

Software Cost Estimating

*Techniques for estimating in a
software development
environment*

“Any sufficiently advanced technology is
indistinguishable from magic.”
- Arthur C. Clarke

Acknowledgments




- ICEAA is indebted to TASC, Inc., for the development and maintenance of the Cost Estimating Body of Knowledge (CEBoK®)
 - ICEAA is also indebted to Technomics, Inc., for the independent review and maintenance of CEBoK®
- ICEAA is also indebted to the following individuals who have made significant contributions to the development, review, and maintenance of CostPROF and CEBoK®
- Module 12 Software Cost Estimating
 - Lead authors: Belinda J. Nethery, Allison L. Horrigan
 - Assistant authors: Tara L. Eng, Heather F. Chelson
 - Senior reviewers: Richard L. Coleman, Michael A. Gallo, Fred K. Blackburn
 - Reviewer: Kenneth S. Rhodes
 - Managing editor: Peter J. Braxton




Unit Index

- Unit I - Cost Estimating
- Unit II - Cost Analysis Techniques
- Unit III - Analytical Methods
- Unit IV - Specialized Costing
 - 11. Manufacturing Cost Estimating
 - 12. **Software Cost Estimating**
- Unit V - Management Applications

Software Cost Estimating Overview

- | | |
|---|--|
| <ul style="list-style-type: none"> • Key Ideas <ul style="list-style-type: none"> - Cost Drivers <ul style="list-style-type: none"> • Size • Complexity • Capability - SLOC vs. ESLOC vs. Function Points - Development Methodologies | <ul style="list-style-type: none"> • Practical Applications <ul style="list-style-type: none"> - ESLOC Sizing - Software Effort Calculation <ul style="list-style-type: none"> • Capability Adjustments • Complexity Adjustments - Schedule Determination - Schedule Compression Factors |
| <ul style="list-style-type: none"> • Analytical Constructs <ul style="list-style-type: none"> - ESLOC Equation - COCOMO II CER Equation <div style="background-color: yellow; padding: 5px; margin: 10px 0;"> $PM = A \cdot Size^E \cdot \prod_{i=1}^n EM_i$ </div> <ul style="list-style-type: none"> - COCOMO II Schedule CER | <ul style="list-style-type: none"> • Related Topics <ul style="list-style-type: none"> - Costing Techniques  - Parametric Estimating  - Regression Analysis  |


Software Cost Estimating Outline

v1.2

- Core Knowledge
 - Software Overview
 - Software Development Approaches
 - Software Cost Drivers
 - Estimating Development Methodologies
 - Estimating Techniques Applied to Software
 - Challenges in Estimating Software
- Summary
- Resources
- Related and Advanced Topics

Introduction

v1.2

-  Software is a key component of almost every system including:
 - Custom Developed Software
 - Commercial-Off-The-Shelf (COTS) Software
 - Databases
 - Enterprise Resource Planning (ERP) Tools
- Software development is both an art and a science, as is *estimating* software development
- Using equations from COCOMO II developed by Barry Boehm in many of the examples
 - Leader in field of software cost estimation
 - Research publicly available in texts

Software - What We Do (and Don't) Know

- Software isn't easy to understand because it's not a tangible item
- Developing software can be extremely costly and time consuming
- "Chaos" has been downgraded
 - Standish Group's 1994 study was revisited in 2000
 - Examined IT developed software projects
 - Schedule overruns have significantly decreased from 222% over the original time estimates in 1994 down to 63% in 2000
 - Cost overruns have gone from 189% over the original cost estimates in 1994 down to 45% in the 2000 study
 - Better tools have been created to monitor and control progress
 - Better management processes have emerged

Extreme Chaos, copyright © 2001 The Standish Group International, Inc.

Software Development Process

- The same basic system engineering steps are followed when developing software as when developing a hardware system

Systems today usually consist of both hardware and software.

- **System Engineering Steps for Software**
 - Step 1: System Requirements and Design (both hardware and software)
 - Step 2: Software Requirements Analysis
 - Step 3: Software Preliminary and Detailed Design
 - Step 4: Code and Unit Test
 - Step 5: Unit Integration and Test
 - Step 6: Software System Test
 - Step 7: System Test (both hardware and software)

Coding is equivalent to building a piece of hardware

MIL STD 498, "Software Development and Documentation," December, 1994

- These steps provide a framework for structuring the Software WBS

WBS for Software Programs

Sample Partial WBS - This is an example, each WBS will have a unique mapping

- 1.1 System SE/PM (Includes Step 1)
- 1.2 System SIT&E (Includes Step 7)
- 1.3 Hardware
- 1.4 Software
 - 1.4.1 Build 1
 - 1.4.1.1 SE/PM (Includes Step 2 & 3)
 - 1.4.1.2 SIT&E (Includes Step 5 & 6)
 - 1.4.1.3 CSCI 1
 - 1.4.1.4 CSCI 2
 - 1.4.1.4.1 CSC 1 (Includes Step 4)
 - 1.4.1.4.2 CSC 2
 - 1.4.2 Build 2
 - 1.4.3 Build 3

- Framework to break large projects into product oriented elements and processes
- Used as a foundation for cost estimating, schedule planning, progress tracking, risk monitoring and many other management functions
- Dept of Defense mandates use of a WBS (guidance in Military Handbook 881A)
- Industry has no mandated standard; however, use of WBS recommended by IEEE

It is important that the cost analyst understand the content associated with a particular cost

IEEE/EIA 12207.2-1997, IEEE/EIA Guide, Industry Implementation of International Standard ISO/IEC 12207; 1995, Standard for Information Technology, Software Life Cycle Processes - Implementation Consideration, April 1998

Comparison to Hardware - Similarities

- Same basic development processes
 - Use same basic techniques for estimating (Analogy, Parametric, Build-Up) 2
- Both also have the same basic sustainment costs
 - Require Support, Maintenance, and Upgrades
- Factors that influence cost are similar
 - Size
 - Length, weight, volume, etc. vs. Source Lines Of Code (SLOC), Function Points, etc.
 - System Complexity
 - Development Capability (Personnel, Facilities, Tools, Etc.)

