Software Engineering

Software Cost Estimation

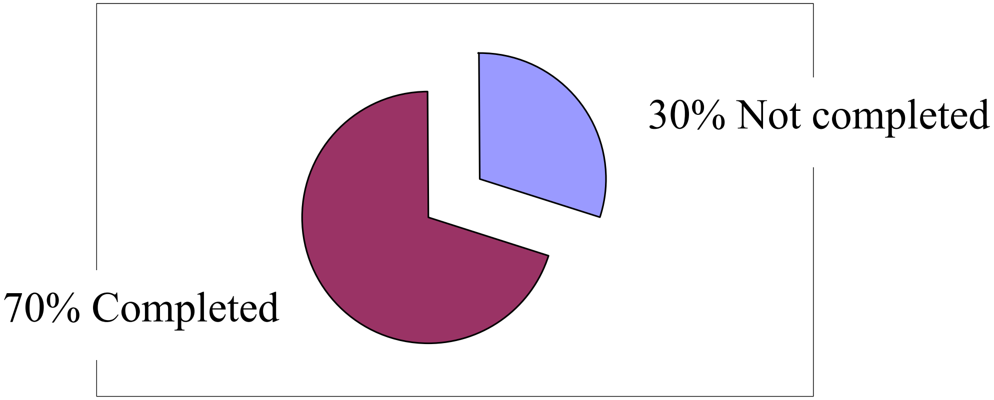
Objectives

- To introduce the fundamentals of software costing and pricing
- To explain software productivity metric
- To explain why different techniques for software estimation:
  - LOC model
  - Function points model
  - Object point model
  - COCOMO (COnstructive COst MOdel): 2 algorithmic cost estimation model
  - UCP: Use Case Points

What is Software Cost Estimation

- Predicting the cost of resources required for a software development process

Software is a Risky Business



- 53% of projects cost almost 200% of original estimate.
- Estimated $81 billion spent on failed U.S. projects in 1995.
  - All surveyed projects used waterfall lifecycle.

Software is a Risky Business

- British Computer Society (BCS) survey:
  - 1027 projects
  - Only 130 were successful !
- Success was defined as:
  - deliver all system **requirements**
  - within **budget**
  - within **time**
  - to the **quality** agreed on

Why early Cost Estimation?

- Cost estimation is needed early for s/w pricing
- S/W price = cost + profit

Fundamental estimation questions

- **Effort**
  - How much effort is required to complete an activity?
  - Units: man-day (person-day), man-week, man-month,,..
- **Duration**
  - How much calendar time is needed to complete an activity? Resources assigned
  - Units: hour, day, week, month, year,..
- **Cost of an activity**
  - What is the total cost of an activity?
- Project estimation and scheduling are interleaved management activities

Software Cost Components

1. Effort costs (dominant factor in most projects)
   - salaries
   - Social and insurance & benefits
2. Tools costs: Hardware and software for development
   - Depreciation on relatively small # of years 300K US$
3. Travel and Training costs (for particular client)
4. Overheads(OH): Costs must take overheads into account
   - costs of building, air-conditioning, heating, lighting
   - costs of networking and communications (tel, fax,)
   - costs of shared facilities (e.g library, staff restaurant, etc.)
   - depreciation costs of assets
   - Activity Based Costing (ABC)

S/W Pricing Policy

S/W price is influenced by

- economic consideration
- political consideration
- and business consideration

Software Pricing Policy/Factors

Factor	Description
Market opportunity	A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed.
Cost estimate uncertainty	If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.
Requirements volatility	If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices may be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a small profit or break even than to go out of business.

Programmer Productivity

- Rate of s/w production
  - Needs for measurements
  - Measure software produced per time unit (Ex: LOC/hr)
    - rate of s/w production
    - software produced including documentation

- Not quality-oriented: although quality assurance is a factor in productivity assessment

Productivity measures

S/W productivity measures are based on:

- Size related measures:
  - Based on some output from the software process
  - Number lines of delivered source code (LOC)
- Function-related measures
  - based on an estimate of the functionality of the delivered software:
    - Function-points (are the best known of this type of measure)
    - Object-points
    - UCP

Measurement problems

- Estimating the size of the measure
- Estimating the total number of programmer-months which have elapsed
- Estimating contractor productivity (e.g. documentation team) and incorporating this estimate in overall estimate

Lines Of Code (LOC)

- Program length (LOC) can be used to predict program characteristics e.g. person-month effort and ease of maintenance
- What's a line of code?
  - The measure was first proposed when programs were typed on cards with one line per card
  - How does this correspond to statements as in Java which can span several lines or where there can be several statements on one line?
- What programs should be counted as part of the system?
- Assumes linear relationship between system size and volume of documentation

Versions of LOC

- DSI : Delivered Source Instructions
- KLOC Thousands of LOC
- DSI
  - One instruction is one LOC
  - Declarations are counted
  - Comments are not counted

LOC

- Advantages
  - Simple to measure
- Disadvantages
  - Defined on code: it can not measure the size of specification
  - Based on one specific view of size: length.. What about complexity and functionality !!
  - Bad s/w may yield more LOC
  - Language dependent
- Therefore: Other s/w size attributes must be included

LOC Productivity

- The lower level the language, the less productive the programmer
  - The same functionality takes more code to implement in a lower-level language than in a high-level language

- Measures of productivity *based on LOC* suggest that programmers who write verbose code are more productive than programmers who write compact code !!!

