick, cheap software                                                   | Determining ROI success conditions          |
### Candidate Software Silver Bullets

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Payoffs</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise Architecture</td>
<td>Interoperability, cross-domain reuse</td>
<td>Accommodating technical change System domain issues remain</td>
</tr>
<tr>
<td>COTS</td>
<td>Cheap, powerful software Rides commercial technical curves</td>
<td>Accommodating special needs Interoperability Evolution controllability</td>
</tr>
<tr>
<td>Hypercode, New CASE Environments</td>
<td>Quick, cheap software Evolution support</td>
<td>Scalability System, domain issues remain</td>
</tr>
<tr>
<td>New Quality Technologies: Cleanroom, PSP</td>
<td>Reduce defect rates</td>
<td>Scalability Interaction with existing technologies (e.g., inspections)</td>
</tr>
</tbody>
</table>
The future of the software practices marketplace

User programming
(55M performers in US in year 2005)

Application generators (0.6M)
Application composition (0.7M)
System integration (0.7M)

Infrastructure (0.75M)
COCOMO 2.0 Coverage of Future SW Practices Sectors

- User Programming: No need for cost model
- Applications Composition: Use object counts or object points
  - Count (weight) screens, reports, 3GL routines
- System Integration; development of applications generators and infrastructure software
  - Prototyping: Applications composition model
  - Early design: Function Points and 7 cost drivers
  - Post-architecture: Source Statements or Function Points and 17 cost drivers
  - Stronger reuse/reengineering model
Outline

- Candidate Software Silver Bullets
- Likely Future Software Practices Marketplace
- Challenges and COCOMO 2.0 Responses
  - Applications Composition
  - New Processes
  - Reuse and Product Line Management
  - Very High Level Languages (upcoming)
  - COTS Integration (upcoming)
  - New software quality technologies (upcoming)
- COCOMO 2.0 status
- Conclusions
Applications Composition

- **Challenge:**
  - Modeling rapid applications composition with graphical user interface (GUI) builders, client-server support, object-libraries, etc.

- **Response:**
  - Object-Point sizing and costing model
  - Reporting estimate ranges rather than point estimates
Baseline Object Point Estimation Procedure

Step 1: Assess Object-Counts: estimate the number of screens, reports, and 3GL components that will comprise this application. Assume the standard definitions of these objects in your ICASE environment.

Step 2: Classify each object instance into simple, medium and difficult complexity levels depending on values of characteristic dimensions. Use the following scheme:

<table>
<thead>
<tr>
<th>For Screens</th>
<th>For Reports</th>
</tr>
</thead>
<tbody>
<tr>
<td># and source of data tables</td>
<td># and source of data tables</td>
</tr>
<tr>
<td>Number of Views Contained</td>
<td>Total &lt; 4 (&lt;2 srvr, &lt;3 clnt)</td>
</tr>
<tr>
<td>&lt;3</td>
<td>simple</td>
</tr>
<tr>
<td>3-7</td>
<td>simple</td>
</tr>
<tr>
<td>&gt;8</td>
<td>medium</td>
</tr>
</tbody>
</table>

Step 3: Weigh the number in each cell using the following scheme. The weights reflect the relative effort required to implement an instance of that complexity level.

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Complexity-Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen</td>
<td>Simple</td>
</tr>
<tr>
<td>Report</td>
<td>1</td>
</tr>
<tr>
<td>3GL Component</td>
<td>2</td>
</tr>
</tbody>
</table>

Step 4: Determine Object-Points: add all the weighted object instances to get one number, the Object-Point count.

Step 5: Estimate percentage of reuse you expect to be achieved in this project. Compute the New Object Points to be developed,

\[ NOP = \frac{(Object-Points) \times (100 - \%reuse)}{100} \]

Step 6: Determine a productivity rate, \( PROD = \frac{NOP}{\text{person-month}} \), from the following scheme:

<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</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

Step 7: Compute the estimated person-months: \( PM = \frac{NOP}{PROD} \).
SW Costing and Sizing Accuracy vs. Phase

- Size (DSI)
- Cost ($)

Relative Size Range

Completed Programs
USAF/ESD Proposals

Concept of Operation
Rqts. Spec.
Product Design Spec.
Detail Design Spec.
Accepted Software

Feasibility
Plans and Rqts.
Product Design
Detail Design
Devel. and Test
Phases and Milestones

10/8/96
Barry Boehm - 10
New Processes

- Challenges
  - Cost and schedule estimation of composable mixes of prototyping, spiral, evolutionary, incremental development, other models

- Responses
  - New milestone definitions
    » Basic requirements consensus, life-cycle architecture
  - Replace development modes by exponent drivers
  - Post-Initial Operational Capability (IOC) evolution
    » Drop maintenance/reuse distinction
Schedule Estimation