Comparison to Hardware - Differences

1

- You may build a piece of hardware over and over again; you build software only once
 - For hardware, you design, build, and test a system that you then build multiple times in Production
 - For software, you design, build, and test a system then simply generate a copy
- Hardware (and, therefore, hardware cost estimating) has been in existence for much longer than software (and, therefore, software cost estimating)
 - HW development processes are more mature and stable
 - Longer period over which to collect historical data and refine CERs
 - Software estimating techniques lag behind those of hardware
 - Harder to find good clean data
 - Less statistically based

Software Development Approaches

- Software Development Methodologies
 - Waterfall
 - Agile
 - Incremental
 - Evolutionary
 - Spiral
- Programming Paradigms
 - “Linear” (non Object-Oriented) vs. Object-Oriented (OO)

[SEI-CMM] *Capability Maturity Model for Software, Version 1.1*, Paulk, Mark C. et.al., Software Engineering Institute, Carnegie Mellon University, February 1993

Overview of Development Methodologies

v1.2

- Determines the sequencing of the steps of the Software Development Process
 - The Software Engineering Steps 1-7
- System level Requirements and Test are the first and last steps, but the sub-system building process can be:
 - A single effort (Waterfall)
 - Short iterations (Agile)
 - In series (Evolutionary)
 - In overlapping series (Incremental)
 - Include additional Risk and Analysis phases (Spiral)

12



Unit IV - Module 12

13

© 2002-2013 ICEAA. All rights reserved.

Programming Paradigms

v1.2

- **“Linear” Programming**
 - Every system is custom built line by line
 - Some ability to adapt code
 - Large systems have problems with standardization and modules not “fitting”
 - Examples include COBOL, FORTRAN
- **Object-Oriented (OO) Programming**
 - System made up of pre-built, standardized, interchangeable objects
 - Objects can be used in any system
 - Large systems don't have standardization or “fit” problems
 - Examples include Ada 95, C++

One man custom-building one gun from scratch

12

Object-Oriented Programming: An Evolutionary Approach, Brad J. Cox, Addison-Wesley, 1987

One man building one part of the gun to a specified standard - The gun is then assembled from interchangeable parts



Unit IV - Module 12

14

© 2002-2013 ICEAA. All rights reserved.



Example Scenario

- New mail order business
 - Expects significant growth over 5 years
 - Increase in customers, inventory, and personnel
- Want a system with necessary capability and minimal disruption of staff
 - System to be developed in 18 months
- Preliminary estimate of the system is that it will require 100,000 SLOC
 - The development will be divided into 3 CSCIs
 - CSCI 1 has 45,000 SLOC
 - CSCI 2 has 35,000 SLOC
 - CSCI 3 has 20,000 SLOC


Cost Drivers

- Size
- Complexity
- Capability

Cost Drivers Overview

- Cost Drivers are used to create a Cost Estimating Relationship (CER) between the drivers and the cost
 - Although it is generally agreed that these are the main cost drivers of software, the CERs based on the drivers differ
 - The COCOMO II CER is commonly used
- COCOMO II CER equation

$$PM = A \cdot \text{Size}^E \cdot \prod_{i=1}^n EM_i$$




Where: PM = Person Months 
 A = Constant = 2.94
 Size = SLOC in thousands (KSLOC)
 E = Sum of Scale Factors (Economies or Diseconomies of Scale)
 EM = Effort Multipliers

Size

Complexity and Capability

Software Cost Estimation with COCOMO II,
Boehm et al., Prentice Hall PTR, 2000

Cost Drivers - Size

-  Size is the primary cost driver of software development costs
-  Methods of measuring size include
 - Source Lines of Code (SLOC)
 - Equivalent Source Lines of Code (ESLOC)
 - Function Points
 -  Object Points

A good assessment of size is critical to a good estimate!



Source Lines of Code (SLOC)

v1.2

3

- Include executable instructions and data declarations
 - Do not include comments, blanks, and continuation lines



Warning: This is just one definition of SLOC, not *the* definition of SLOC.

- Can be accurately and consistently counted after completed development with automated counting tools



- Delivered source lines of code (DSLOC)

- Prior to development, must be estimated using standard estimating techniques

2

- Analogy is the most common

Guidelines for Successful Acquisition and Management of Software Intensive Systems: Weapon Systems, Command and Control Systems, Management Information Systems, Version 3.0, Dept of the Air Force, Software Technology Support Center, 2000

SLOC Issues

v1.2

- Advantages
 - Widely utilized in real-time systems and many legacy IT systems
 - Easily counted; can use automated counters
 - Less subjectivity in counting than with other measures
- Disadvantages
 - Wide discrepancies occur even with standard definitions
 - Logical vs. physical SLOC counts
 - Driven by language choices
 - Different software languages require a different number of lines of code for same function
 - Does not adequately address COTS-based systems

Counting Reusable Code

- Software is often a mix of new code and code developed in previous efforts

 - Reused code requires no modification

 - Adapted code requires some amount of redesign, recoding, and retesting

- Rework may be major or minor

Software Engineering Economics, Barry W. Boehm, Prentice Hall, 1981

- Software estimating models are usually based on new lines of developed code

- Provide input to models on amount of reused/adapted code; or...

- Calculate equivalent new source lines of code (ESLOC)



Warning: There are many terms and conventions to denote code from another source, so defining terms is crucial

Equivalent Source Lines of Code

-  • Equivalent Source Lines of Code (ESLOC)

4

- The effective size of reused and adapted code adjusted to its equivalent in new code + The size of the new code
- The adjustment is based on the additional effort it takes to modify the code for inclusion in the product

- ESLOC Equation from COCOMO

$$\text{ESLOC} = \text{SLOC} * [(40\% * \% \text{ Design Modified}) + (30\% * \% \text{ Code Modified}) + (30\% * \% \text{ Integration \& Test})]$$

Assumes: - 40% of effort is for design
- 30% of effort is for coding
- 30% of effort is for test

Example on the following slide

You may have to change percentages for your environment



Warning: Beware of claims that no testing will be required.

Software Engineering Economics, Barry W. Boehm, Prentice Hall, 1981



ESLOC Example

19

- Suppose from the Example Scenario that the code sizes given are Reused and Adapted Code
 - Find the ESLOC given:
 - Total Reused and Adapted Code = 100,000 SLOC
 - CSCI 1 - 45,000 SLOC; 20% retest
 - CSCI 2 - 35,000 SLOC; 80% redesign, 100% recode and retest
 - CSCI 3 - 20,000 SLOC; 50% recode and retest
 - Calculations:

$$\text{ESLOC} = \text{SLOC} * [(40\% * \% \text{ Design}) + (30\% * \% \text{ Code}) + (30\% * \% \text{ Test})]$$

$$\text{CSCI 1 - ESLOC} = 45,000 * [(40\% * 0\%) + (30\% * 0\%) + (30\% * 20\%)] = 2,700$$

$$\text{CSCI 2 - ESLOC} = 35,000 * [(40\% * 80\%) + (30\% * 100\%) + (30\% * 100\%)] = 32,200$$

$$\text{CSCI 3 - ESLOC} = 20,000 * [(40\% * 0\%) + (30\% * 50\%) + (30\% * 50\%)] = 6,000$$

$$\text{Total ESLOC} = 2,700 + 32,200 + 6,000 = 40,900$$

You may need to adjust the mix of total effort applied to design, code, and test for the project you are estimating. *Ask the engineers!*

Code Size Growth

- Delivered project is bigger than estimated
- Increase driven by:
 - Poor understanding of requirements initially
 - New requirements added during development
 - Underestimate of required SLOC
 - Code reuse optimism
- Key is to know the track record and account for expected growth
 - Some commercial tools have options for the confidence level of the size estimates
 - Use industry metrics to adjust



Warning: Beware requirements creep!

