Software Engineering Economics (CS656)

Software Cost Estimation w/ COCOMO II

Jongmoon Baik

KOREA ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY
KAIST
한국과학기술원
SINCE 1971

Software Cost Estimation

"You can not control what you can not see"

– Tom Demarco –

Why Estimate Software?

- 30% of project never complete
- 100-200% cost overruns not uncommon
- Average project exceeds cost by 90%; Schedule by 120%
- 15% of large project never deliver anything
- Only 16.2% of projects are successful

* 1998, 1999, 2000 Standish report, Choas

When to Estimate?

- Estimation during the Bid
  - Short duration, fastest possible, least understanding
- Estimation at project Start
  - Creating full plan, allocating resources, detailed estimation
- Estimation during the project
  - How do you handle change

Why are we bad at estimating?

- Complexity of the systems
  - Infrequency - How often do we do the "same thing"
    - vs. manufacturing or construction
  - Underestimation bias
    - Computers are "easy"; software is "easy"
  - We deal with Goals not estimates
    - Must be done by June
  - Complexity is what makes estimating hard

Why are we bad at estimating?

- Complexity of the systems
  - ~1000 FP in a pace maker (50K)
  - ~18,800 FP in shuttle test scaffolding (1,000,000 LOC)
  - ~75,400 FP in Nynex Switch (4,000,000LOC)

- "Human brain capacity is more or less fixed, but software complexity grows at least as fast as the square of the size of the project" Tony Bowden

Determining "Development effort"

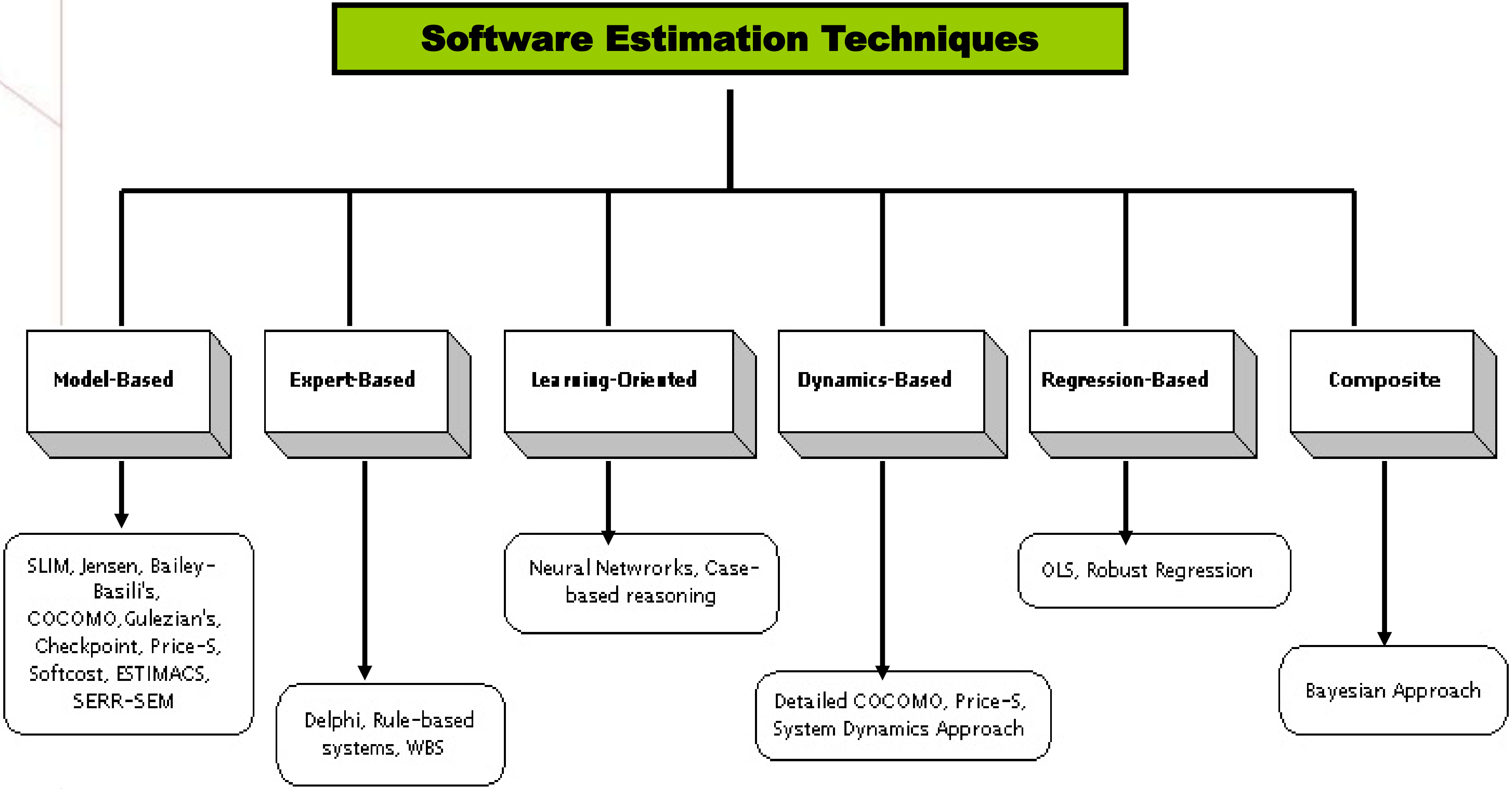
- Development effort measures
  - Person-Month
  - LOC per Hour
  - Function point per hour
  - Requirement per hour
- Most common is person-months (or hours)
- We will look at ways to get development effort

Why Is It Important??

- Software cost is big and growing
- Many useful software products are not getting developed
- Get us better software not just more software

Boehm et. Al, "Understanding and Controlling Software Cost", IEEE TSE, SE4, 10, pp1462-77

Software Estimation Techniques



Software Cost Estimation Steps

1. Establish Objectives
   - Rough Sizing
   - Make-or-Buy
   - Detailed Planning
2. Allocate Enough Time, Dollars, Talent
3. Pin down Software Requirements
   - Documents Definition, Assumption
4. Work out as much detail as Objectives permit
5. Use several independent Techniques + Sources
   - Top-Down vs. Bottom-Up
   - Algorithm Vs. Expert-Judgement
6. Compare and iterate estimates
   - Pin down and resolve inconsistencies
   - Be Conservative
7. Follow up

WHO SANG COCOMO?

