Software Cost Estimation

Basic Ideas

- “You can’t manage what you can’t measure”.

- Some model, however flawed, is better than no model at all.

Function-Point Analysis (FPA)

(Some slides from James Gain, University of Cape Town)

What is FPA?

- A Function Point is an isolatable feature of the product.
- A way to estimate the amount of effort required for a project based on surface characteristics.
- To some degree, the kinds of function points can be standardized and the amount of effort attached to them can be as well.

Function Points vs. Use Cases

- Similar, but not identical, idea.
- FP's don't need to satisfy the same criteria of being a coherent transaction that Use Cases do.
- FP's are finer grain.

Advantages of FPA

- Independent of programming language. Some programming languages are more compact, e.g. C++ vs. Assembler.
- Use readily countable characteristics of the "information domain" of the problem.
- Does not "penalize" inventive implementations that require fewer LOC than others.
- Makes it easier to accommodate reuse and object-oriented approaches.

Computing Function Points

Analyze information domain of the application and develop counts

Establish *count* for input domain and system interfaces

Weight each count by assessing complexity

Assign level of complexity (simple, average, complex) or *weight* to each count

Assess the influence of global factors that affect the application

Grade significance of external factors, complexity adjustment factors $\mathbf{F_i}$, such as reuse, concurrency, OS, ...

Tally function points

FP = SUM(count x weight) x C
where
complexity multiplier C = (0.65+0.01 x N)
degree of influence N = SUM(F_i)

Function Point Computing Form

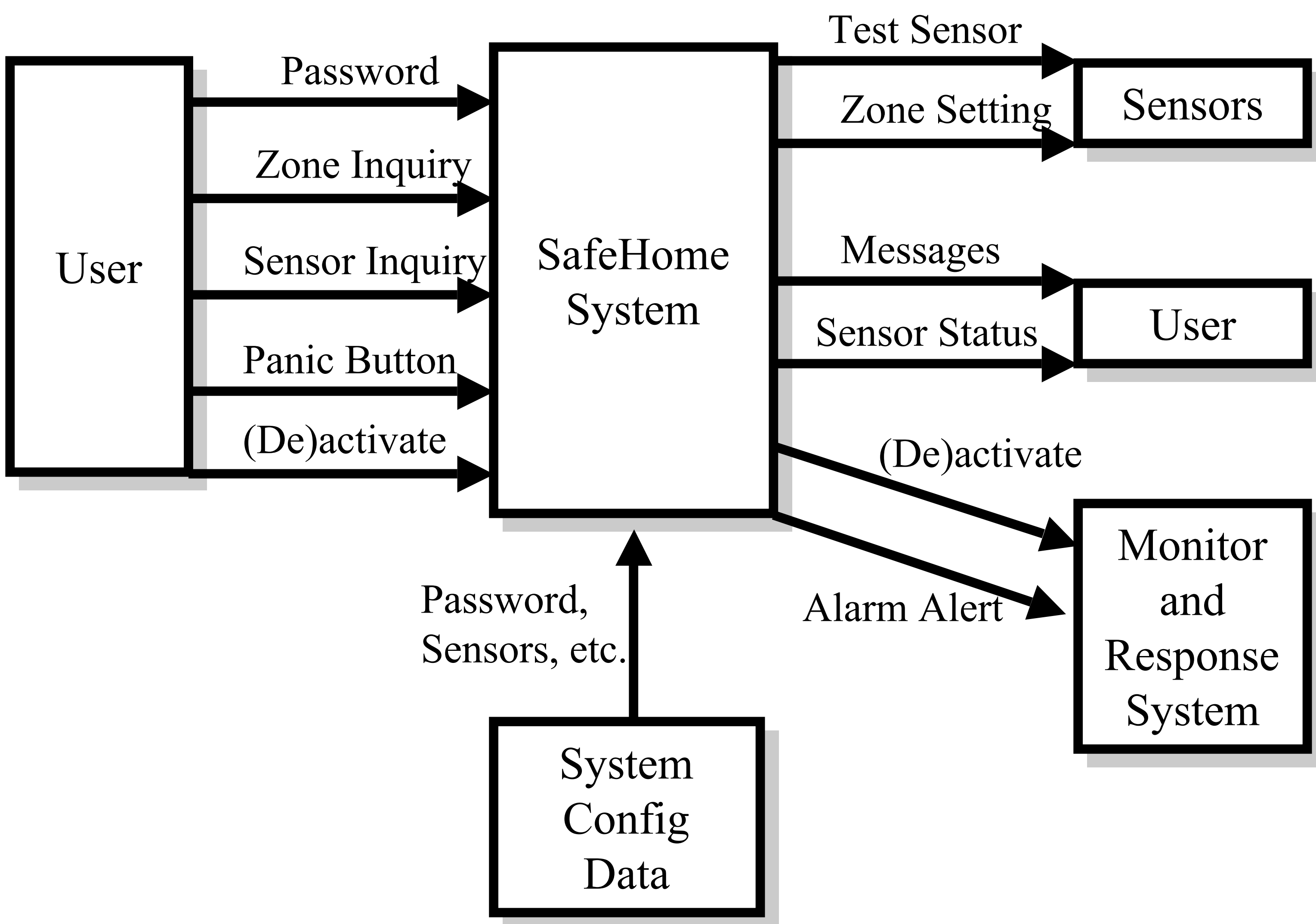
measurement parameter	count		weighting factor (circle) simple	avg.	complex		
number of user inputs		X	3	4	6	=	
number of user outputs		X	4	5	7	=	
number of user inquiries		X	3	4	6	=	
number of files		X	7	10	15	=	
number of ext.interfaces		X	5	7	10	=	
count-total						→	
complexity multiplier							
function points						→	