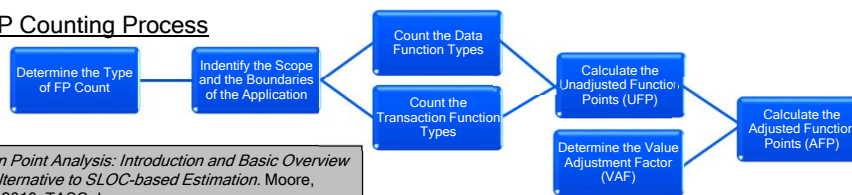
Function Points



18

- Considers the number of functions being developed based on the requirements specification
- The requirements of a system can be gathered from:
 - People: The Program's Primary Users, Program Developers, System Analysts, Project Managers
 - Documents: Architecture diagrams, Data models, Detailed design specifications and requirements, Business function/process models, User manuals, Screen prints, Function Point Counting Practices Manual
- FP Analysis can be performed with as many/few of these documents as long as sufficient understanding can be gained

FP Counting Process



Function Point Analysis: Introduction and Basic Overview as an Alternative to SLOC-based Estimation. Moore, Tucker. 2010, TASC, Inc.

Functions

- Transaction Files - Made up of the processes exchanged between the user, the internal files, and the external files
 - External Inputs (EI): User inputs that provide data
 - External Outputs (EO): Output to users such as reports, screens, error message
 - External Inquiries (EQ): Data sent to other applications
 - Each Transaction Function is broken down into File Types Referenced (FTRs) and then into Data Element Types (DETs)
- Data Functions - Made up of the Internal and External “resources” that affect the system
 - Internal Logical Files (ILF): Online input that results in software response
 - External Interface Files (EIF): Machine readable interfaces used to transmit information to another system (disks and tapes)
 - Each Data Function is broken down into Record Element Types (RETs) and then into Data Element Types (DETs)

Software Engineering, A Practitioner's Approach, 3rd ed, Roger S. Pressman, McGraw Hill, Inc., 1992

FP Calculations

- Tables are used to calculate the number of UFPs

EI Table

FTR's	DATA ELEMENTS		
	1-4	5-15	> 15
0-1	Low	Low	Ave
2	Low	Ave	High
3 or more	Ave	High	High

Shared EO & EQ Table

FTR's	DATA ELEMENTS		
	1-5	6-19	> 19
0-1	Low	Low	Ave
2-3	Low	Ave	High
> 3	Ave	High	High

UFP Conversion

Rating	VALUES		
	EO	EQ	EI
Low	4	3	3
Average	5	4	4
High	7	6	6

ILF/EIF Table

RET's	DATA ELEMENTS		
	1-19	20 - 50	> 50
1	Low	Low	Ave
2-5	Low	Ave	High
> 5	Ave	High	High

UFP Conversion

Rating	Values	
	ILF	EIF
Low	7	5
Average	10	7
High	15	10

- A Value Adjustment Factor (VAF) is then computed
 - Based on 14 general system characteristics (GSCs) that rate the general functionality of the application being counted by degree of influence (0-5)
 - Using the IFPUG Value Adjustment Equation: $VAF = 0.65 + [(Ci) / 100]$, where i = is from 1 to 14 representing each GSC
- The final Function Point Count is obtained by multiplying the VAF times the UAF: $FP = UAF * VAF$

Software Metrics, <http://www.softwaremetrics.com>, 2009.

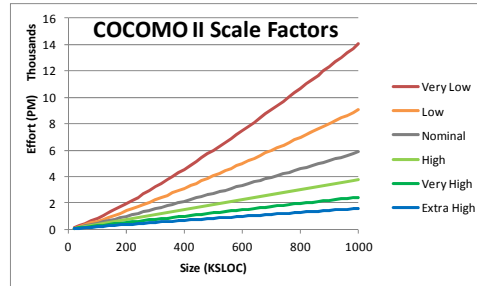
Function Points Issues

- Advantages
 - Countable early in the development effort
 - Language/technology independent
- Disadvantages
 - Subjectivity involved in counting
 - Don't capture non-functional requirements (*how* SW must *perform*) or technical and design constraints (*how* SW will be *built*)
- International Function Point Users Group (IFPUG), <http://www.ifpug.org>
 - Provides information and training on how to count and use function points
 - Certified Function Point Specialist (CFPS) certification

Software Engineering, A Practitioner's Approach, 3rd ed, Roger S. Pressman, McGraw Hill, Inc., 1992

Response of Cost to Size

- Cost increases as size increases
- The nature of the increase depends on development factors such as the management of communication and coordination
- Economies of scale occur if:
 - *Project big enough to warrant tools purchase*
 - *Can manage communication and coordination problems*



- Diseconomies of scale occur if:
 - *Tools are insufficient*
 - *Cannot manage communication and coordination problems*

Software Cost Estimation with COCOMO II,
Boehm et al., Prentice Hall PTR, 2000

5

Cost Drivers - Complexity

- Factors that relate to the software itself
 - Language
 - Function (intended use)
 - Hardware Limitations
 - Number of Modules
 - Amount of Integration
 - Percent of New Code
 - Quality of Development (for maintenance)



Warning: These are generally assumed to be cost drivers, but this is difficult to show statistically

Names and groupings may vary from model to model.