- The Beach Boys [1988]
- *"KoKoMo"*

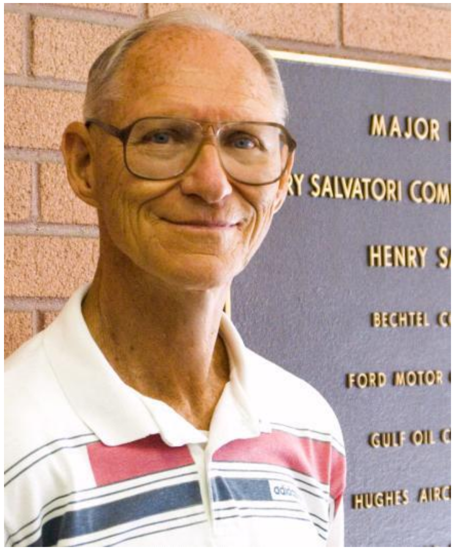
Aruba, Jamaica,ooo I wanna take you
To Bermuda, Bahama,come on, pretty mama
Key Largo, Montego, baby why don't we go jamaica
Off the Florida Keys there's a place called Kokomo
That's where you want to go to get away from it all
Bodies in the sand
Tropical drink melting in your hand
We'll be falling in love to the rhythm of a steel drum band
Down in Kokomo
....
....

Who are COCOMO?

KBS2 – "an exploration party to challenge the globe"
Sep. 4, 2005

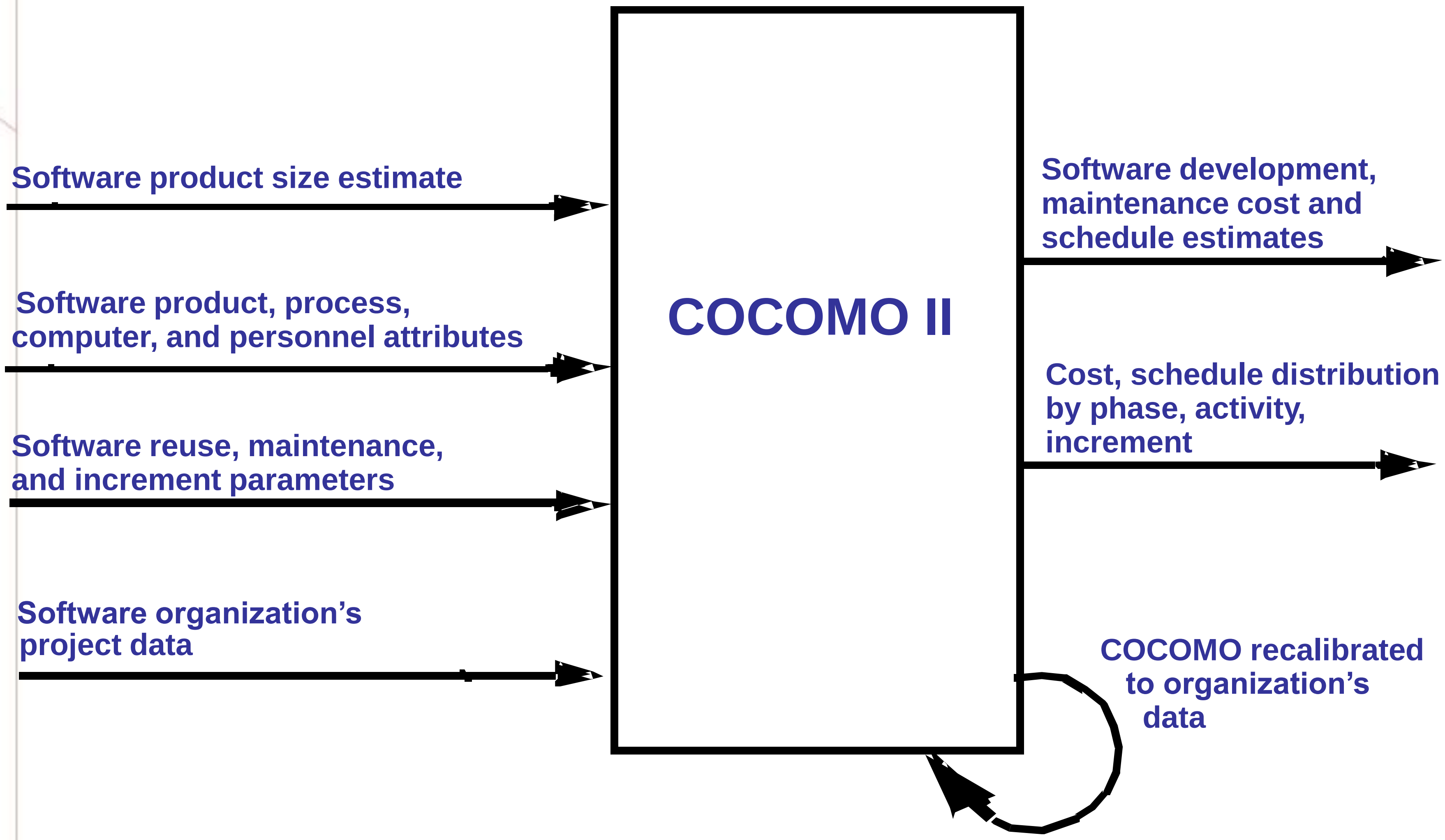
A tribe in Kenya

What is COCOMO?



"COCOMO (COnstructive COst MOdel) is a model designed by Barry Boehm to give an estimate of the number of programmer-months it will take to develop a software product."

COCOMO II Overview - I



COCOMO II Overview - II

- Open interfaces and internals
  - Published in Software Cost Estimation with COCOMO II, Boehm et. al., 2000
    - COCOMO – Software Engineering Economics , Boehm, 1981

- Numerous Implementation, Calibrations, Extensions
  - Incremental Development, Ada, new environment technology
  - Arguably the most frequently-used software cost model worldwide

List of COCOMO II

- USC COCOMO II.2000 - USC
- Costar – Softstar Systems
- ESTIMATE PROFESSIONAL – SPC
- CostXpert – Marotz
- Partial List of COCOMO Packages (STSC, 1993)
  - CB COCOMO, GECOMO Plus, COCOMOID, GHL COCOMO, COCOMO1, REVIC, CoCoPro, SECOMO, COSTAR, SWAN, COSTMODL

COCOMO II User Objectives

- Making investment or other financial decisions involving a software development
- Setting project budgets and schedules as a basis for planning and control
- Deciding on or negotiating tradeoffs among software cost, schedule, functionality, performance or quality factors
- Making software cost and schedule risk management decisions
- Deciding which parts of a software system to develop, reuse, lease or purchase
- Making legacy software inventory decisions: what parts to modify, phase out, outsource, etc.
- Setting mixed investment strategies to improve your organization's software capability, via reuse, tools, process maturity, outsourcing, etc.
- Deciding how to implement a process improvement strategy

COCOMO II Objectives

- Provide accurate cost and schedule estimates for both current and likely future software projects.
- Enabling organizations to easily recalibrate, tailor or extend COCOMO II to better fit their unique situations.
- Provide careful, easy-to-understand definition of the model's inputs, outputs and assumptions.
- Provide constructive model.
- Provide a normative model.
- Provide evolving model.

