Cost Estimation

Outline

- Software size
- Productivity
- Project cost estimation
- Algorithmic cost modelling

Why Cost Estimation

Hardware
Software
Engineers

Project cost $<$ $\geq$ Price to customer

Travelling
Supporting staff
Rent
Others

Software Project Estimation

- Profit vs. Loss
- Success vs. Failure

Software Project Estimation

Project cost estimation ← Cost on engineers

How many engineers

How many months

How many person-months (pm)
man-month (mm)
(Effort estimation)

Measure and Metric

- Measure
  – Provides an extent, amount, dimension, capacity, or size of some attribute of a project or a process
- Metric
  – A quantitative measure of the degree to which a system, component, or process possesses a given attribute
- Indicator
  – A metric or combination of metrics that provide insight into the software process

Measure and Metric

- Quality – an example
  - Number of errors – measure
  - Number of functions – measure
  - Number of errors/Number of functions – metric
  - Mean time to fix a bug – metric
  - The above two metrics can be used to determine the quality of a program – indicator.

Metrics – Software Size

- Types
  - Size-related metrics
    - Lines of code
  - Function-related metrics
    - Function points
    - Object points

Software Size – Lines of Code

- There is NO unique standard on how to count the lines of code
- (Source) Lines of code (LOC, SLOC)
  - Only source lines that are delivered as part of the product are included
  - Source code created by applications generators is excluded
  - One SLOC is one logical line of code (vs. physical line of code)
  - Comments, declarations, blank lines, curly brace lines are not counted as LOC

Software Size – Lines of Code

- KLOC: Thousands of lines of code
- 30 ~ 900 KLOC /person-month (pm)

Software Size – Lines of Code

```
#define ARRAY_SIZE 14
int my_array[ARRAY_SIZE];
void fill_array()
{

    int indx;
    for (indx=0; indx < ARRAY_SIZE; ++indx)
    {
        my_array[indx] = rand();
    }
    /* my_array[ARRAY_SIZE - 1] = ARRAY_SIZE / 3; */
}
```

Software Size – Function Points

- Function points (FPs) (≠ number of functions)
- Based on a combination of program characteristics
  - external inputs
  - external outputs;
  - user interactions;
  - external interfaces;
  - files
- The function point count is modified by complexity of the project
- Function point estimation is subjective.

FPC = (number of each above type $\times$ weight)

Software Size – Function Points

- AVC – AVerage KLOC per function point for a given language
  - 200 ~ 400 for assembly language
  - 2 ~ 4 for advanced language