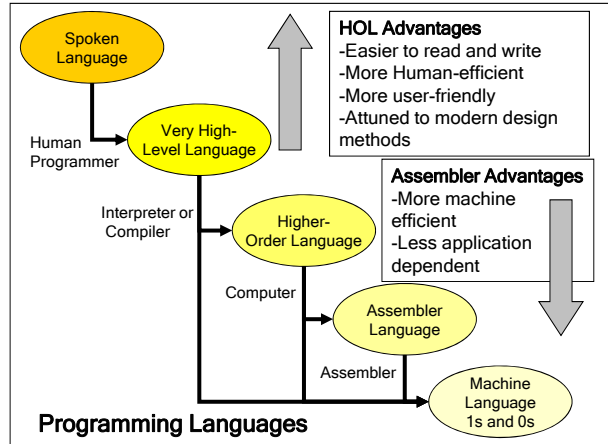
PRICE S Users Manual, Price Systems,
<http://www.pricystems.com>

Software Cost Estimation with COCOMO II,
Boehm et al., Prentice Hall PTR, 2000

Complexity - Language

- Vary in complexity from Machine to Very High Order (4GL)
- Models adjust for differences in language complexity, length, etc.
- Drives the amount of design vs. code vs. test
 - Object-Oriented languages require more design and less code and test

6



Quantitative Management of Software, Graduate School of Logistics and Acquisition Management, Air Force Institute of Technology, Air University, 1995

Unit IV - Module 12

31

Complexity - Function

- Function: purpose of software and required reliability
- Typical Applications include:
 - Statistical/Mathematical
 - String Manipulation
 - Graphical User Interface (GUI)
 - Data Storage and Retrieval
 - Graphical Functions
 - On-line Communications
 - Control Functions
 - Multi-media
 - Real Time
 - Interactive
 - Operating System
 - Logical Functions

7



Warning: This is the PRICE S model definition of the Application (APPL) cost driver - other models will have other unique terminology/definitions

PRICE S Users Manual, Price Systems,
<http://www.pricystems.com>

Unit IV - Module 12

32

Complexity - Other Factors

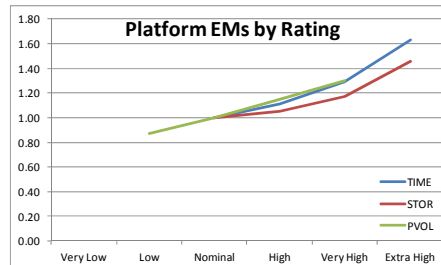
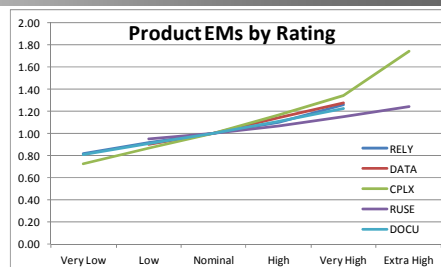
8

- Hardware Limitations: hardware on which software will run may drive the need for more efficient code, requirements uncertainty, schedule delays
- Number of Modules: drives integration, standardization, communication and coordination
- Quality of Developed Software (for Maintenance): better software requires less and easier-to-perform maintenance

Response of Cost to Complexity

- Cost increases as complexity increases
- Effort is greatest at the highest levels of complexity
- Relationship is generally thought to be exponential

Software Cost Estimation with COCOMO II,
Boehm et al., Prentice Hall PTR, 2000



Cost Drivers - Capability

v1.2



- Factors that relate to the developers and the development environment

- Application Experience



Warning: These are generally assumed to be cost drivers, but this is difficult to show statistically

- Skill

- Schedule Constraints



Warning: For these cost drivers, large projects tend to regress to the mean, relative to the underlying project database

- Tools Experience

- Development Location



Warning: These cost drivers are subjective in nature and so may introduce bias

Names and groupings may vary from model to model.

PRICE S Users Manual, Price Systems,
<http://www.pricystems.com>

Software Cost Estimation with COCOMO II,
Boehm et al., Prentice Hall PTR, 2000

Capability

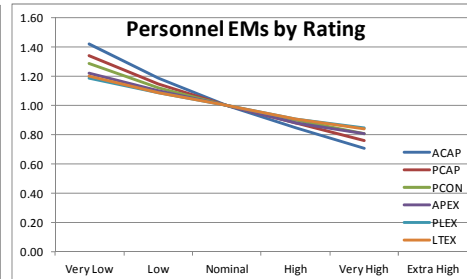
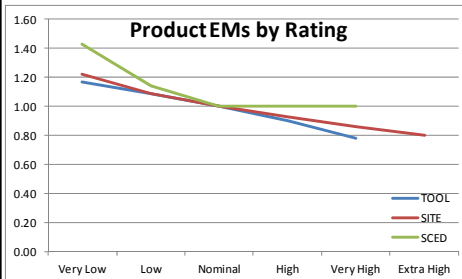
v1.2

- Overall Skill of Developer: Better skill requires less effort - Increased productivity offsets higher cost
- Experience with the Application: No learning required
- Experience with Development Tools: No learning required
- Schedule Constraints may cause developers to:
 - Increase the number of programmers leading to communication problems
 - Minimize requirements analysis and design which leads to more expensive fixes in code and test
 - Limit documentation leading to higher maintenance/reuse costs
- Development Location: Separation makes communication and coordination more difficult

Response of Cost to Capability

- Costs decrease as capability increases
- Impact is greater between lower and nominal capability

Software Cost Estimation with COCOMO II, Boehm et al., Prentice Hall PTR, 2000



Cost Drivers Example - Size

- For the mail order business Example Scenario, suppose the Developer proposes 3 solutions
- Find the cost for each given:
 - 10 - "Barebones" Solution: 50,000 SLOC
 - Original Solution: 100,000 SLOC
 - 13 - "Bells & Whistles" Solution: 150,000 SLOC
 - Nominal Complexity
 - Nominal Capability
 - Labor Rate \$16,000/month fully burdened
 - COCOMO II CER for software dev effort

COCOMO II CER

$$PM = A \cdot Size^E \cdot \prod_{i=1}^n EM_i$$

Where: PM = Person Months
 A = Constant = 2.94
 Size = KSLOC
 E = Sum of Scale Factors
 EM = Effort Multipliers

Software Cost Estimation with COCOMO II, Boehm et al., Prentice Hall PTR, 2000

Calculations:

50,000 SLOC	➔	$2.94 * 50^{1.0997} * 1 * \$16,000 = \$3,473,959$
100,000 SLOC	➔	$2.94 * 100^{1.0997} * 1 * \$16,000 = \$7,445,045$
150,000 SLOC	➔	$2.94 * 150^{1.0997} * 1 * \$16,000 = \$11,628,264$



Cost Drivers Example - Complexity

v1.2

- For the Example Scenario suppose the **100,000 SLOC** solution is chosen, but the complexity of the solution varies
- Find the cost for each given:
 - Low Complexity: EM = 0.6**
 - Nominal Complexity: EM = 1.0**
 - High Complexity: EM = 3.5**
 - Nominal Capability
 - Labor Rate **\$16,000/month fully burdened**
 - COCOMO II CER for software dev effort
- Calculations:

COCOMO II CER

$$PM = A \cdot Size^E \cdot \prod_{i=1}^n EM_i$$

Where: PM = Person Months
 A = Constant = 2.94
 Size = KSLOC
 E = Sum of Scale Factors
 EM = Effort Multipliers