App. Comp

- Spiral-type
- Basic Rqts Consensus
- Schedule models TBD
- App. Comp. schedule model
- IOC Accept. Test
- Ev. Dev., Spiral
- W’fall, IncDev, EvDev, Spiral

Sys Devel

- Spiral-type
- Basic Rqts Consensus
- SW/Sys Arch. Review
- COCOMO schedule model
- IOC Accept. Test
# Life-cycle Architecture as Critical Milestone

## Candidate Critical Milestones

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete Requirements</td>
<td>• Premature decisions, e.g. user interface features</td>
</tr>
<tr>
<td>Specifications</td>
<td>• Inflexible point solutions</td>
</tr>
<tr>
<td></td>
<td>• Gold plating</td>
</tr>
<tr>
<td>Beta-Test Code</td>
<td>• High-risk downstream functions</td>
</tr>
<tr>
<td></td>
<td>• Inflexible point solutions</td>
</tr>
<tr>
<td></td>
<td>• Capabilities too far from user needs</td>
</tr>
<tr>
<td>Life-Cycle Architecture</td>
<td>• Provides flexibility to address concerns above</td>
</tr>
<tr>
<td></td>
<td>• Further concerns:</td>
</tr>
<tr>
<td></td>
<td>• Predicting directions of growth, evolution</td>
</tr>
<tr>
<td></td>
<td>• High-risk architectural elements</td>
</tr>
</tbody>
</table>
Nature of Life-Cycle Architecture

- Definition of life-cycle requirements envelope
- Definition of software components and relationships
  - physical and logical
  - data, control, timing relationships
  - shared assumptions
  - developer, operator, user, manager views
- Evidence that architecture will accommodate requirements envelope
- Resolution of all life-cycle-critical COTS, reuse, and risk issues
New Scaling Exponent Approach

- Nominal person-months = \( A \times (\text{size})^B \)
- \( B = 1.01 + 0.01 \) (exponent driver ratings)
  - \( B \) ranges from 1.01 to 1.26
  - 5 drivers; ratings from 0 to 5
- Exponent drivers:
  - Precedentedness
  - Development flexibility
  - Architecture/ risk resolution
  - Team cohesion
  - Process maturity (being derived from SEI CMM)
Outline

- Candidate Software Silver Bullets
- Likely Future Software Practices Marketplace
- Challenges and COCOMO 2.0 Responses
  - Applications Composition
  - New Processes
  - Reuse and Product Line Management
  - Very High Level Languages (upcoming)
  - COTS Integration (upcoming)
  - New Software quality technologies (upcoming)
- COCOMO 2.0 status
- Conclusions
Ada PROCESS MODEL: KEY DISTINCTIONS

"ALL TOO FREQUENT PROCESS MODEL"

LOTS OF PEOPLE WRITING CDRLs
Hordes of people trying to understand design, make progress
Long I&T phase

Massive as-built

Ada PROCESS MODEL

Small, top team working architecture, resolving high-risk items

Increment 3
Reuse and Product Line Management

- **Challenges**
  - Estimate costs of both reusing software and developing software for future reuse
  - Estimate extra effects on schedule (if any)

- **Responses**
  - New nonlinear reuse model
  - Cost of developing reusable software expanded from Ada COCOMO
  - Gathering schedule data
Function Points, SLOC, and Backfiring

- Effort/FP varies by language level (LL)
  -- But so does effort/SLOC!
- Proposed approach
  - SLOC, LL ==> Effort:
    - Compute effort as $F(\text{SLOC as 3GL})$
    - Apply stage multipliers
  - UFP, LL ==> Effort
    - Computer effort as $F\left(\text{UFP} \cdot \left[ \frac{3\text{GL-SLOC}}{\text{UFP}} \right] \right)$
    - Apply stage multipliers
4GL Cost and Schedule Effects
[Verner-Tate, 1988]

- Correspondence school information system
- Estimated size: 15 KDSI ALL [4GL], 95 KDSI COBOL
- Actual size: 13.9 KDSI ALL, 93.6 KDSI equiv. COBOL
- Data on phase distribution of effort and schedule
### 4GL Estimates vs. Actuals

<table>
<thead>
<tr>
<th>Quantity</th>
<th>COCOMO-COBOL</th>
<th>COCOMO-4GL</th>
<th>Actual Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>93.6 KSLOC</td>
<td>13.9</td>
<td>13.9</td>
</tr>
<tr>
<td>Schedule</td>
<td>--</td>
<td>11.6 Mo</td>
<td>12.0</td>
</tr>
<tr>
<td>Effort</td>
<td>247.5 PM</td>
<td>37.7</td>
<td>61.6</td>
</tr>
<tr>
<td>Plans&amp;Rqts</td>
<td>14.8</td>
<td>2.3</td>
<td>14.9</td>
</tr>
<tr>
<td>Prel Design</td>
<td>51.2</td>
<td>7.2</td>
<td>20.5</td>
</tr>
<tr>
<td>DD/CUT/I&amp;T</td>
<td>196.3</td>
<td>30.5</td>
<td>26.2</td>
</tr>
</tbody>
</table>
Effort Distribution Relative to 3GL Development