$$LOC = AVC \times FPC$$

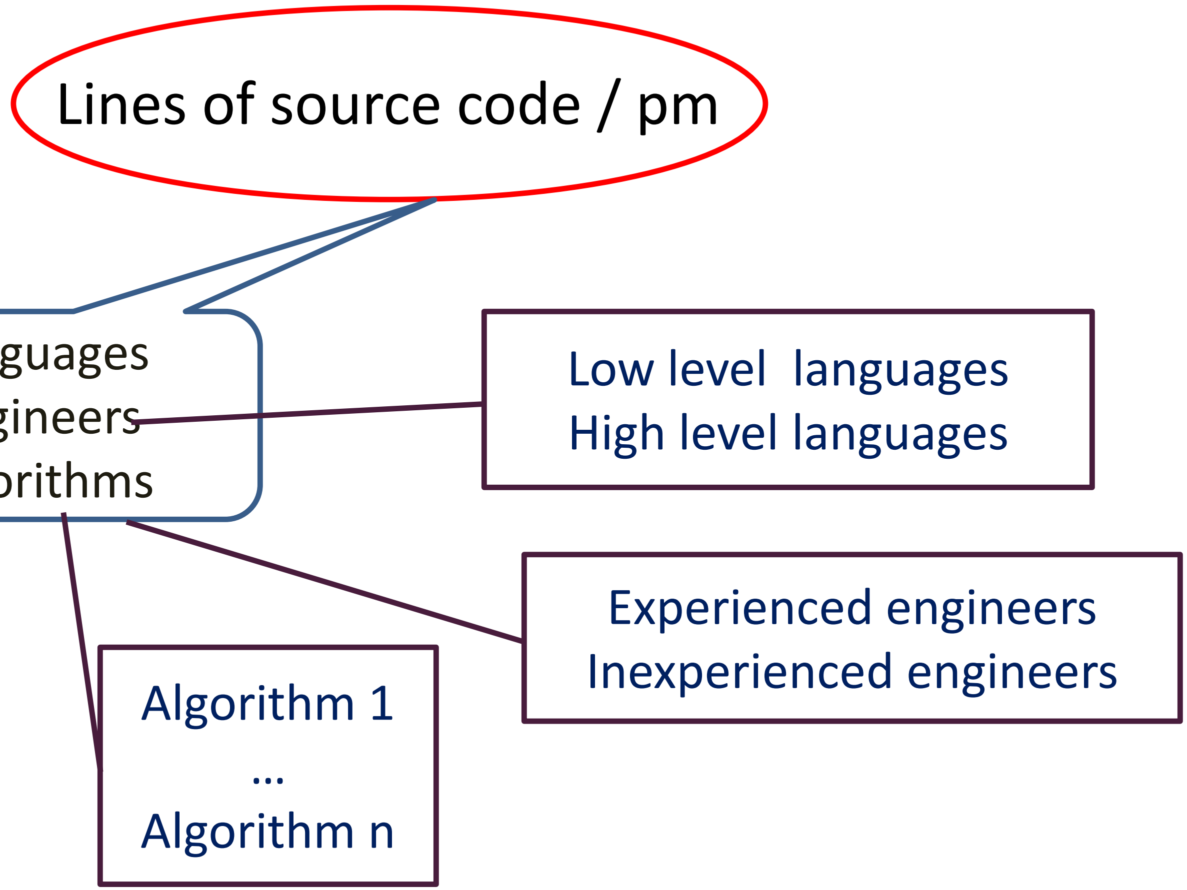
Software Size - Object Points

- An alternative function-related measure to function points
- Object points ≠ number of object classes.
- The number of object points in a program is a weighted estimate of
  - the number of separate screens that are displayed;
  - the number of reports that are produced by the system;
  - the number of program modules that must be developed to supplement the database code

Software Size - Object Points

- Easier to be estimated than function points
- 4 ~ 50 object points /person-month
- Also called application points.

Productivity



pm: person-month

Productivity

Function points or object points

↓

Lines of source code

↓

Productivity

unfair

Other Considerations

- Functionality
- Quality
- Performance
- Maintainability
- Code generator
- Code reuse

unfair

Cost Estimation Techniques

- There are no simple ways to give accurate estimation
  - Different teams, different developers
  - Familiarity with new techniques
  - Supporting tools
  - ...

Cost Estimation Techniques

- Algorithmic cost modelling
  - Use the metric related to the project cost and the model that predicts the effort
- Expert judgement
  - Based on the estimates from experts
- Estimation by analogy
  - Compared with other similar projects
- Parkinson's law
  - Based on the resource rather than the objectives
- Pricing to win
  - The cost is estimated to the available investment from the customer.

Cost Estimation Techniques

- Algorithmic cost modelling
  - Use the metric related to the project cost and the model that predicts the effort
- Expert judgement
  - Based on the estimates from experts
- Estimation by analogy
  - Compared with other similar projects
- Parkinson's law
  - Based on the resource rather than the objectives
- Pricing to win
  - The cost is estimated to the available investment from the customer.

Algorithmic Cost Model

- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers
- $\text{Effort} = A \times \text{Size}^B \times M$
  - A: an organisation-dependent constant
  - B: the exponent (disproportionate effort) for large projects and
  - M: a multiplier reflecting product, process and people attributes.
  - Most models are similar but they use different values for A, B, and M

COCOMO 2

- COstructive COst MOdel (COCOMO)