Software Cost Estimation with COCOMO II, Boehm et al., Prentice Hall PTR, 2000

Low: EAF = 0.6	➔	$2.94 * 100^{1.0997} * 0.6 * \$16,000 = \$4,467,027$
Nom: EAF = 1.0	➔	$2.94 * 100^{1.0997} * 1.0 * \$16,000 = \$7,445,045$
High: EAF = 3.5	➔	$2.94 * 100^{1.0997} * 3.5 * \$16,000 = \$26,057,657$



Cost Drivers Example - Capability

v1.2

- For the Example Scenario suppose the **100,000 SLOC** solution is chosen, but the potential programmer capability varies
- Find the cost for each given:
 - Best Programmers: EM = 0.33**
 - Labor Rate **\$20,000/month fully burdened**
 - Average Programmers : EM = 1.0**
 - Labor Rate **\$16,000/month fully burdened**
 - Junior Programmers: EM = 5.22**
 - Labor Rate **\$14,000/month fully burdened**
 - Nominal Capability
 - COCOMO II CER for software dev effort
- Calculations:

COCOMO II CER

$$PM = A \cdot Size^E \cdot \prod_{i=1}^n EM_i$$

Where: PM = Person Months
 A = Constant = 2.94
 Size = KSLOC
 E = Sum of Scale Factors
 EM = Effort Multipliers

Software Cost Estimation with COCOMO II, Boehm et al., Prentice Hall PTR, 2000

Best: EM = 0.33	➔	$2.94 * 100^{1.0997} * 0.33 * \$20,000 = \$3,071,081$
Avg: EM = 1.0	➔	$2.94 * 100^{1.0997} * 1.0 * \$16,000 = \$7,445,045$
Jr: EM = 5.22	➔	$2.94 * 100^{1.0997} * 5.22 * \$14,000 = \$34,005,242$

Development Schedule

- Often have to estimate schedule as well as cost
- Issues
 - Schedule driven by contract or need date not by reality
 - Developers don't have a good understanding of scheduling
- Schedule vs. Effort
 - **Schedule months** = Number of calendar months to develop
 - **Effort months** = Number of calendar months * the number of people working per month (Person Months)

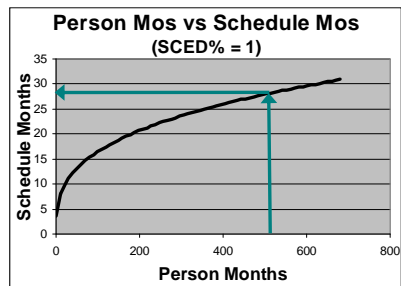
20



Schedule Example

- Let's suppose the Owner wants to know how long the schedule (TDEV) would be with no compression (SCED % = 1.0) for the 100,000 SLOC solution
- Given:
 - 465 Person months
 - 1 Person month = 152 hrs
 - SCED% = 1.0 (100%)
 - E = 1.1433
 - COCOMO II CER for schedule

11



COCOMO II Schedule CER

$$TDEV = [C * (PM_{NS})^F] * SCED\%$$

$$TDEV = [C * (PM_{NS})^{(D+0.2*[E-B])}] * SCED\%$$

Where: TDEV = Calendar time in months
 E = Sum of Scale factors
 C = Constant = 3.67 B = Constant = 0.91
 D = Constant = 0.28 F = D + 0.2 X (E-B)
 PM_{NS} = Person Months (un-scaled)
 SCED% = The amount of schedule compression or stretch-out as a percent of the nominal value

• Calculations:

$$TDEV = [3.67 * (465)^{(0.28+0.2*(1.143-0.91))}] * 1 = 27.28 \text{ months}$$

Software Cost Estimation with COCOMO II, Boehm et al., Prentice Hall PTR, 2000



Schedule Compression Example

v1.2

- Let's suppose the Owner wants to know the Compression (1-SCED%) the Developer is counting on to meet the 18 month deadline for the 100,000 SLOC solution

- Given:
 - 18 month Schedule
 - 465 Person months
 - E = 1.1433
 - COCOMO II CER for schedule

- Calculations:

$$\text{TDEV} = [3.67 * (465)^{(.28 + .2 * (1.143 - .91))}] * \text{SCED\%} = 18 \text{ months}$$

$$\text{SCED\%} = 18 / [3.67 * (465)^{(.28 + .2 * (1.143 - .91))}] = 66\%$$

$$(1 - \text{SCED\%}) = (1 - 66\%) = 33\%$$

Software Cost Estimation with COCOMO II, Boehm et al., Prentice Hall PTR, 2000

COCOMO II Schedule CER


$$\text{TDEV} = [C * (\text{PM}_{\text{NS}})^F] * \text{SCED\%}$$

$$\text{TDEV} = [C * (\text{PM}_{\text{NS}})^{(D + 0.2 * [E - B])}] * \text{SCED\%}$$

Where: TDEV = Calendar time in months
 F = D + 0.2 X (E-B)
 C = 3.67
 PM_{NS} = Person Months (un-scaled)
 D = 0.28
 E = Sum of Scale factors
 B = 0.91
 SCED% = The amount of schedule compression or stretch-out as a percent of the nominal value





Post-Deployment Software Support

v1.2

- Like hardware, software has an operational phase
 - Costs must be accounted for in life cycle cost (LCC)
- Operations and support (O&S) for software termed  Post-Deployment Software Support (PDSS)
 - Includes software maintenance
 - Also includes help desk/trouble ticket functions
- Models only account for software maintenance
 - Other areas need to be addressed outside of model

Remember, how well software was originally developed has a **major impact** on software support costs. *You pay in development or you pay in support.*

Software Maintenance

- Software doesn't degrade or wear out like hardware
 - Use may uncover bugs not addressed in testing
 - When introduced to a new environment, software may "break"
-  Software Maintenance includes:
 -  Corrective: Fixes defects in the code
 -  Adaptive: Modifies the software to accommodate changes in the external environment
 -  Perfective: Extends the software beyond its original functional requirements
- 17 • For Software, there is overlap between Maintenance and Development
 - Portions of code may need maintenance during development
 - When additional capability is added, Software maintenance can be thought of as a mini-development effort
- Cost drivers are the same + the quality of the code being maintained

Software Engineering, A Practitioner's Approach, 3rd ed, Roger S. Pressman, McGraw Hill, Inc., 1992