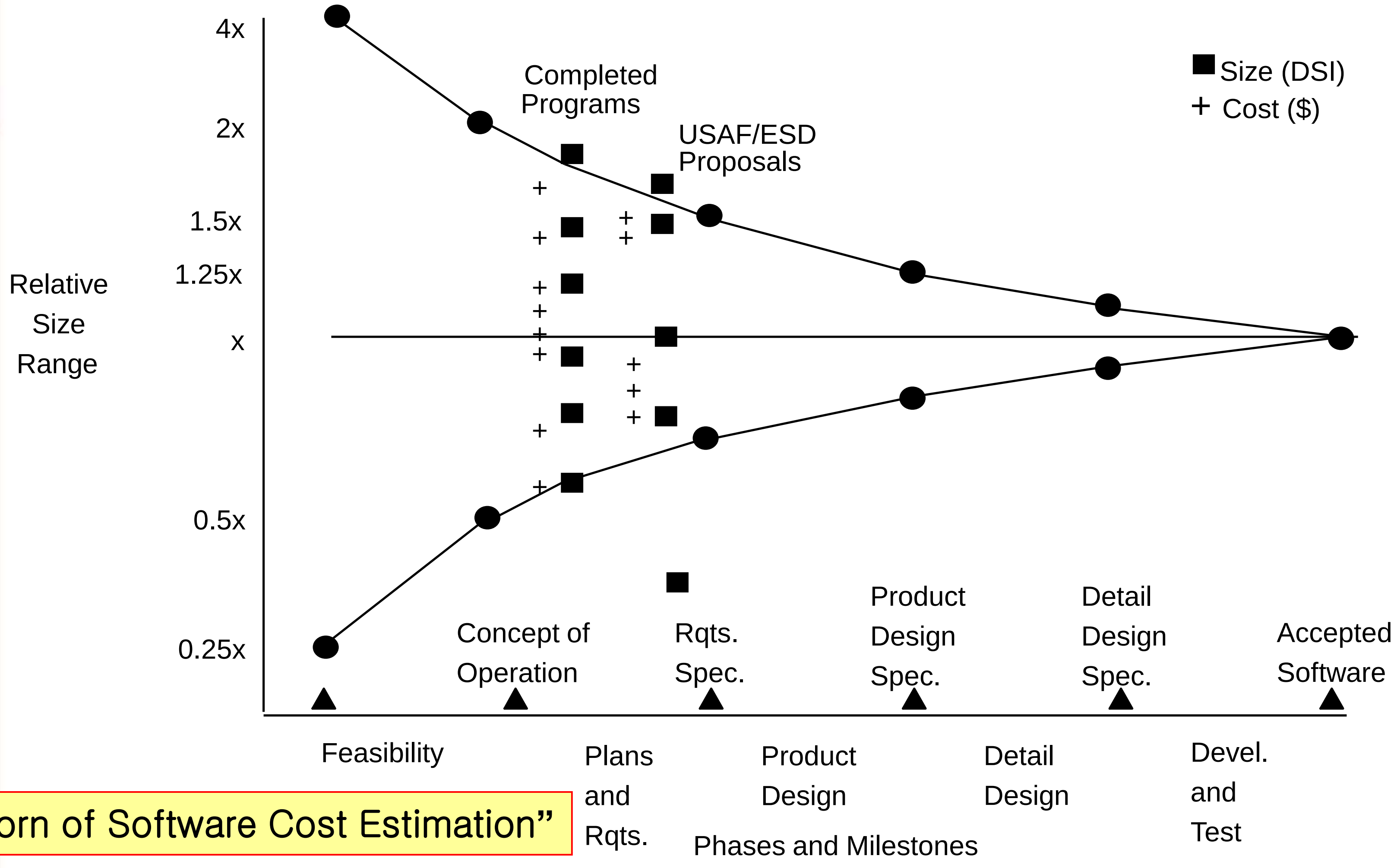
COCOMO II Evolution

- Proceed incrementally
  - Estimation issues of most importance and tractability w.r.t modeling, data collection, and calibration.
- Test the models and their concepts on first-hand experience
  - Use COCOMO II in annual series of USC Digital Library projects
- Establish a COCOMO II Affiliates' program
  - Enabling us to draw on the prioritized needs, expertise, and calibration data of leading software organizations

COCOMO II Evolution

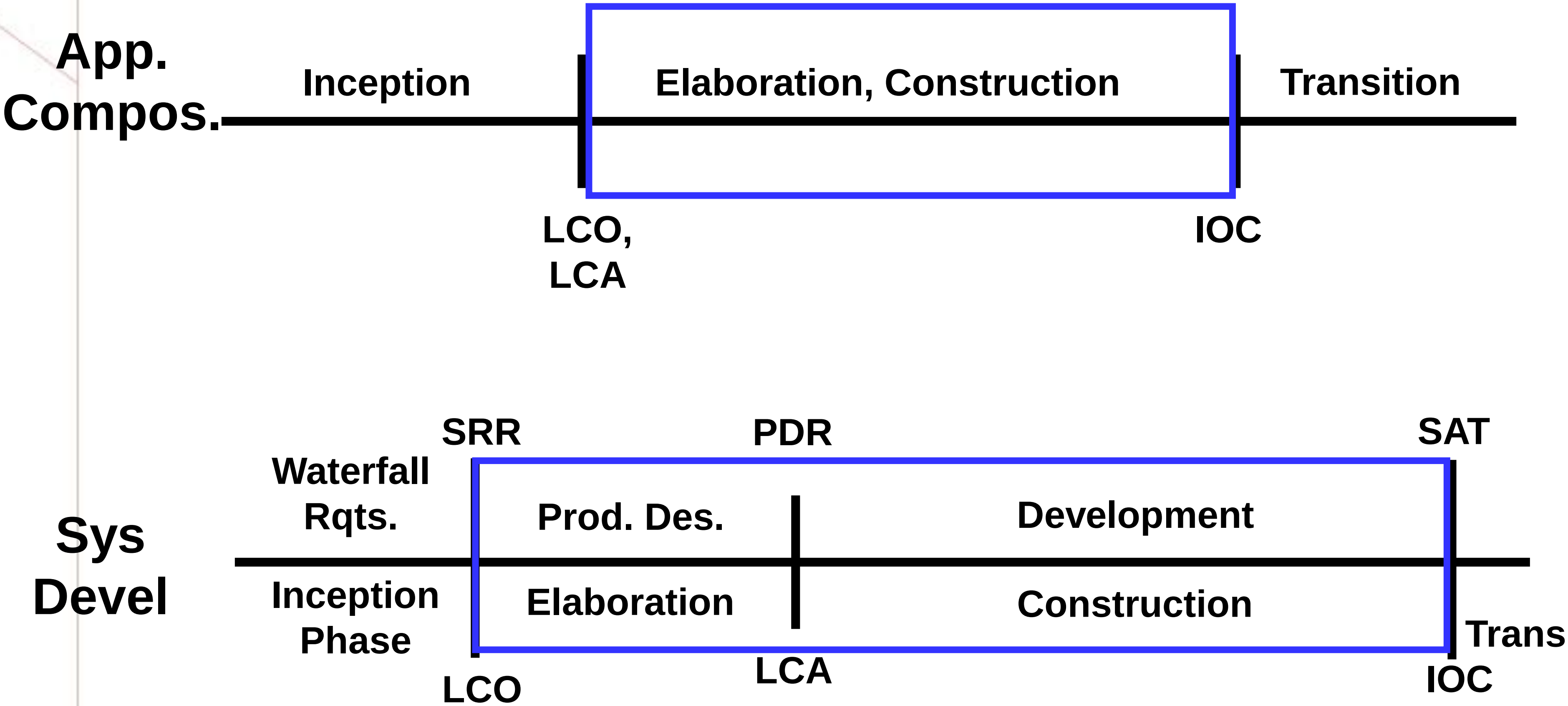
- Provide an externally and internally open model.
- Avoid unnecessary incompatibilities with COCOMO 81.
- Experiment with a number of model extensions.
- Balanced expert- and data- determined modeling.
- Develop a sequence of increasingly accurate models.
- Key the COCOMO II models to projections of future software life cycle practices.