<table>
<thead>
<tr>
<th>Rapid App. Devel.</th>
<th>Spiral-type</th>
<th>Ev. Dev., Spiral</th>
<th>LCO, LCA</th>
<th>SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>5GL</td>
<td>4</td>
<td></td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sys Devel</th>
<th>Spiral-type</th>
<th>Waterfall, Spiral-type</th>
<th>W'fall, IncDev, EvDev, Spiral, Design-to-Cost, etc.</th>
<th>LCO</th>
<th>LCA</th>
<th>SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4GL</td>
<td>6</td>
<td></td>
<td></td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>3GL</td>
<td>7</td>
<td></td>
<td></td>
<td>17</td>
<td>25</td>
<td>58</td>
</tr>
<tr>
<td>2GL</td>
<td>8</td>
<td></td>
<td></td>
<td>19</td>
<td>27</td>
<td>154</td>
</tr>
</tbody>
</table>

BB 04/22/98
Function Points and TANSTAAFL
(There Ain't No Such Thing As A Free Lunch)

"Effort/FP varies by language level"

means both

"Something like FP's are needed for productivity metric across language levels (vs. SLOC)"

and

"FP’s are a poor language-independent effort estimator"
COTS Integration

- **Challenge:**
  - Estimate costs, schedules of complex COTS integration activities

- **Responses:**
  - Separately address COTS platforms, COTS tools via existing cost drivers
  - Rough experimental model of COTS integration into product
COTS Integration

- Consider as experimental downstream COCOMO 2.0 addition
- Continue to use current cost drivers for COTS tool and platform effects
- COTS portions of application: Use Loral-type weighted COTS external interfaces as size add-on
  - Separate COTS-App, COTS-COTS, COTS-Platform interface factors
  - Develop 5-step technical complexity scales for each, including other Loral factors such as release interval
- Use activity based costing approach for COTS acquisition-related costs
COTS Cost and Risk Modeling: Some Results from USC-CSE Affiliates' Workshop

<table>
<thead>
<tr>
<th>Cost</th>
<th>Risk</th>
<th>Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>COTS immaturity and lack of support</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>Staff inexperience with COTS integration, COTS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Incompatibility of COTS with</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>- Application</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>- Other COTS, reused software</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>- Computing platform, infrastructure</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>- Performance, scalability, reliability, ... rqtts.</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td>Licenses, tailoring, training</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td>Lack of COTS evolution controllability</td>
</tr>
</tbody>
</table>

5/30/95
Reuse and Product Line Management

- Challenges
  - Estimate costs of both reusing software and developing software for future reuse
  - Estimate extra effects on schedule (if any)

- Responses
  - New nonlinear reuse model
  - Cost of developing reusable software expanded from Ada COCOMO
  - Gathering schedule data
Nonlinear Reuse Effects

Data on 2954 NASA modules [Selby, 1988]

Usual Linear Assumption

Amount Modified

Relative cost

0.046

0.25

0.5

0.75

1.0

0.25

0.5

0.75

1.0

0.55

0.70

1.0

10/8/96

Barry Boehm - 18
Reuse and Reengineering Effects

- Add Assessment & Assimilation increment (AA)
  - Similar to conversion planning increment
- Add software understanding increment (SU)
  - To cover nonlinear software understanding effects
  - Apply only if reused software is modified
- Results in revised Adaptation Adjustment Factor (AAF)
  - Equivalent DSI = (Adapted DSI) * (AAF/100)
  - AAF = AA + SU + 0.4(DM) + 0.3 (CM) + 0.3 (IM)
Prototype of Software Understanding Rating / Increment

<table>
<thead>
<tr>
<th>Structure</th>
<th>Very Low</th>
<th>Moderately Low</th>
<th>Reasonably High</th>
<th>High</th>
<th>Strong Modularity, Information Hiding in Data Control Structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cohesion</td>
<td>Low</td>
<td>Moderate</td>
<td>High</td>
<td>Strong Cohesion, Low Coupling.</td>
<td></td>
</tr>
<tr>
<td>Coupling</td>
<td>High</td>
<td>Moderate</td>
<td>High</td>
<td>Strong Cohesion, Low Coupling.</td>
<td></td>
</tr>
<tr>
<td>Spaghetti Code</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application Clarity</th>
<th>No Match Between Program and Application World Views</th>
<th>Some Correlation Between Program and Application</th>
<th>Moderate Correlation Between Program and Application</th>
<th>Good Correlation Between Program and Application</th>
<th>Clear Match Between Program and Application World Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>SU Increment to ESLOC</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SU Increment to ESLOC</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>
COTS Integration