Estimating Development Methodologies

- Waterfall
- Agile
- Other Methods



Waterfall

v1.2

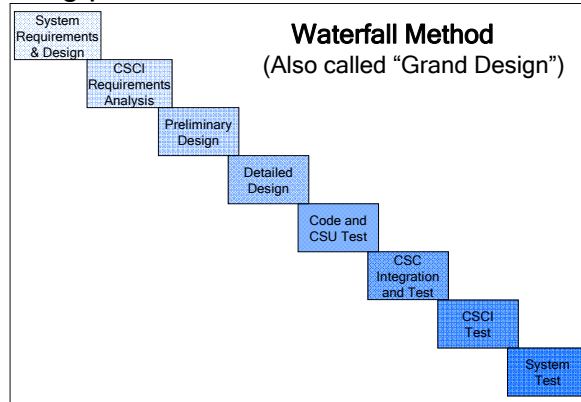
- Traditional Development method follows a basic System Engineering process

Benefits:

- Good when there are stable requirements - provides structure to development

Pitfalls:

- Doesn't allow prototyping
- No product to look at until completely done
- Not attuned to evolving needs



Quantitative Management of Software, Graduate School of Logistics and Acquisition Management, Air Force Institute of Technology, Air University, 1995

PRICE S Users Manual, Price Systems, <http://www.pricystems.com>



Unit IV - Module 12

47

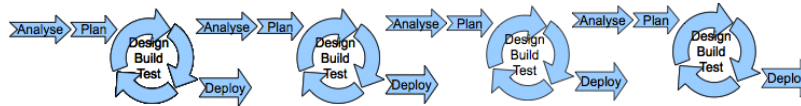
© 2002-2013 ICEAA. All rights reserved.



Agile

v1.2

- Iterative development which prioritizes evolution of requirements and solutions through collaboration of cross-functional teams
 - Each iteration is a full software development cycle
 - At the end of an iteration, the product can be reviewed and evaluated by the customer for feedback
- Agile development stresses team work and face-to-face communication



Benefits:

- Adaptable to change
- Prioritizes customer satisfaction and communication
- Focus on business need and business value
- Sustainable development pace



AKA Scrum

"Agility XL", Schaaf, R.J., Systems and Software Technology Conference 2007, Tampa, FL, 2007

Pitfalls:

- Not structured enough for architecture design or re-design work
- May need to be combined with waterfall methodology to fit organizational needs

Are Parametric Techniques Relevant for Agile Development Projects?, Minkiewicz, Arlene. PRICE Systems, 2012.



Unit IV - Module 12

48

© 2002-2013 ICEAA. All rights reserved.

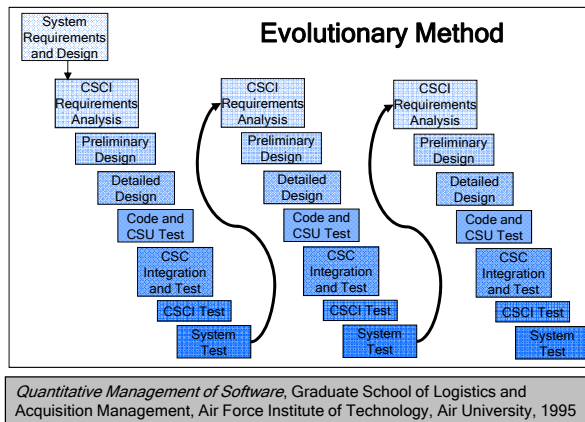




Evolutionary

v1.2

- Begins with a prototype containing core capability. Customer provides feedback; prototype is adjusted and additional capability added. Process is repeated until the system is complete



Quantitative Management of Software, Graduate School of Logistics and Acquisition Management, Air Force Institute of Technology, Air University, 1995

Evolutionary:

- Need general objectives, not requirements to start
- Prototypes are paper, software model, working product, existing product

Benefits:

- Gets a product to customer quickly and encourages customer involvement

Pitfalls:

- More time consuming than other methods for final product
- Must have a plan for execution even without complete requirements



Modeling Evolutionary

v1.2

- Model as multiple Waterfalls
 - Model each pass as a separate Waterfall including the previous pass as reused and/or adapted and deleted code
 - Include all phases (system requirements through system test) in each pass but make adjustments for reused and adapted code
 - Passes are sequential therefore may need to adjust productivity for later passes
- Have to determine what will be done in each pass even though requirements are not complete

We'll look again at our example for Evolutionary

v1.2

Incremental

- Software is built in increments, complete requirements for the entire system are defined up front and allocated to increments

Incremental Method

Quantitative Management of Software, Graduate School of Logistics and Acquisition Management, Air Force Institute of Technology, Air University, 1995

Increments:

- Normally sequential; can be concurrent
- Includes design, code and test for requirements in that increment

Benefits:

- Increased communication
- More frequent and faster deliveries

Pitfalls:

- Requirements must be defined
- Need a sound architecture
- Only deliver a small part of a system at a time

Unit IV - Module 12

© 2002-2013 ICEAA. All rights reserved.

v1.2

Modeling Incremental

- Model as multiple Waterfalls
 - Model each increment as a separate Waterfall - use effort estimated from CSCI design through test
- For system costs, model entire system as a single Waterfall and use only system level costs such as requirements analysis and system test
- Increment may be at lower level with CSCI treated as system level
- If increments are sequential:
 - May need to adjust productivity for later increments
 - May need to estimate system test after each increment is delivered- including only those parts of the code being tested

We'll look again at our example for Incremental

Unit IV - Module 12

© 2002-2013 ICEAA. All rights reserved.

v1.2

Evolutionary Example

- Recommendation of third consultant

Run 1: First Evolution or Pass

- Customer Order Management System
 - Core Capabilities
 - Prototype Interface
- COTS Package 1
- COTS Package 2

Run 2: Second Evolution or Pass

- ☐ Reuse code
 - Customer Order Mgt System
 - COTS Packages
- ☐ Adapted code
 - Prototype Interface from Pass 1
- ☐ New code
 - Built in double checks


Run 3: Third Evolution or Pass

- ☐ Re-test code
 - Customer Order Mgt System
 - COTS Packages
- ☐ Adapted code
 - Prototype Interface from Pass 2
- ☐ New code
 - Auto-generated notification

Adjust factors to reflect that is only a prototype

Adapted code - use ESLOC or make adjustments for reduced design, code and test

Reused code treated like COTS - included for integration and re-test



Unit IV - Module 12

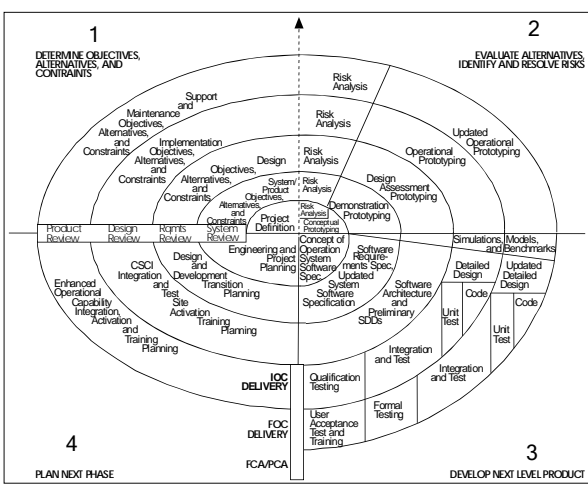
53

© 2002-2013 ICEAA. All rights reserved.

v1.2

Spiral

- Breaks effort into pre-defined spirals to allow for Risk Assessment



Spirals:

- Breaks effort into 4 quadrants
- Uses other methods and adds risk review
 - Waterfall
 - Incremental
 - Evolutionary


Benefits:

- Emphasizes alternative analysis
- Risk driven approach

Pitfalls:

- Hard to use contractually
- Takes longer to develop

Software Engineering, A Practitioner's Approach, 3rd ed, Roger S. Pressman, McGraw Hill, Inc., 1992



Unit IV - Module 12

54

© 2002-2013 ICEAA. All rights reserved.