S/W Estimation Accuracy vs. Phase



"Corn of Software Cost Estimation"

MBASE/Rational Anchor Point Milestones



Application Composition

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

- Responses:
  - Application-Point Sizing and Costing Model
  - Reporting estimate ranges rather than point estimate

Application Point Estimation Procedure

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

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

For Screens				For Reports			
# and source of data tables				# and source of data tables			
Number of Views Contained	Total < 4 (<2 srvr, <3 clnt)	Total <8 (<3 srvr, 3 - 5 clnt)	Total 8+ (>3 srvr, >5 clnt)	Number of Sections Contained	Total < 4 (<2 srvr, <3 clnt)	Total <8 (<3 srvr, 3 - 5 clnt)	Total 8+ (>3 srvr, >5 clnt)
<3	simple	simple	medium	0 or 1	simple	simple	medium
3-7	simple	medium	difficult	2 or 3	simple	medium	difficult
>8	medium	difficult	difficult	4+	medium	difficult	difficult

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.

Element Type	Complexity-Weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL Component			10

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

Step 5: Estimate percentage of reuse you expect to be achieved in this project. Compute the New Application Points to be developed *NAP =(Application-Points) (100-%reuse) / 100.*

Step 6: Determine a productivity rate, PROD=NAP/person-month, from the following scheme:

Developer's experience and capability	Very Low	Low	Nominal	High	Very High
ICASE maturity and capability	Very Low	Low	Nominal	High	Very High
PROD	4	7	13	25	50

Step 7: Compute the estimated person-months: PM=NAP/PROD.

Sizing Methods

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

Source Lines of Code

- Best Source : Historical data form previous projects
- Expert-Judged Lines of Code
- Expressed in thousands of source lines of code (KSLOC)
- Difficult Definition – Different Languages
- COCOMO II uses 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

Definition Checklist for Source Statements Counts							
Definition name:	Logical Source Statements (basic definition)			Date:_______________ Originator: COCOMO II			
Measurement unit:			**Physical source lines**				
			Logical source statements	√			
Statement type	**Definition**	√	**Data Array**			**Includes**	**Excludes**
When a line or statement contains more than one type, classify it as the type with the highest precedence.							
1 Executable		**Order of precedence:**			1	√	
2 Nonexecutable							
3 Declarations					2	√	
4 Compiler directives					3	√	
5 Comments							
6 On their own lines					4		√
7 On lines with source code					5		√
8 Banners and non-blank spacers					6		√
9 Blank (empty) comments					7		√
10 Blank lines					8		√
How produced	**Definition**	√	**Data array**			**Includes**	**Excludes**
1 Programmed						√	
2 Generated with source code generators							√
3 Converted with automated translators						√	
4 Copied or reused without change						√	
5 Modified						√	
6 Removed							√
Origin	**Definition**	√	**Data array**			**Includes**	**Excludes**
1 New work: no prior existence						√	
2 Prior work: taken or adapted from							
3 A previous version, build, or release						√	
4 Commercial, off-the-shelf software (COTS), other than libraries							√
5 Government furnished software (GFS), other than reuse libraries							√
6 Another product							√
7 A vendor-supplied language support library (unmodified)							√
8 A vendor-supplied operating system or utility (unmodified)							√
9 A local or modified language support library or operating system							√
10 Other commercial library							√
11 A reuse library (software designed for reuse)						√	
12 Other software component or library						√	
Usage	**Definition**	√	**Data array**			**Includes**	**Excludes**

Unadjusted Function Points

- 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.

Unadjusted Function Points

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.

For ILF and EIF				For EO and EQ				For EI			
Record Elements	Data Elements			File Types	Data Elements			File Types	Data Elements		
	1-19	20-50	51+		1-5	6-19	20+		1-4	5-15	16+
1	**Low**	**Low**	**Avg**	**0 or 1**	**Low**	**Low**	**Avg**	**0 or 1**	**Low**	**Low**	**Avg**
2-5	**Low**	**Avg**	**High**	**2-3**	**Low**	**Avg**	**High**	**2-3**	**Low**	**Avg**	**High**
6+	**Avg**	**High**	**High**	**4+**	**Avg**	**High**	**High**	**4+**	**Avg**	**High**	**High**

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.

Function Type	Complexity Weight		
	Low	**Average**	**High**
Internal Logical File (ILF)	**7**	**10**	**15**
External Interface Files (EIF)	**5**	**7**	**10**
External Inputs (EI)	**3**	**4**	**6**
External Outputs	**4**	**5**	**7**
External Inquiries	**3**	**4**	**6**

Step 4. Compute Unadjusted Function Points. Add all the weight functions counts to get one number, the Unadjusted Function Points.