- **Challenge:**
  - Estimate costs, schedules of complex COTS integration activities

- **Responses:**
  - Separately address COTS platforms, COTS tools via existing cost drivers
  - Rough experimental model of COTS integration into product
COTS Integration

- Consider as experimental downstream COCOMO 2.0 addition
- Continue to use current cost drivers for COTS tool and platform effects
- COTS portions of application: Use Loral-type weighted COTS external interfaces as size add-on
  - Separate COTS-App, COTS-COTS, COTS-Platform interface factors
  - Develop 5-step technical complexity scales for each, including other Loral factors such as release interval
- Use activity based costing approach for COTS acquisition-related costs
# COTS Cost and Risk Modeling:
## Some Results from USC-CSE Affiliates’ Workshop

<table>
<thead>
<tr>
<th>Cost</th>
<th>Risk</th>
<th>Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>COTS immaturity and lack of support</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>Staff inexperience with COTS integration, COTS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Incompatibility of COTS with</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Application</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Other COTS, reused software</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Computing platform, infrastructure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Performance, scalability, reliability, ... rqts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Licenses, tailoring, training</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td>Lack of COTS evolution controllability</td>
</tr>
</tbody>
</table>

10/8/96

Barry Boehm - 23
Examples of COTS Integration Cost Driver Scales

<table>
<thead>
<tr>
<th>COTS Defect Level</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>COTS Defect Level</td>
<td>Very robust, Mature COTS</td>
<td>Some defects; rarely serious</td>
<td>Moderate level of defects; occasionally serious</td>
<td>Many defects; some serious</td>
<td>Many serious interacting defects</td>
</tr>
<tr>
<td>Effort Multipliers</td>
<td>0.80</td>
<td>0.89</td>
<td>1.00</td>
<td>1.20</td>
<td>1.45</td>
</tr>
<tr>
<td>COTS Volatility</td>
<td>Major change every 12 months; Minor: 1 month</td>
<td>Major: 6 months</td>
<td>Major: 2 months Minor: 1 week</td>
<td>Major: 2 weeks Minor: 2 days</td>
<td></td>
</tr>
<tr>
<td>Effort Multipliers</td>
<td>0.87</td>
<td>1.00</td>
<td>1.15</td>
<td>1.30</td>
<td></td>
</tr>
</tbody>
</table>
Legacy Software Reengineering

- **Challenge:**
  - Varying reengineering efficiencies based on tool applicability, source-target system differences

- **Response:**
  - Exploratory model based on NIST IRS data
NIST Reengineering Case Study [Ruhl-Gunn, 1991]

- IRS Centralized Scheduling Program
- Mix of batch, database query/update, on-line processing
- 50 KSLOC of COBOL74, 2.4 KSLOC of assembly code
- Unisys 1100 with DMS 1100 network database
- Reengineer to SQL database, open systems
- Tools for program and data reengineering
## NIST Reengineering Automation Data

<table>
<thead>
<tr>
<th>Program Group</th>
<th>Automated (%)</th>
<th>Manual (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch</td>
<td>96%</td>
<td>4%</td>
</tr>
<tr>
<td>Batch with SORT</td>
<td>90%</td>
<td>10%</td>
</tr>
<tr>
<td>Batch with DBMS</td>
<td>88%</td>
<td>12%</td>
</tr>
<tr>
<td>Batch, SORT, DBMS</td>
<td>82%</td>
<td>18%</td>
</tr>
<tr>
<td>Interactive</td>
<td>50%</td>
<td>50%</td>
</tr>
</tbody>
</table>
COCOMO 2.0 Current Status

- Model structure iterated with Affiliates
  - Largely converged
  - Recent function point backfiring refinements being considered
- Model being calibrated to Affiliates', other data
  - Currently around 65 solid data points
- Model being beta-tested by Affiliates
  - Reasonable results so far
Current COCOMO 2.0 Affiliates (27)

- Commercial Industry (9)
  - AT&T, Bellcore, CSC, EDS, Motorola, Rational, Sun, TI, Xerox
- Aerospace Industry (10)
  - E-Systems, Hughes, Litton, Lockheed Martin, Loral, MDAC, Northrop Grumman, Rockwell, SAIC, TRW
- Government (3)
  - AFCAAA, USAF Rome Lab, US Army Research Labs
- FFRDC’s and Consortia (5)
  - Aerospace, IDA, MCC, SEI, SPC
Conclusions

- Gone are the simple days
  - Proliferation of process models, COTS/reuse opportunities, languages
- COCOMO 2.0 trying to cope with trends
  - Composable, tailororable model elements
- Would appreciate feedback on approach