Taking Complexity into Account

Complexity Adjustment Values (F_i) are each rated on a scale of 0 (not important) to 5 (very important):

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. System to be run in an existing, heavily utilized environment?
6. Does the system require on-line data entry?
7. On-line entry requires input over multiple screens or operations?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and instillation included in the design?
13. Multiple installations in different organizations?
14. Is the application designed to facilitate change and ease-of-use?

Example: SafeHome Functionality



Example: SafeHome FP Calc

measurement parameter	count		weighting factor (circle) simple	avg.	complex		
number of user inputs	3	X	3	4	6	=	9
number of user outputs	2	X	4	5	7	=	8
number of user inquiries	2	X	3	4	6	=	6
number of files	1	X	7	10	15	=	7
number of ext.interfaces	4	X	5	7	10	=	22
count-total							52
complexity multiplier	$[0.65 + 0.01 \times \sum F_i] = [0.65 + 0.46]$						1.11
function points							58

Self-Chartered Standards Body

International Function Points User Group (IFPUG)

5009-28 Pine Creek Drive Westerville, OH 43081-4899
Voice: (614) 895-7130; FAX: (614) 895-3466
E-mail: 102214.2013@compuserve.com
URL: http://www.ifpug.org/ifpug

Productivity Ranges

From **Software Engineering Baselines (Rome Laboratory)**
http://www.dacs.dtic.mil/techs/baselines/productivity.html

Definition: FPs per person month or SLOC per person month.

FPs are a weighted sum of the number of program inputs, outputs, user inquiries, files, and external interfaces.

Ranges:

- **Productivity:** From 2 to 23 FPs per Person Month with a median of 5.6 FPs per Person Month, or from 80-400 SLOC per Person Month.

- **Size:** From 100 to 2,300 FPs with a median of 993 FPs, or from 2 to 512 KSLOC.

- **Function Point Conversion:** From 6 to 320 SLOC per FP.

Notes: FPs are most applicable to Management Information Systems (MIS) and other business applications.

Language Factor

From **Software Engineering Baselines (Rome Laboratory)**

http://www.dacs.dtic.mil/techs/baselines/productivity.html

Table 2.1-2: Sloc Per FP By Language

Language	SLOC per FP
Assembler	320
Macro Assembler	213
C	150
Algol	106
Chill	106
Cobol	106
Fortran	106
Jovial	106
Pascal	91
RPG	80
PL/I	80
Modula-2	71
Ada	71
Prolog	64
Lisp	64
Forth	64
Basic	64
Logo	53
4th Generation Database	40
Strategem	35
APL	32
Objective - C	26
Smalltalk	21
Query Languages	16
Spreadsheet Languages	6

Cost Analysis: COCOMO
COnstructive COst MOdel

(some slides contributed by Simon Harada, HMC)

What is Cocomo?