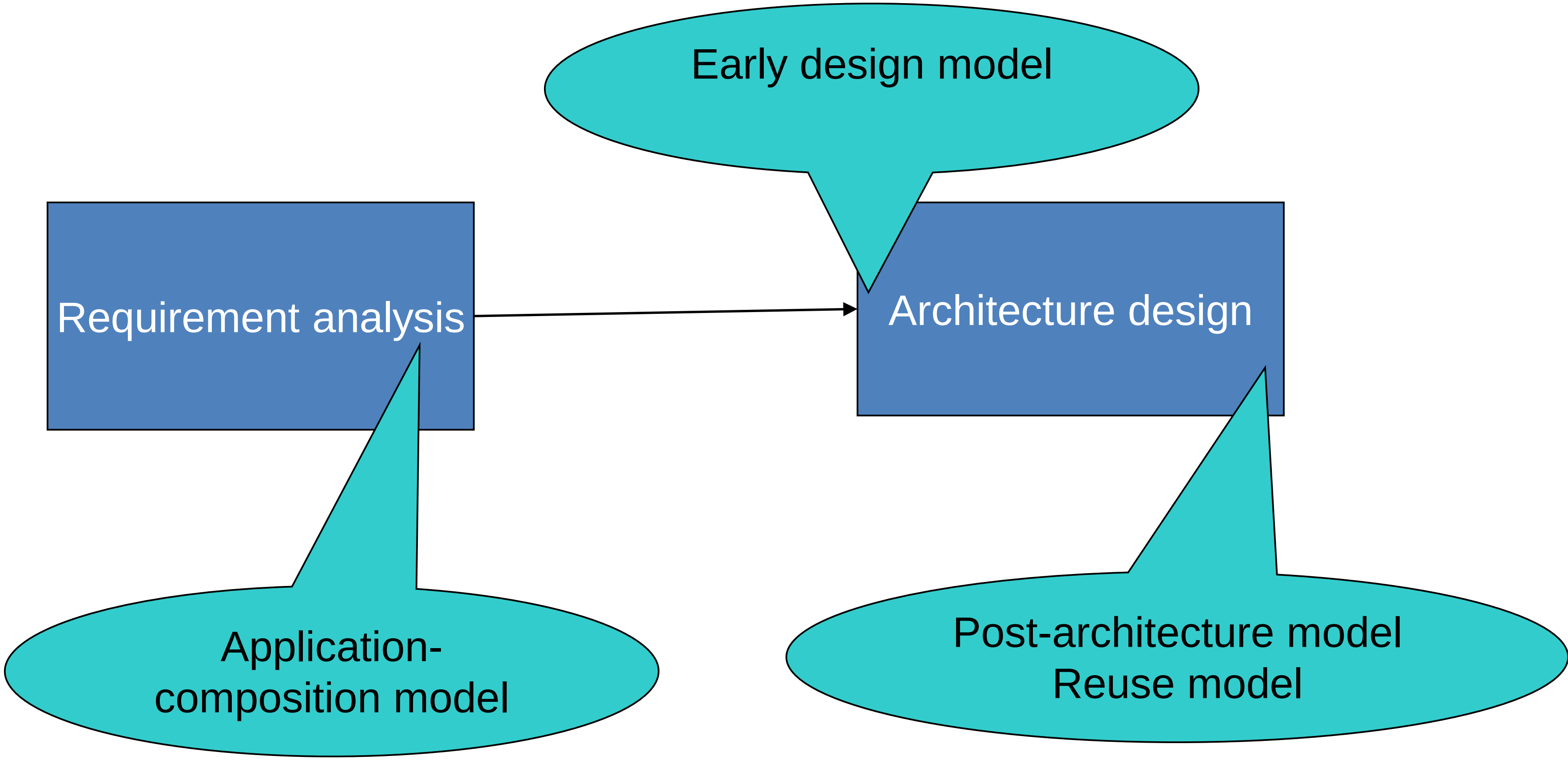
COCOMO 2

- COstructive COst MOdel (COCOMO)
- An empirical model based on experiences
- Initial version – 1981
- COCOMO 2 – 2000
  - Embraces the different approaches to software development
    - Prototyping, spiral, component composition
  - Produce detailed software estimates
  - Incorporate a set of sub-models
  - The application of these sub-models depends on the real development environments.

COCOMO 2

- Sub-models
  - Application composition model
  - Early design model
  - Reuse model
  - Post-architecture model.

COCOMO 2



Early design model
Requirement analysis
Architecture design
Application-composition model
Post-architecture model
Reuse model

Application Composition Model

- Supports prototyping projects and projects where there is extensive reuse
- Based on standard estimates of developer productivity in object-points/person-month
  - Takes CASE tool use into account.
- PM = (NOP × (1 - *%reuse/100*)) / PROD
  - PM : Person-Month
  - NOP: Number of Object Points (Application Points, NAP)
  - PROD: Productivity (number of object points per person per month)
  - *%reuse/100*: Percentage of reused object points

Application Composition Model

- PROD

Maturity and capability of process and developers	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

Application Composition Model

NOP: 1000
Reused object points: 25%
The productivity level: high

$$
\begin{aligned}
PM &= (NOP \times (1 - \%reuse/100)) / PROD \\
&= 1000 \times (1 - 25\%) / 25 \\
&= 30 \text{ person-months}
\end{aligned}
$$

Early Design Model

- Used when requirements are available but design has not yet started.
- Estimates can be made after the requirements have been agreed.
- $PM = A \times Size^{B} \times M$
  - A = 2.94
  - Size: KLOC
  - B: 1 ~ 1.24
    - novelty of the project
    - development flexibility
    - risk management approaches
    - the process maturity
  - M = PERS $\times$ RCPX $\times$ RUSE $\times$ PDIF $\times$ PREX $\times$ FCIL $\times$ SCED.