Relating UFPs to SLOC

- USC-COCOMO II
  - Use conversion table (Backfiring) to convert UFPS into equivalent SLOC
  - Support 41 implementation languages and USR1-5 for accommodation of user's additional implementation languages
  - Additional Ratios and Updates : http://www.spr.com/library/0Langtbl.htm

Language	SLOC/UFP	Language	SLOC/UFP
Access	38	Jovial	107
Ada 83	71	Lisp	64
Ada 95	49	Machine Code	640
.		.	
.		.	
.		USR_1	1
.		USR_2	1
.		.	
.		.	
.		.	

Exercise - I

- Suppose you are developing a stand-alone application composed of 2 modules for a client
  - Module 1 written in C
    - FP multiplier C ➔ 128
  - Module 2 written in C++
    - FP multiplier C++ ➔ 53
- Determine UFP's and equivalent SLOC

Information on Two

Function Type	Low	Average	High
Internal Logical Files	7	10	15
External Interface	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

FP default weight values

Module 1

ype	Complexity Weight		
	Low	Average	High
Internal Logical File (ILF)	0	1	0
External Interface Files (EIF)	2	0	0
External Inputs (EI)	0	0	3
External Outputs	0	1	0
External Inquiries	0	0	2

Function Type	Complexity Weight		
	Low	Average	High
Internal Logical File (ILF)	2	0	0
External Interface Files (EIF)	0	5	0
External Inputs (EI)	0	4	0
External Outputs	0	2	0
External Inquiries	0	0	10

Module 2

Early Design & Post-Architecture Models

$$PM_{NS} = A \times Size^{E} \times \prod_{i=1}^{n} EM_i$$

$$\text{where } E = B + 0.01 \times \sum_{j=1}^{5} SF_j$$

$$TDEV_{NS} = C \times (PM_{NS})^{F}$$

$$\text{where } F = D + 0.2 \times 0.01 \times \sum_{j=1}^{5} SF_j$$

$$= D + 0.2 \times (E - B)$$

A = 2.94 B = 0.91

C = 3.67 D = 0.28

- Early Design Model [6 EMs]:
- Post Architecture Model [16 EMs]:
  *Exclude SCED driver

EMs: Effort multipliers to reflect characteristics of particular software under development

A : Multiplicative calibration variable

E : Captures relative (Economies/Diseconomies of scale)

SF: Scale Factors

Scale Factors & Cost Drivers

- Project Level – 5 Scale Factors
  - Used for both ED & PA models
- Early Design – 7 Cost Drivers
- Post Architecture – 17 Cost Drivers
  - Product, Platform, Personnel, Project

Project Scale Factors - I

$$PM = A \times (Size)^{E} \times \prod_{i=1}^{n} EM_i$$

where $A = 2.94$ (for COCOMO II.2000)

- Relative economies or diseconomies of scale
  - E < 1.0 : economies of scale
    - Productivity increase as the project size increase
    - Achieved via project specific tools (e.g., simulation, testbed)
  - E = 1.0 : balance
    - Linear model : often used for cost estimation of small projects
  - E > 1.0 : diseconomies of scale
    - Main factors : growth of interpersonal communication overhead and growth of large-system overhead

Project Scale Factors - II

Scale Factors (SF_i)	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedente	largely unprecedente	somewhat unprecedente	generally familiar	largely familiar	throughly familiar
	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
	5.07	4.05	3.04	2.03	1.01	0.00
RESL	little (20%)	some (40%)	often (60%)	generally(75%)	mostly (90%)	full (100%)
	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
	5.48	4.28	3.29	2.19	1.10	0.00
PMAT	SW-CMM Level 1 Lower	SW-CMM Level 1 Upper	SW-CMM Level 2	SW-CMM Level 3	SW-CMM Level 3	SW-CMM Level 5
	Or the Estimated Process Maturity Level (EPML)					
	7.80	6.24	4.68	3.12	1.56	0.00

PMAT == EPML

- EPML (Equivalent Process Maturity Level)

Key Process Areas	Almost Always (>90%)	Frequently (60-90%)	About Half (40-60%)	Occasionally (10-40%)	Rarely If Ever (<10%)	Does Not Apply	Don't Know
1 Requirements Management	❑	❑	❑	❑	❑	❑	❑
2 Software Project Planning	❑	❑	❑	❑	❑	❑	❑
3 Software Project Tracking and Oversight	❑	❑	❑	❑	❑	❑	❑
4 Software Subcontract Management	❑	❑	❑	❑	❑	❑	❑

⋮

$$EPML = 5 \times \left(\sum_{i=1}^{n} \frac{KPA\%_i}{100} \right) \times \frac{1}{n}$$

PA Model – Product EMs

Effort Multiplier	Very Low	Low	Nominal	High	Very High	Extra High
RELY	slight inconven-ience	low, easily recoverable losses	moderate, easily recoverable losses	high financial loss	risk to human life	
	0.82	0.92	1.00	1.10	1.26	n/a
DATA		DB bytes/Pgm SLOC < 10	10 <= D/P < 100	100 <= D/P < 1000	D/P>=1000	
	n/a	0.90	1.00	1.14	1.28	n/a
RUSE		none	across project	across program	across product line	across multiple product lines
	n/a	0.95	1.00	1.07	1.15	1.24
DOCU	Many life-cycle needs uncovered	Some life-cycle needs uncovered.	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very excessive for life-cycle needs	
	0.81	0.91	1.00	1.11	1.23	n/a
CPLX	See CPLX table					
	0.73	0.87	1.00	1.17	1.34	1.74

PA Model - CPLX

Effort Multiplier	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations	User Interface Management Operations
Very Low	Straight-line code with a few non-nested structured programming operators: DOs, CASEs, IF-THEN-ELSEs. Simple module composition via procedure calls or simple scripts.	Evaluation of simple expressions: e.g., A=B+C*(D-E)	Simple read, write statements with simple formats.	Simple arrays in main memory. Simple COTS-DB queries, updates.	Simple input forms, report generators.
Low	…	…	…	…	…
Nominal	Mostly simple nesting. Some intermodule control. Decision tables. Simple callbacks or message passing, including middleware-supported distributed processing	Use of standard math and statistical routines. Basic matrix/vector operations.	I/O processing includes device selection, status checking and error processing.	Multi-file input and single file output. Simple structural changes, simple edits. Complex COTS-DB queries, updates.	Simple use of widget set.
High	…	…	…	…	…
Very High	…	…	…	…	…
Extra High	Multiple resource scheduling with dynamically changing priorities. Microcode-level control. Distributed hard real-time control.	Difficult and unstructured numerical analysis: highly accurate analysis of noisy, stochastic data. Complex parallelization.	Device timing-dependent coding, micro-programmed operations. Performance-critical embedded systems.	Highly coupled, dynamic relational and object structures. Natural language data management.	Complex multimedia, virtual reality, natural language interface.