Modeling Spiral

v1.2

- Model as multiple passes - similar to Evolutionary
 - Model each spiral as a separate pass - but include previous spiral as reused and/ or adapted code
 - Include only those phases actually addressed in that spiral and make adjustments for reused and adapted code
 - For example, in the spiral diagram, the second spiral only has software requirements specification and system software specification - Later spirals have just code and test
 - Spirals are sequential therefore may need to adjust productivity for later passes
 - If model or CER doesn't accommodate spiral, may need to add effort for risk assessment, planning, and analysis of objectives



Development Methodologies

v1.2

- A software development methodology is an overall approach to system development
 - Need to understand methodology being used for proper modeling, calibration, and CER development
- Commonly used methodologies are
 - **15** Waterfall: conventional, "theoretical" methodology
 - Agile: based on iterative and incremental development
 - **16** Other common methods
 - Incremental: breaks development into clearly-defined, stand alone system increments
 - Evolutionary: built to satisfy requirements as they evolve
 - Spiral: risk based analysis of alternatives approach
 - More detailed information provided in the advanced topics



Estimating Techniques Applied to Software

- Analogy
- Parametric



Analogy

- Analogy: Performing a comparative analysis of similar systems with adjustments to account for differences between new and analogous systems
- Example:
 - Let's suppose for our mail order example that the owner chooses the First consultants Waterfall methodology solution
 - The First consultant from Waterfall methodology must estimate his cost to set up the COTS software
 - Using "ACME" Office and a new COTS package for human resources, accounting, and inventory management functions
 - Consultant just completed a similar effort using 4 COTS products for a company twice the size of the Mail Order company
 - Previous effort was:
 - Set up software - **280 hours**
 - Load Data - **80 hours**
 - Implement at customer's site - **100 hours**
 - Train users - **20 hours**

12





Analogy Example

v1.2

- Differences in new effort:
 - 2 COTS packages reduces effort by 20%
 - Data load same- company is smaller but data is not as automated
 - Engineers say the implementation is expected to require 15% less effort because the company is smaller
- Calculations:



Warning: The basis for this analogy is not strong. This is a YELLOW BOE at best. The comparison between the two programs has been simplified for purposes of the example

	<u>Hours</u>
Set-up COTS product: $280 - (280 * 0.2) = 224$	224
Data Load: 80	80
Implementation: $100 - (100 * 0.15) = 85$	85
Training: $20 - (20 * 0.15) = 17$	<u>17</u>
Total	406
Given a labor rate of \$100K/year:	
$(406 \text{ Hrs} / 1920 \text{ Hrs/Year}) * \$100,000 = \text{\$21,145}$	



Parametric

v1.2

- Most common estimating technique for software development
 - Commercial models are parametric based
- Parametric based method is a mathematical relationship between a physical (size) or performance (reliability) parameter and the cost of a system
- Example:
 - Mail order company continued - First Consultant's waterfall solution
 - Must estimate cost of custom code
 - $5,000 + 7,500 + 3,000 + 1,000 + 2,500 = 19,000 \text{ SLOC}$
 - Vendor uses COCOMO II to estimate jobs - has calibrated to his company
 - $E = 1.0405$ (calibrated on similar efforts)
 - $EM = 0.38$ (skilled development team)
 - No adjustments were necessary for the code itself
 - Labor rate is **\\$16,000/month**



Parametric Example - SLOC

v1.2

- Mail order company continued - First Consultant's waterfall solution
- Given:
 - Must estimate cost of custom code (**19,000 SLOC**)
 - Labor rate is **\$16,000/month**

COCOMO II CER

$$PM = A \cdot Size^E \cdot \prod_{i=1}^n EM_i$$

Where: PM = Person Months
 A = Constant = 2.94
 Size = KSLOC
 E = Sum of Scale Factors
 EM = Effort Multipliers

Software Cost Estimation with COCOMO II, Boehm et al., Prentice Hall PTR, 2000



- Calculations:

$$\text{Person Months} = 2.94 * 19^{1.0405} * 0.38 = 23.92$$

$$\text{Cost} = 23.92 * \$16,000 = \$382,643$$



Parametric Example - Schedule

v1.2

- Mail order company continued - First Consultant's waterfall solution
- Given:
 - Must estimate schedule for development of custom code (19,000 SLOC)
 - Person Months = **23.92**
 - E = 1, so F = 0.298
 - Nominal schedule, SCED% = 1.0

COCOMO II Schedule CER

$$TDEV = [C * (PM_{NS})^{(D+0.2*[E-B])}] * SCED\%$$

Where:
 TDEV = Calendar time in months
 E = Sum of Scale factors
 C = 3.67 B = 0.91
 D = 0.28 F = D + 0.2 X (E-B)
 PM_{NS} = Person Months (un-scaled)
 SCED% = The amount of schedule compression or stretch-out as a percent of the nominal value




- Calculation:

$$\text{Schedule Months} = 3.67 * 23.92^{0.298} * 1.0 = 9.45$$

Software Cost Estimation with COCOMO II, Boehm et al., Prentice Hall PTR, 2000



Software CER Development

- Software CER development is the same as hardware or other CER development processes
 - Allows statistical inferences to be made
 - Underlying assumption is that future reflects the past
 - Expanded discussion in Modules 2, 3 and 8
- Important reminders when developing your CERs
 - Variable selection process very important 
 - Stay within the relevant range 
 - Normalize the data 
 - Test relationships
 - Perform regression


Challenges in Estimating Software

- System Definition
- Sizing and Tech
- Quality
- COTS
- Calibration
- Databases
- Growth and Demand

