Estimating Size and Effort

Dr. James A. Bednar

jbednar@inf.ed.ac.uk

http://homepages.inf.ed.ac.uk/jbednar

Dr. David Robertson

dr@inf.ed.ac.uk

http://www.inf.ed.ac.uk/ssp/members/dave.htm

Estimating SW Size and Effort

Most methods for estimating the total effort required for a software project (to decide on schedule, staffing, and feasibility) depend on the size of the software project.

Unfortunately, it is difficult to measure size meaningfully, it is difficult to estimate size in advance, and it is difficult to extrapolate from size to what we are really interested in.

We will first look at methods for estimating size, then at how size can be used to estimate effort (e.g. using COCOMO).

Approaches to Estimating Size

- Through expert consensus (Wideband-Delphi)
- From historical “population” data (Fuzzy logic)
- From standard components (Component estimating)
- From a model of function (Function points)

See Humphrey (2002) for more information on these methods and other more complicated ones.

Wideband-Delphi Estimating

1. Group of experts: $[E_1, \ldots, E_i, \ldots, E_n]$
2. All E_i meet to discuss project
3. Each anonymously estimates size: $[X_1, \ldots, X_i, \ldots, X_n]$
4. Each E_i gets to see all X_j (anonymously)
5. Stop if the estimates are sufficie ntly close together
6. Otherwise, back to step 2

Helps get a group of engineers committed to a particular schedule.

Fuzzy-Logic Estimating

Break previous products into categories by size:

Range	Nominal KLOC	KLOC range included
V Small	2	1 – 4
Small	8	4 – 16
Medium	32	16 – 64
Large	128	64 – 256
V Large	512	256 – 1028

Then look at the previous projects in each category and decide which category contains projects similar to this one.

Problem: Only a very rough estimate, yet requires several relevant historical datapoints in each range (rare)

Standard Component Estimating

- Gather historical data on types and sizes of key components
- For each type (i), guess how many you will need (M_i)
- Also guess largest (L_i) and smallest (S_i) extremes
- Estimated number (E_i) is a function of M_i, L_i and S_i, e.g.: $E_i = (S_i + 4M_i + L_i)/6$
- Total size calculated from estimated number and average size (X_i) of each type: $X = \sum_i E_i X_i$

Helps break down a large project into more-easily guessable chunks.

Function Point Estimating (1)

Popular method based on a weighted count of common functions of software. The fiv e basic functions are:

Inputs : Sets of data supplied by users or other programs

Outputs : Sets of data produced for users or other programs

Inquiries : Means for users to interrogate the system

Data files : Collections of records which the system modifies

Interfaces : Files/databases shared with other systems

Function Point Estimating (2)

Function	Count	Weight	Total
Inputs	8	4	32
Outputs	12	5	60
Inquiries	4	4	16
Data files	2	10	20
Interfaces	1	7	7
Total			135

May adjust function point total using “influence factors”.

Estimating Total Effort

Once we have the size estimate, we can try to estimate the total effort involved, e.g. in person-months, e.g. to decide on staffing levels.

Unfortunately, the total amount of effort required depends on the staffing levels – cf. *The Mythical Man-Month* (Brooks 1995). So it is easy to get stuck in circular reasoning.

Still, with some big assumptions, it is possible to try to use historical experience with similarly sized projects.

COCOMO Model

The Constructive Cost Model (COCOMO; Boehm 1981) is popular for effort estimation. COCOMO is a mathematical equation that can be fit to measurements of effort for different-sized completed projects, providing estimates for future projects.

COCOMO II (Boehm et al. 1995) is the current version (see http://sunset.usc.edu/research/COCOMOII/), but we will focus on the original simpler equation.

All we are hoping to get is a rough (order of magnitude) estimate.