PA Model – Platform EMs

Effort Multiplier	Very Low	Low	Nominal	High	Very High	Extra High
TIME			≤ 50% use of available execution time	70% use of available execution time	85% use of available execution time	95% use of available execution time
	n/a	n/a	1.00	1.11	1.29	1.63
STOR			≤ 50% use of available storage	70% use of available storage	85% use of available storage	95% use of available storage
	n/a	n/a	1.00	1.05	1.17	1.46
PVOL		Major change every 12 mo.;	Major: 6 mo.; Minor: 2 wk.	Major: 2 mo.;Minor: 1	Major: 2 wk.;Minor: 2	
	n/a	0.87	1.00	1.15	1.30	n/a

PA Model – Personnel EMs

Effort Multiplier	Very Low	Low	Nominal	High	Very High	Extra High
ACAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
	1.42	1.19	1.00	0.85	0.71	n/a
PCAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
	1.34	1.15	1.00	0.88	0.76	n/a
PCON	48% / year	24% / year	12% / year	6% / year	3% / year	
	1.29	1.12	1.00	0.90	0.81	n/a
APEX	<= 2 months	6 months	1 year	3 years	6 years	
	1.22	1.10	1.00	0.88	0.81	n/a
LTEX	<= 2 months	6 months	1 year	3 years	6 year	
	1.20	1.09	1.00	0.91	0.84	n/a
PLEX	<= 2 months	6 months	1 year	3 years	6 year	
	1.19	1.09	1.00	0.91	0.85	n/a

PA Model – Project EMs

Effort Multiplier	Very Low	Low	Nominal	High	Very High	Extra High
TOOL	edit, code, debug	simple, frontend, backend CASE, little integration	basic life-cycle tools, moderately integrated	strong, mature life-cycle tools, moderately integrated	strong, mature, proactive life-cycle tools, well integrated with processes, methods, reuse	
	1.17	1.09	1.00	0.90	0.78	n/a
SITE	Inter-national	Multi-city and Multi-company	Multi-city or Multi-company	Same city or metro. area	Same building or complex	Fully collocated
	Some phone, mail	Individual phone, FAX	Narrow band email	Wideband electronic communication.	Wideband elect. comm., occasional video conf.	Interactive multimedia
	1.22	1.09	1.00	0.93	0.86	0.80
SCED	75% of nominal	85% of nominal	100% of nominal	130% of nominal	160% of nominal	
	1.43	1.14	1.00	1.00	1.00	n/a

ED EMs vs. PA EMs

Early Design Cost Driver	Counterpart Combined Post-Architecture Cost Drivers
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE (Same as P-A RUSE)
PDIF	TIME, STOR, PVOL
PERS	ACAP, PCAP, PCON
PREX	APEX, PLEX, LTEX
FCIL	TOOL, SITE
SCED	SCED (Same as P-A SCED)

ED Model EMs - RCPX

RCPX Descriptors:	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Sum of RELY, DATA, CPLX, DOCU Ratings	5, 6	7, 8	9 - 11	12	13 - 15	16 - 18	19 - 21
Emphasis on reliability, documentation	Very Little	Little	Some	Basic	Strong	Very Strong	Extreme
Product complexity	Very simple	Simple	Some	Moderate	Complex	Very complex	Extremely complex
Database size	Small	Small	Small	Moderate	Large	Very Large	Very Large
Effort Multiplier	0.49	0.60	0.83	1.00	1.33	1.91	2.72

ED Model EMs - PDIF

PDIF Descriptors:	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Sum of TIME, STOR, and PVOL ratings	8	9	10 - 12	13 - 15	16, 17	Sum of TIME, STOR, and PVOL	8
Time and storage constraint	<=50%	<= 50%	65%	80%	90%	Time and storage constraint	? 50%
Platform volatility	Very stable	Stable	Somewhat volatile	Volatile	Highly volatile	Platform volatility	Very stable
Effort Multiplier	0.87	1.00	1.29	1.81	2.61	0.87	1.00

ED Model EMs - PERS

PERS Descriptors:	**Extra Low**	**Very Low**	**Low**	Nominal	High	Very High	Extra High
Sum of ACAP, PCAP, PCON Ratings	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
Combined ACAP and PCAP	20%	35%	45%	55%	65%	75%	85%
Annual Personnel	45%	30%	20%	12%	9%	6%	4%
Effort Multiplier	2.12	1.62	1.26	1.00	0.83	0.63	0.50

ED Model EMs - PREX

PREX Descriptors:	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Sum of APEX, PLEX, and LTEX ratings	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
Applications, Platform, Language and Tool Experience	<= 3 mo.	5 months	9 months	1 year	2 years	4 years	6 years
Effort Multiplier	1.59	1.33	1.22	1.00	0.87	0.74	0.62

ED Model EMs - FCIL

FCIL Descriptors:	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Sum of TOOL and SITE ratings	2	3	4, 5	6	7, 8	9, 10	11
TOOL support	Minimal	Some	Simple CASE tool collection	Basic life-cycle tools	Good; moderately integrated	Strong; moderately integrated	Strong; well integrated
Multisite conditions	Weak support of complex multisite development	Some support of complex M/S devel.	Some support of moderately complex M/S devel.	Basic support of moderately complex M/S devel.	Strong support of moderately complex M/S devel.	Strong support of simple M/S devel.	Very strong support of collocated or simple M/S devel.
Effort Multiplier	1.43	1.30	1.10	1.0	0.87	0.73	0.62

Calibration & Prediction Accuracy

COCOMO Calibration

	COCOMO 81	COCOMO II.1997	COCOMO II.2000
Project Data Points	63	83	161
Calibration		10% Data, 90% Expert	Bayesian

MRE: PRED (.30) Values

	COCOMO 81	COCOMO II.1997	COCOMO II.2000
Effort - By Organization	81%	52% 64%	75% 80%
Schedule - By Organization	65%	61% 62%	72% 81%

COCOMO II Family

Model	No. of Drivers		Sizing
	Environment	Process	
Application Composition	**2**	**0**	**Application Points**
Early Design	**7**	**5**	**Function Points or SLOC**
Post Architecture	**17**	**5**	**Function Points or SLOC**
COCOMO81	**15**	**1**	**SLOC (FP Extension)**