The Exponent Term B

Scale Factors (W_i)	Very Low (5)	Low (4)	Nominal (3)	High (2)	Very High (1)	Extra High (0)
Precedentedness	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely famil- iar	throughly familiar
Development Flexibility	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
Architecture / risk resolution*	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
Team cohesion	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
Process maturity†	Weighted average of "Yes" answers to CMM Maturity Questionnaire					

$$B = 1.01 + 0.01 \Sigma W_i$$

Early Design Model

- $M = PERS \times RCPX \times RUSE \times PDIF \times PREX \times FCIL \times SCED$

  (Cost drivers: very low 1 – extra high 6)

  - RCPX: product reliability and complexity;
  - RUSE: the reuse required;
  - PDIF: platform difficulty;
  - PREX: personnel experience;
  - PERS: personnel capability;
  - SCED: required schedule;
  - FCIL: the team support facilities.

The Reuse Model

- Used to compute the effort of integrating reusable components
- Two versions
  - Components reused without change
  - Components understood and integrated
    - The equivalent new source code must be estimated.

The Reuse Model (1)

- For code that does not need to be changed
  - PM = (ASLOC × %AT/100)/ATPROD
  - ASLOC : the total number of lines of adapted code
  - %AT/100: the percentage of adapted code that does not change (or can be automatically generated)
  - ATPROD: the productivity of engineers in integrating this code.

The Reuse Model (1)

ASLOC: 20 KLOC
AT: 30%
ATPROD: 2.4 KLOC/month

The effort required to integrate this code is

$$PM = (ASLOC * \%AT/100)/ATPROD$$
$$= (20 \times 30\%) / 2.4$$
$$= 2.5 \text{ person-months}$$

The Reuse Model (2)

- For code that has to be understood and integrated
  - $ESLOC = ASLOC \times (1\text{-}\%AT/100) * AAM$
  - ESLOC: The equivalent number of lines of new source code
  - ASLOC : the total number of lines of adapted code
  - %AT/100: the percentage of code that does not need to be understood and integrated (e.g., does not change or can be automatically generated)
  - AAM: the adaptation adjustment multiplier

The Reuse Model (2)

ASLOC : 20 KLOC
AT: 25%
AAM: 1.0

ESLOC = ASLOC * (1-%AT/100) * AAM
= (20 $\times$ (1 – 25%)) *1.0
= 15 KLOC

Post-Architecture Model

- Used once the system architecture has been designed and more information about the system is available
- $PM = A \times Size^{B} \times M$

Post-Architecture Model

- $PM = A \times Size^B \times M$
  - A = 2.94
  - Size: KLOC
    - the number of lines of new code to be developed (NSLOC);
    - the equivalent number of lines of new code computed using the reuse model (ESLOC);
    - the number of lines of code that have to be modified according to requirements changes (MSLOC)
  - B: Same as used in Early Design Model

Post-Architecture Model

- $PM = A \times Size^{B} \times M$
  - $M = PERS \times RCPX \times RUSE \times PDIF \times PREX \times FCIL \times SCED$ ...
  - Product attributes
    - Concerned with required characteristics of the software product being developed.
  - Computer attributes
    - Constraints imposed on the software by the hardware platform.
  - Personnel attributes
    - Multipliers that take the experience and capabilities of the people working on the project into account.
  - Project attributes
    - Concerned with the particular characteristics of the software development project.

Post-Architecture Model

NSLOC: 50 KLOC
ESLOC: 20 KLOC
MSLOC: 18 KLOC
B: 1.17
RELY: 1.39
CPLX: 1.3
STOR: 1.21
TOOL: 1.12
SCED: 1.29

$$PM = A \times Size^{B} \times M$$
$$= 2.94 \times (50+20+18)^{1.17} \times 1.39 \times 1.3 \times 1.21 \times 1.12 \times 1.$$
$$= 1750 \text{ person-months}$$