Basic COCOMO Model

In its simplest form COCOMO is:

$$E = C * P^s * M$$

where:

- E is the estimated effort (e.g. in person-months)
- C is a complexity factor
- P is a measure of product size (e.g. KLOC)
- s is an exponent (usually close to 1)
- M is a multiplier to account for project stages

Basic COCOMO Model Examples

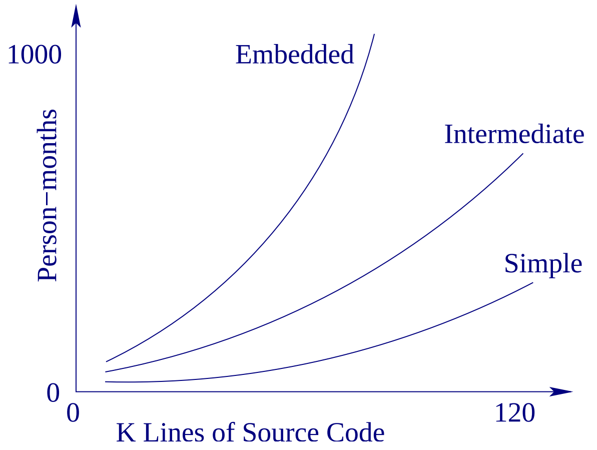
We ignore the multiplier, M, so $E = C * P^s$. Then we fit C and s to historical data from different types of projects:

Simple ($E = 2.4 * P^{1.05}$) : A well understood application developed by a small team.

Intermediate ($E = 3.0 * P^{1.12}$) : A more complex project for which team members have limited experience of related systems.

Embedded ($E = 3.6 * P^{1.20}$) : A complex project in which the software is part of a complex of hardware, software, regulations and operational constraints.

Behavior of the Basic Examples



Extending the COCOMO Model

The basic examples didn't use the multiplier, M.

M can be used to adjust the basic estimate by including expert knowledge of the specific attributes of this project.

Potential attributes/constraints to consider:

- Product attributes (e.g. reliability)
- Computer attributes (e.g. memory constraints)
- Personnel attributes (e.g. programming language experience)
- Project attributes (e.g. project development schedule)

COCOMO Multiplier Example 1

If the basic estimate is 1216 person-months, can add estimates of the effect of various constraints or attributes:

Attribute	**Magnitude**	**Multiplier**
Reliability	V high	1.4
Complexity	V high	1.3
Memory constraint	High	1.2
Tool use	Low	1.1
Schedule	Accelerated	1.23

New E: $1216 * 1.4 * 1.3 * 1.2 * 1.1 * 1.23 = 3593$

COCOMO Multiplier Example 2

Using the basic estimate of 1216 person-months, changing estimates of the constraints/attributes changes the result:

Attribute	**Magnitude**	**Multiplier**
Reliability	V low	0.75
Complexity	V low	0.7
Memory constraint	None	1
Tool use	High	0.9
Schedule	Normal	1

New E: $1216 * 0.75 * 0.7 * 1 * 0.9 * 1 = 575$

COCOMO Limitations 1

Like any mathematical model, COCOMO has two main potential types of error: model error and parameter error.

Model error: Do projects really scale with KLOC as modeled?

From the COCOMO II web site: “The 1998 version of the model has been calibrated to 161 data points [projects]... Over those 161 data points, the ’98 release demonstrates an accuracy of within 30% of actuals 75% of the time”.

Thus even looking retroactively, with accurate KLOC estimates, 25% of projects are more than 30% mis-estimated.

COCOMO Limitations 2

Parameter estimation error: Can the various parameters be set meaningfully?

E.g. result depends crucially on KLOC, which is difficult to estimate accurately.

The other parameters can also be difficult to estimate for a new project, particularly at the beginning when scheduling and feasibility need to be decided.

Estimation Limitations

Model predictions can be sensitive to small changes in parameters, so be sure to perform a sensitivity analysis for different parameter estimates.

In any case, early estimates are likely to be wrong, and should be revised once more data is available.

Also, predictions can strongly affect the outcome:

- If estimate is too high, programmers may relax and work on side issues or exploring many alternatives
- If estimate is too low, quality may be sacrifice d to meet the deadline

Summary

- No size estimation method is foolproof or particularly accurate
- Even once size is available, hard to extrapolate to effort, cost, estimated schedule, etc.
- Estimates can be self-fulfilling or self-defeating
- Thus it is difficult to evaluate how well estimation is working, even retroactively
- Use an appropriate method for how much data you have – if no data, then gut instinct estimation is reasonable
- Try to avoid depending on your estimates being accurate

References

Boehm, B. (1981). *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall.

Boehm, B., Clark, B., Horowitz, E., Madachy, R., Shelby, R., & Westland, C. (1995). Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering*.

Brooks, F. P., Jr. (1995). *The Mythical Man-Month*. Reading, MA: Addison-Wesley. Expanded reprint of 1975 edition.

Humphrey, W. S. (2002). *A Discipline for Software Engineering*. Reading, MA: Addison-Wesley.