COCOMO Model

	COCOMO	Ada COCOMO	COCOMO II: Application Composition	COCOMO II: Early Design	COCOMO II: Post-Architecture
Size	Delivered Source Instructions (DSI) or Source Lines of Code (SLOC)	DSI or SLOC	Application Points	Function Points (FP) and Language or SLOC	FP and Language or SLOC
Reuse	Equivalent SLOC = Linear *f*(DM,CM,IM)	Equivalent SLOC = Linear *f*(DM,CM,IM)	Implicit in Model	Equivalent SLOC = nonlinear *f*(AA, SU,UNFM,DM,CM,IM)	Equivalent SLOC = nonlinear *f*(AA, SU,UNFM,DM,CM,IM)
Rqts. Change	Requirements Volatility rating: (RVOL)	RVOL rating	Implicit in Model	Change % : RQEV	RQEV
Maintenance	Annual Change Traffic (ACT) = %added + %modified	ACT	Object Point ACT	*f*(ACT,SU,UNFM)	*f*(ACT,SU,UNFM)
Scale (b) in $MM_{NOM}=a(Size)^b$	Organic: 1.05 Semidetached: 1.12 Embedded: 1.20	Embedded: 1.04 -1.24 depending on degree of: • early risk elimination • solid architecture • stable requirements • Ada process maturity	1.0	.91-1.23 depending on the degree of: • precedentedness • conformity • early architecture, risk resolution • team cohesion • process maturity (SEI)	.91-1.23 depending on the degree of: • precedentedness • conformity • early architecture, risk resolution • team cohesion • process maturity (SEI)
Product Cost Drivers	RELY, DATA, CPLX	RELY*, DATA, CPLX*, RUSE	None	RCPX*■, RUSE*■	RELY, DATA, DOCU*■, CPLX■, RUSE*■
Platform Cost Drivers	TIME, STOR, VIRT, TURN	TIME, STOR, VMVH, VMVT, TURN	None	Platform difficulty: PDIF *■	TIME, STOR, PVOL(=VIRT)
Personnel Cost Drivers	ACAP, AEXP, PCAP, VEXP, LEXP	ACAP*, AEXP, PCAP*, VEXP, LEXP*	None	Personnel capability and experience: PERS*■, PREX*■	ACAP*, AEXP■, PCAP*, PEXP *■, LTEX*■, PCON *■
Project Cost Drivers	MODP, TOOL, SCED	MODP*, TOOL*, SCED, SECU	None	SCED, FCIL*■	TOOL*■, SCED, SITE*■