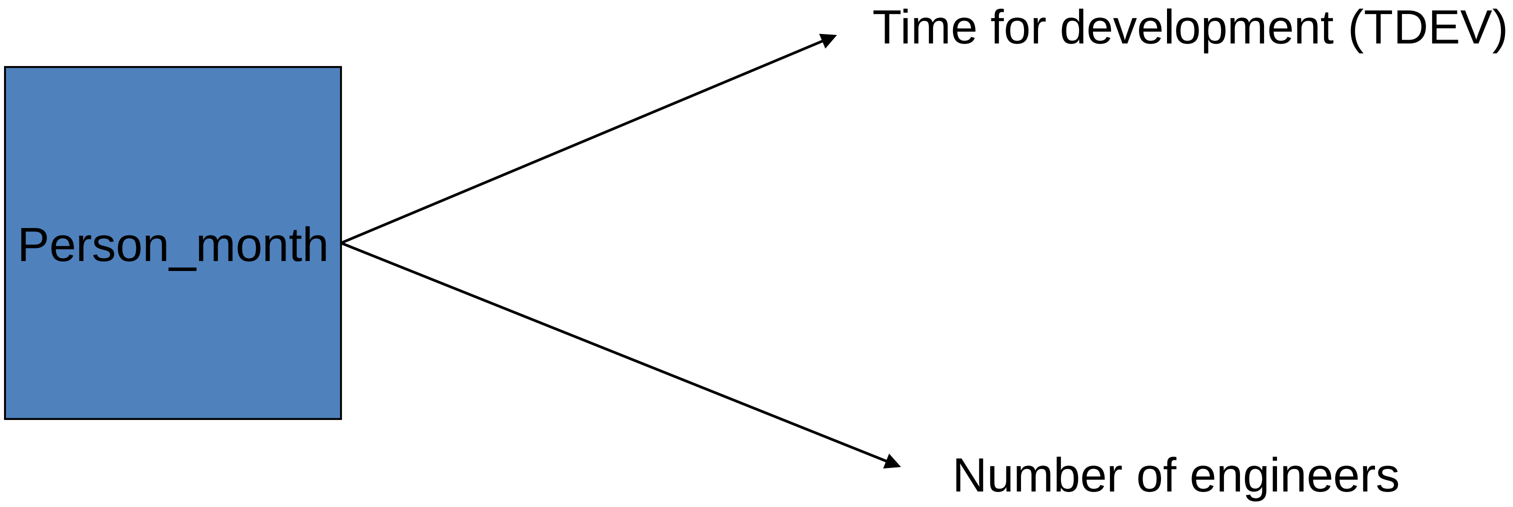
Pricing to Win

- The project costs whatever the customer has to spend on it.
- Advantages
  - You get the contract
  - when detailed information is lacking it may be the only appropriate strategy
- Disadvantages
  - The probability that the customer gets the system he or she wants is small. Costs do not accurately reflect the work required
  - This approach may seem unethical and un-businesslike
  - The project cost is agreed on the basis of an outline proposal and the development is constrained by that cost.

Pricing to Win

Can it be like this….

Staffing and Calendar Time



Staffing and Calendar Time

12 pm (person_month)

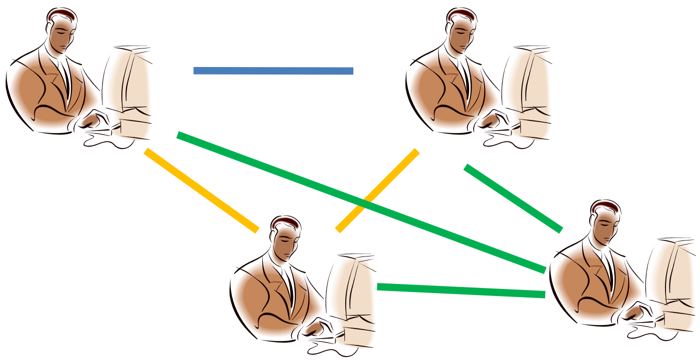
1 staff × 12 months

2 staff × 6 months

12 staff × 1 month

240 staff × 1 day

Staffing and Calendar Time



Staffing and Calendar Time (Nominal Schedule)

$$\text{Effort} = A \times \text{Size}^{B} \times M$$

$$\text{TDEV} = 3 \times (\text{PM})^{(0.33 + 0.2 \times (B - 1.01))}$$

E.g., PM = 60, B = 1.17

$$\text{TDEV} = 3 \times (\text{PM})^{(0.33 + 0.2 \times (1.17 - 1.01))} = 13 \text{ months}$$

Number of staffs: 60 / 13 = 4.6

Class Exercise

How many software engineers can be allocated to the development of the project (project description is available in iSpace) according to the COCOMO model?

Tools

Tools:

http://www.softstarsystems.com/demo.htm

Summary

- We can use different ways to measure size of software
- According to the size and other factors, we can roughly estimate the cost of the project and the staff number
- Four sub-models under algorithmic cost modelling
  - Application composition model
  - Early design model
  - Reuse model
  - Post-architecture model.