Challenges - System Definition

- Obtaining System Definition
 - Must work with experts
 - Define notional system based on known requirements and include risk assessment for unknowns
 - Definition often at a high level
 - May include use of COTS software
 - Talk to commercial vendors for inputs
 - Multiple packages may be used
 - For custom code, look at similar systems for functions that are required
 - Assess need for both internal and external interfaces
 - Refine definition over time as system takes shape

Challenges - Sizing and Tech

- Sizing Is An Estimate Too
 - Use standard estimating methods 
- Rapid Technology Change
 - Changes during the development process may have to be addressed
 - COTS Upgrades
 - May have to reintegrate
 - Simple retest to complete redo or no change at all - depends on COTS
 - Development Tool Changes
 - Newer tools may simplify effort (but still require learning)
 - May force change to the development process

Challenges - Quality

- Difficulty in Assessing Quality
 - “Don’t know how good it is until you’re done”
 - Good planning impacted by tight schedules and other constraints
 - Software quality measures may help
 - Defects Identified
 - Defects Fixed
 - Failed Fixes
 - Severity of Defects
 - Location of Defect
 - Degree of Cohesion and Coupling
 - Requirements satisfied
 - Depth of testing
 - Adequacy of Documentation
 - MTTD Errors
 - McCabe’s cyclomatic complexity

Space Systems Cost Analysis Group Software Methodology Handbook, Version 1.0, June 1995, <https://sscag.saic.com/>

Challenges - COTS



• Using Commercial Off-the-Shelf (COTS) Software

14

- No code visibility
- Difficult to customize - no source code
- Effort dependent on the software architecture
- Might be too rigid to handle changing requirements
- Assumes many users will find errors - need additional testing
- Upgrades to COTS may force reintegration with custom code
- Support costs for custom code may be affected and will vendor need support for COTS
- Still must perform requirements definition, design, and test of the overall system
- Dealing with licensing, royalties, incompatibilities between packages, lack of source code and understanding package
- Estimation of COTS integration not mature

Warning:
COTS ≠ Cheap!




Challenges - Calibration

- Calibration of models
 - Most models built on industry averages therefore calibration may increase accuracy
 - Adjusts relationships/values to achieve representative known outcomes
 - Understand how your model calibrates
 - Must collect cost, technical and programmatic data
 - Check content of actual data vs. content of model
 - Generally models have a calibration mode but may need to tweak the model

3

Calibration of models must be done with care but is generally an improvement over default values

Challenges - Databases

-  Database Development
 - Most models don't address major database development
 - Must estimate outside of model using other estimating techniques
 - Consider
 - Number of feeder systems
 - Number of data elements
 - Number of uses
 - Number of users
 - COTS database software for development and feeder systems

2

Challenges - Growth and Demand

v1.2

- Requirements and Code Growth
 - Delivered project is bigger than estimated
 - Increase driven by:
 - Poor understanding of requirements initially
 - New requirements added during development
 - Underestimate of required SLOC
 - Code reuse optimism
 - Key is to know the track record and account for expected growth
- Supply and Demand of Labor
 - Affects personnel availability and cost of qualified personnel



Warning: Beware requirements creep!

Software Cost Estimating Summary

v1.2

- Understanding software cost estimation is critical because software is part of almost every estimate
- Software cost estimating is in many ways similar to hardware estimating
- There are a variety of software development approaches that can affect development cost and must be modeled accordingly to estimate
- Analogy and Parametric are commonly used to estimate software development costs
- There are a number of commercial parametric models available to estimate software costs
- Software provides a number of specific challenges for the estimator

Resources

- [Pressman] *Software Engineering, A Practitioner's Approach, Third Edition*, Roger S. Pressman, McGraw Hill, Inc., 1992
- [Boehm 81] *Software Engineering Economics*, Barry W. Boehm, Prentice Hall, 1981
- [Boehm 2000] *Software Cost Estimation with COCOMO II*, Boehm et al., Prentice Hall PTR, 2000
- [ISPA 1999] Spring 2nd Edition *Joint Industry/Government PARAMETRIC ESTIMATING HANDBOOK*, <http://www.ispa-cost.org/PEIWeb/toc.htm>
- [GSAM 2000] *Guidelines for Successful Acquisition and Management of Software Intensive Systems: Weapon Systems, Command and Control Systems, Management Information Systems, Version 3.0*, Dept of the Air Force, Software Technology Support Center, 2000
- [AFIT] *Quantitative Management of Software*, Graduate School of Logistics and Acquisition Management, Air Force Institute of Technology, Air University, 1995
- [IFPUG] *International Function Point Users Group*, <http://www.ifpug.org>
- [Taylor] *Object-Oriented Technology: A Manager's Guide*, David A. Taylor, Addison-Wesley, 1990
- [Cox] *Object-Oriented Programming: An Evolutionary Approach*, Brad J. Cox, Addison-Wesley, 1987
- *SEER-SEM*, Galorath Inc., <http://www.galorath.com>
- [STSC] *Crosstalk - The Journal of Defense Software Engineering*, <http://www.stsc.hill.af.mil/CrossTalk/2003/07/index.html>

Resources

- [Reifer] "Quantifying the Debate: Ada vs. C++," Donald J. Reifer, *Crosstalk: The Journal of Defense Software Engineering*, Vol. 9, Number 7, July 1996
- [Jensen] "Software Estimating Model Calibration," Randall W. Jensen, *Crosstalk: The Journal of Defense Software Engineering*, Vol. 14, Number 7, July 2001
- [Jones 1] *Applied Software Measurement: Assuring Productivity and Quality*, 2nd ed, Capers Jones, McGraw Hill, 1996
- [Jones 2] *Estimating Software Costs*, T. Capers Jones, McGraw Hill, 1998
- *COCOMO II*, <http://sunset.usc.edu>
- [PRICE S] *PRICE S Users Manual*, Price Systems, <http://www.pricystems.com>
- [MIL STD 498] *Military Standard 498, "Software Development and Documentation"*, December 1994
- [IEEE] *IEEE/EIA 12207.2-1997, IEEE/EIA Guide, Industry Implementation of International Standard ISO/IEC 12207: 1995, Standard for Information Technology, Software Life Cycle Processes - Implementation Consideration*, April 1998
- [MIL-HDBK-881A] Department of Defense Handbook Work Breakdown Structures for Defense Materiel Items, July, 2005
- [SEI-CMM] *Capability Maturity Model for Software, Version 1.1*, Paulk, Mark C. et al., Software Engineering Institute, Carnegie Mellon University, February 1993
- [Schaaf] "Agility XL", Schaaf, R.J., Systems and Software Technology Conference 2007, Tampa, FL, 2007
- *Are Parametric Techniques Relevant for Agile Development Projects?*, Minkiewicz, Arlene. PRICE Systems, 2012.