* Different Multipliers
■ Different Rating Scale

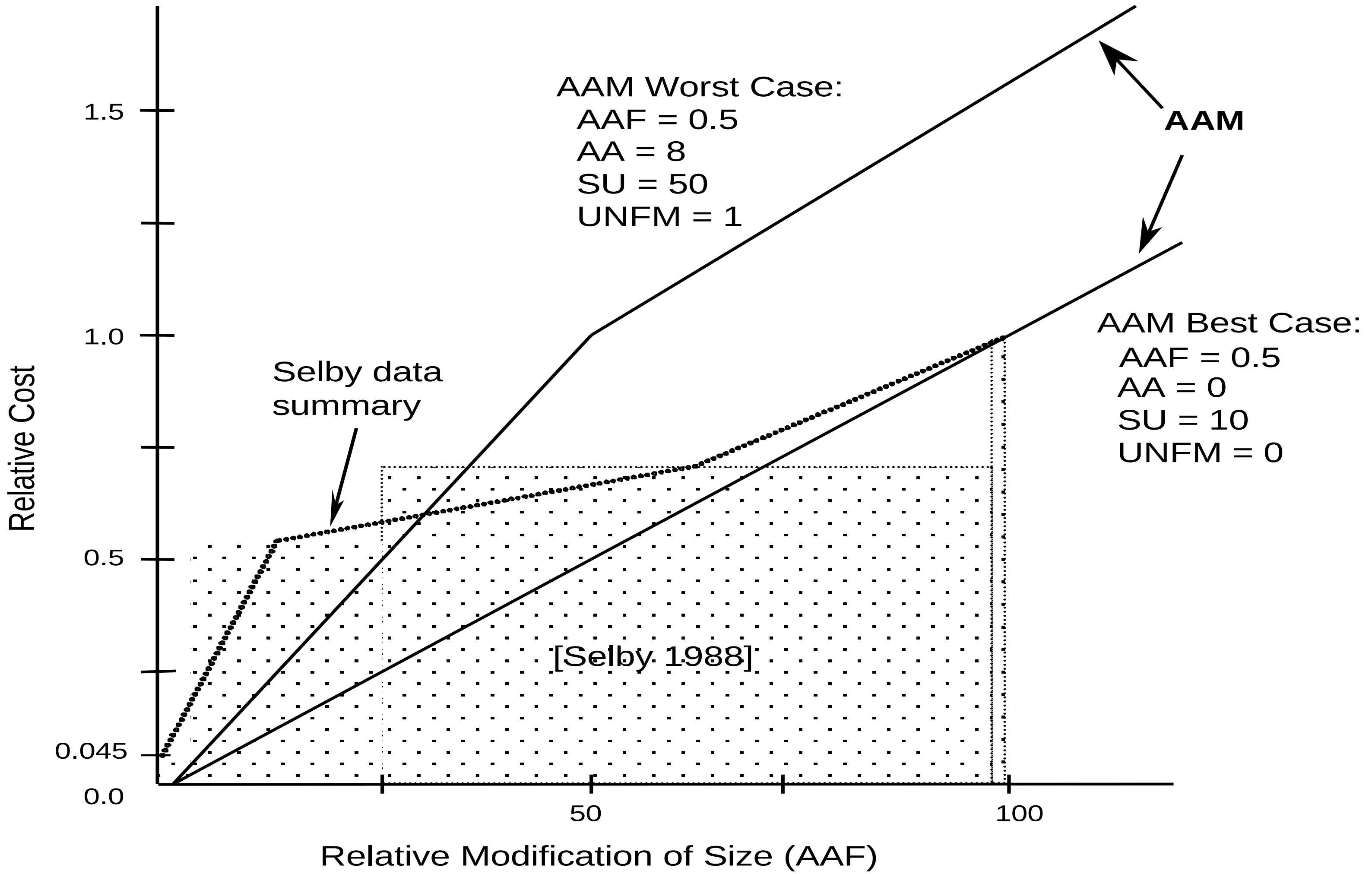
USC-COCOMO II.2000

Reuse & Product Line

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

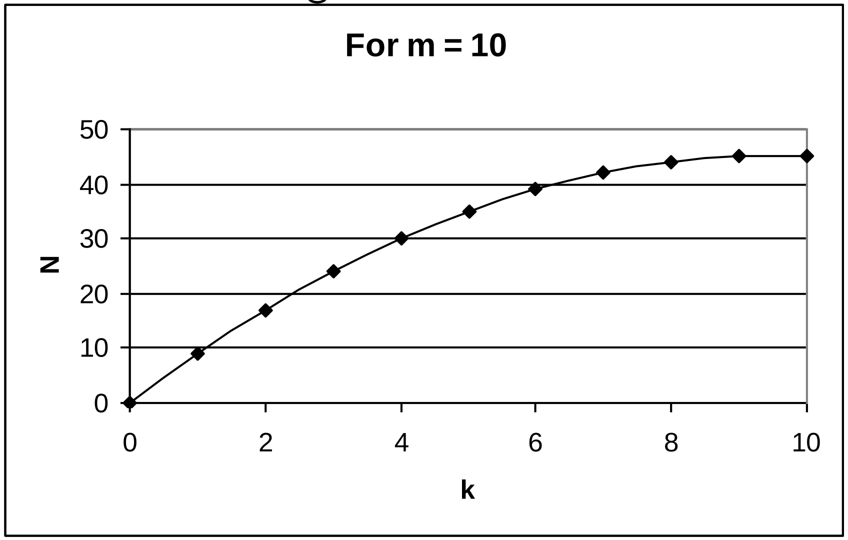
- Responses
  - New nonlinear reuse model for effective size
  - Cost of developing reusable software estimated by RUSE effort multiplier
  - Gathering schedule data

Non-Linear Reuse Effect



Primary Cost Factors for Reuse (NASA)

- Cost of Understanding
  - 47% of the effort in SW maintenance involves understanding the SW to be modified [Parikh-Zvegintzov 1983]
- Relative cost of Checking Module Interfaces
  - Relation b/w no. of modified modules and no. of module interface checking [Gerlich-Denskat 1994]



$$N = k \times (m - k) + k \times \left(\frac{k-1}{2}\right)$$

N: number of module interface checks required
m: number of modules for reuse
k: number of modules modified

COCOMO II Reuse Model

- Add Assessment & Assimilation increment (AA)
  - – Similar to conversion planning increment
- Add software understanding increment (SU)
  - To cover nonlinear software understanding effects
  - Coupled with software unfamiliarity level (UNFM)
  - Apply only if reused software is modified

$$AAF = (0.4 \times DM) + (0.3 \times CM) + (0.3 \times IM)$$

$$AAM = \begin{cases} \dfrac{[AA + AAF(1 + (0.02 \times SU \times UNFM))]}{100}, & \text{for } AAF \leq 50 \\ \dfrac{[AA + AAF + (SU \times UNFM)]}{100}, & \text{for } AAF > 50 \end{cases}$$

$$\text{Equivalent KSLOC} = \text{Adapted KSLOC} \times \left(1 - \frac{AT}{100}\right) \times AAM$$

Software Understanding

SU	**Very Low**	**Low**	Nominal	High	Very High
Structure	Very low cohesion, high coupling, spaghetti code.	Moderately low cohesion, high coupling.	Reasonably well-structured; some weak areas.	High cohesion, low coupling.	Strong modularity, information hiding in data / control structures.
Application Clarity	No match between program and application world-views.	Some correlation between program and application.	Moderate correlation between program and application.	Good correlation between program and application.	Clear match between program and application world-views.
Self-Descriptive-ness	Obscure code; documentation missing, obscure or obsolete	Some code commentary and headers; some useful documentation.	Moderate level of code commentary, headers, documentation.	Good code commentary and headers; useful documentation; some weak areas.	Self-descriptive code; documentation up-to-date, well-organized, with design rationale.
SU Increment to ESLOC	50	40	30	20	10

Assessment and Assimilation (AA)

AA Increment	Level of AA Effort
0	None
2	Basic module search and documentation
4	Some module Test and Evaluation (T&E), documentation
6	Considerable module T&E, documentation
8	Extensive module T&E, documentation

Unfamiliarity (UNFM)

UNFM Increment	Level of Unfamiliarity
0.0	Completely familiar
0.2	Mostly familiar
0.4	Somewhat familiar
0.6	Considerably familiar
0.8	Mostly unfamiliar
1.0	Completely unfamiliar

Guidelines for Quantifying Adapted

Code Category	DM	CM	IM	AA	SU	UNFM
New - all original software	not applicable					
Adapted - changes to preexisting software	0% - 100% normally > 0%	0^{+}% - 100% usually > DM and must be > 0%	0% - 100+% IM usually moderate and can be > 100%	0% – 8%	0% - 50%	0 - 1
Reused - unchanged existing software	0%	0%	0% - 100% rarely 0%, but could be very small	0% – 8%	not applicable	
COTS - off-the-shelf software (often requires new glue code as a wrapper around the COTS)	0%	0%	0% - 100%	0% – 8%	not applicable	

Requirement Evolution & Volatility (REVL)

- Adjust the effective size of the product
  - Causal factors: user interface evolution, technology upgrades, or COTS volatility

- Percentage of code discarded due to requirement evolution
  - E.g., SLOC = 100K and REVL = 20
    - Project effective size = 120K

$$Size = \left(1 + \frac{REVL}{100}\right) \times Size_D$$

Example: Manufacturing Control System

- Reused Code: 100 SLOC
- Full Cost: $2.94(100)^{1.10}$ (1.18) ($8K/PM) = $4400K
- International Factory Reuse: halfway between VH and XH
- Recommended Reliability rating: 1 level lower
- Recommended Documentation rating: High
- Develop for Reuse: $4400 (1.195)(1.18)(1.11) = $6824K

Effort Multipliers	**Very Low**	**Low**	Nominal	High	Very High	Extra High
Developed for Reuse		.95	1.00	1.07	1.15	1.24
Required Reliability	0.82	0.92	1.00	1.10	1.26	
Required Documentation	0.81	0.91	1.00	1.11	1.23	

Subsequent Development w/ Reuse

- Black-box plug-and-play: 30 KSLOC
- Reuse with modifications: 30 KSLOC
- New factory-specific SW: 40 KSLOC
- Assessment and assimilation (AA): 2%
- Software understanding factor (SU): 10%
- Unfamiliarity factor (UNFM): 0.3
- % design modified (DM): 10%
- % code modified (CM): 20%
- % integration modified (IM): 20%
- $AAF = \frac{.4(10) + .3 (20) + .3 (20)}{100} = .16$
- ESLOC = 40 + (30) (.02) + (30) (.02 + (.3) (.1) + .16)
- = 40 + 0.6 + 6.3 = 46.9
- COST = 2.94 $(46.9)^{1.10}$ (1.18) (1.195) (1.18) (1.1) ($8K) = $2966K

Reuse vs. Redevelopment

Number of Factories	**Redevelopment Cost**	**Product Line cost**	Investment Return
1	$4,400	$6,824	-$2,424
2	$8,800	$9,790	-$990
3	$13,200	$12,755	$444
4	$17,600	$15,722	$1,878

Q & A