- Estimates the number of person months it will take to develop a product
- Designed by Barry Boehm (the Spiral-model guy)
- Function points can be used
  (by translating FP to
  SLOC = “Source lines of code”
  or DSI = “Delivered source instructions”)

The Main Idea

- Input: Program size (SLOC)
- Parameters: Product, personnel, hardware, project attributes etc.
- Output: Person Months

3 Levels of Cocomo

- **Basic**: Static, single-valued model computes software development effort as a function of only program size.
- **Intermediate**: In addition to program size, uses “cost drivers” that include subjective assessment of project parameters.
- **Advanced**: Includes all intermediate factors, as well as an assessment of the cost drivers’ impact on each step of the engineering process.

Basic Cocomo Model

- Cocomo Modes
  - **Organic**: Small groups working on well understood problems
  - **Semi-detached**: Intermediate problem difficulty
  - **Embedded**: Complex, poorly understood problems

Basic Model (cont'd)

- Effort (PM) = $A(KDSI)^b$
  - PM = person months
  - A, b = constants determined by complexity factor
  - KDSI = thousands of delivered source instructions

Basic Model (cont'd)

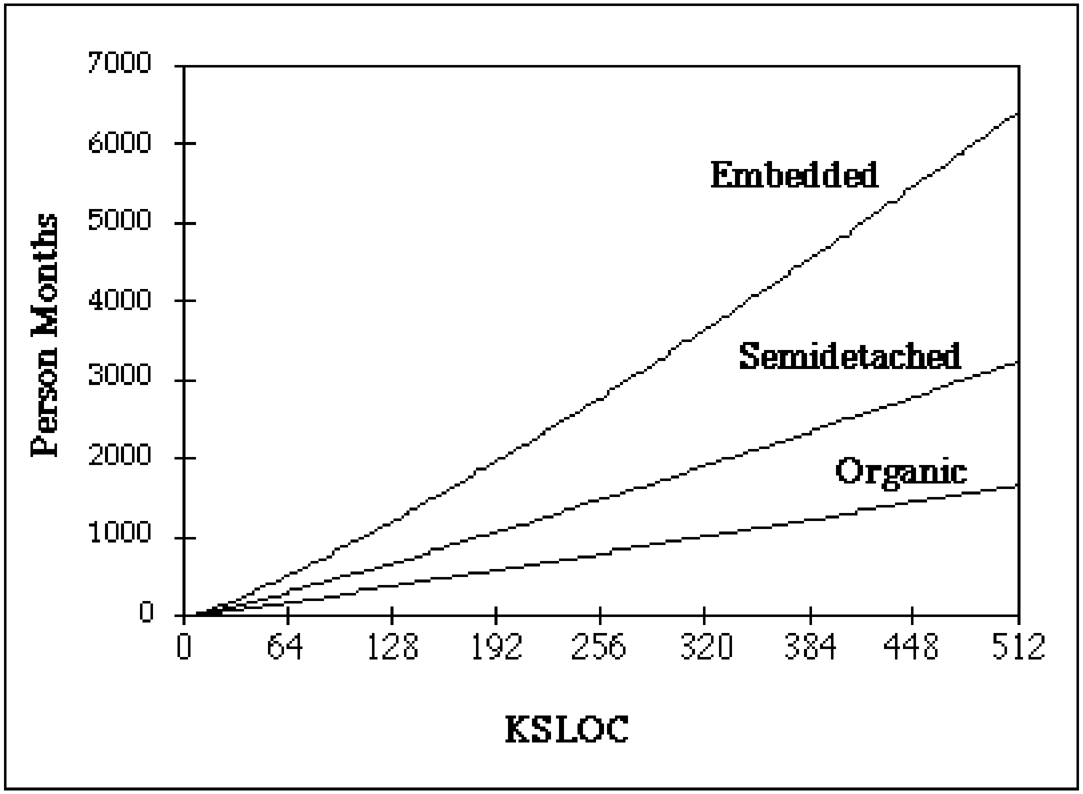


Figure 2.1-4: Effort as a Function of KSLOC

- Fitted Parameters:
  - Organic:
    $PM = 2.4(KDSI)^{1.05}$
  - Semi-detached:
    $PM = 3.0(KDSI)^{1.12}$
  - Embedded:
    $PM = 3.6(KDSI)^{1.20}$

Intermediate Cocomo Model

- Product Attributes
  - Reliability, database size, complexity
- Computer Attributes
  - Execution time constraints, storage constraint, machine volatility, turnaround time
- Personnel Attributes
  - Capability, application experience, machine experience, program language experience
- Project Attributes
  - SW tools, schedule, programming practices

Intermediate Cocomo Estimates

- **Intermediate Model** vs. **Basic Model**

	Intermediate Model	Basic Model
Organic:	PM = $3.2(KDSI)^{1.05}$	$2.4(KDSI)^{1.05}$
Semi-Detached:	PM = $3.0(KDSI)^{1.12}$	$3.0(KDSI)^{1.12}$
Embedded:	PM = $2.8(KSSI)^{1.20}$	$3.6(KDSI)^{1.20}$

- In Intermediate Cocomo, effort estimates are multiplied by 15 cost factor drivers, such as

Effect of failure	slight inconvenience	easily recoverable	recoverable	high financial loss	risk to human life
Factor	0.75	0.88	1.00	1.15	1.40

Limitations of Cocomo

- Factors not considered by Cocomo
  - Requirements volatility
  - Personnel continuity
  - Management quality
  - Customer interface quality

Summary: Cocomo Strengths and Weaknesses

Pros

- Objective, repeatable analyzable formula
- Efficient, good for sensitivity analysis
- Objectively calibrated to experience

Cons

- Subjective inputs
- No assessment of exceptional circumstances
- Calibrated to past, not necessarily future

COCOMO II

Some slides courtesy of Jonathan Hsu, HMC

What is COCOMO II?

- **COCOMO II**, developed in 1997, is a refinement and revision of the previous model, now known as **COCOMO 81**.

Formula for Project Effort

$$PM_{nominal} = A * Size^{B}$$

- $\mathbf{PM_{nominal}}$ is the number of person months required to finish.
- **Size** is measured in **thousands of source lines of code** (**KSLOC**).
- **A** is a constant representing effects of **increasing project size**.
- **B** is the **scale factor**, accounting for **economy or diseconomy of scale in the project**.

How Size is Derived

- Determine number of **function points** by **type**.
- Classify each **function point type** from **low to high**.
- Translate classifications to numerals via chart to obtain **Unadjusted Function Points** (**UFP**).
- Multiply the **UFP** by a **constant** determined by language:

Language	Lines of Code
Assembly	320
C	128
C++	29
Pascal	91

Formula for B

$$B = 0.91 + 0.01 \times \Sigma W_i$$

- Previously, **COCOMO 81** had only **3 scaling factors**: **Organic**, **Semidetached**, & **Embedded**.
- These had scaling factors of **1.05**, **1.12**, and **1.20** respectively.
- **COCOMO II** uses **scale drivers:** (**PREC**, **FLEX**, **RESL**, **TEAM**, **PMAT**).
- Each **scale driver** has a **weight** attached to it. These are summed and added to the constant **B**.

Effort Multipliers

- The **Early Design** model has **7** Effort Multipliers.
- The **Post Architectural** model has **17** Effort Multipliers.
- Each multiplier is **evaluated** and **assigned a weight between 1 and 5**.
- These weights are **multiplied together** and applied to the $\mathbf{PM_{nominal}}$ to find the $\mathbf{PM_{modified}}$.

Adjusting for Re-Usage

$$PM_{nominal} = A*Size^{B} + ASLOC^{AT/100}/ATPROD$$

- **ASLOC** is the **actual source lines of code**.
- **AT** is the **percentage of code** that can be recycled using **automatic translation** methods.
- **ATPROD** is a **constant empirically determined** by studies.

COCOMO II Products

- **Costar v5.0** (Win95, Win3.1), produced by **Softstar**.
- **A.S.K. P.E.T.E** (Win95/98), produced by the **NASA Glen Research Center**.
- **USC COCOMO II.1999.0** (Win95/98/NT, SunOS 4.x/5.x), produced by the **University of Southern California**.

Reference

USC Center for Software Engineering

http://sunset.usc.edu/cse/pub/tools/

for even more tools see

http://sunset.usc.edu/research/cocomosuite/suite_main.html

Cocomo II Applet

http://www.cs.utexas.edu/users/lwerth/cs373/gil/example.html