Function Points: FP

Function Points is used in 2 contexts:

- Past: To develop **metrics** from historical data
- Future: Use of available metrics to size the s/w of a new project

Function Points

- Based on a combination of program characteristics
- The number of :
  - External (user) inputs: input transactions that update internal files
  - External (user) outputs: reports, error messages
  - User interactions: inquiries
  - Logical internal files used by the system:

    Example a purchase order logical file composed of 2 physical files/tables Purchase_Order and Purchase_Order_Item
  - External interfaces: files shared with other systems
- A weight (ranging from 3 for simple to 15 for complex features) is associated with each of these above
- The function point count is computed by multiplying each raw count by the weight and summing all values

Function Points - Calculation

measurement parameter	count		weighting factor 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 →							☐

Function Points – Taking Complexity into Account -14 Factors Fi

Each factor is rated on a scale of:

Zero: not important or not applicable

Five: absolutely essential

1. Backup and recovery
2. Data communication
3. Distributed processing functions
4. Is performance critical?
5. Existing operating environment
6. On-line data entry
7. Input transaction built over multiple screens

Function Points – Taking Complexity into Account -14 Factors Fi (cont.)

8. Master files updated on-line
9. Complexity of inputs, outputs, files, inquiries
10. Complexity of processing
11. Code design for re-use
12. Are conversion/installation included in design?
13. Multiple installations
14. Application designed to facilitate change by the user

Function Points – Taking Complexity into Account -14 Factors Fi (cont.)

$$FP = UFC * \left[0.65 + 0.01 * \sum_{i=1}^{i=14} F_i \right]$$

UFC: Unadjusted function point count

$0 <= F_i <= 5$

FP: Advantages & Disadvantages

- Advantages
  - Available early .. We need only a detailed specification
  - Not restricted to code
  - Language independent
  - More accurate than LOC

- Disadvantages
  - Ignores quality issues of output
  - Subjective counting .. depend on the estimator
  - Hard to automate.. Automatic function-point counting is impossible

Function points and LOC

- FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language
  - LOC = AVC * number of function points
  - AVC is a language-dependent factor varying from approximately 300 for assemble language to 12-40 for a 4GL

Relation Between FP & LOC

Programming Language	LOC/FP (average)
Assembly language	320
C	128
COBOL	106
FORTRAN	106
Pascal	90
C++	64
Ada	53
Visual Basic	32
Smalltalk	22
Power Builder (code generator)	16
SQL	12

Function Points & Normalisation

- Function points are used to normalise measures (same as for LOC) for:
  - S/w productivity
  - Quality

- Error (bugs) per FP (discovered at programming)
- Defects per FP (discovered after programming)
- $ per FP
- Pages of documentation per FP
- FP per person-month

Expected Software Size

- Based on three-point
- Compute Expected Software Size (S) as weighted average of:
  - Optimistic estimate: S(opt)
  - Most likely estimate: S(ml)
  - Pessimistic estimate: S(pess)

- $S = \{ S(opt) + 4\, S(ml) + S(pess) \} / 6$

  - Beta probability distribution

Example 1: LOC Approach

- A system is composed of 7 subsystems as below.
- Given for each subsystem the size in LOC and the 2 metrics: productivity LOC/pm (pm: person month) ,Cost $/LOC
- Calculate the system total cost in $ and effort in months .

Functions	estimated LOC	LOC/pm	$/LOC
UICF	2340	315	14
2DGA	5380	220	20
3DGA	6800	220	20
DSM	3350	240	18
CGDF	4950	200	22
PCF	2140	140	28
DAM	8400	300	18

Example 1: LOC Approach

Functions	estimated LOC	LOC/pm	$/LOC	Cost	Effort (months)
UICF	2340	315	14	32,000	7.4
2DGA	5380	220	20	107,000	24.4
3DGA	6800	220	20	136,000	30.9
DSM	3350	240	18	60,000	13.9
CGDF	4950	200	22	109,000	24.7
PCF	2140	140	28	60,000	15.2
DAM	8400	300	18	151,000	28.0
Totals	33,360			655,000	145.0

Example 2: LOC Approach

Assuming

- Estimated project LOC = 33200
- Organisational productivity (similar project type) = 620 LOC/p-m
- Burdened labour rate = 8000 $/p-m

Then

- Effort = 33200/620 = (53.6) = 54 p-m
- Cost per LOC = 8000/620 = (12.9) = 13 $/LOC
- Project total Cost = 8000 * 54 = 432000 $

Example 3: FP Approach

	A	B	C	D	E	F	G
1	Info Domain	Optimistic	Likely	Pessim.	Est Count	Weight	FP count
2	# of inputs	22	26	30	26	4	104
3	# of outputs	16	18	20	18	5	90
4	# of inquiries	16	21	26	21	4	84
5	# of files	4	5	6	5	10	50
6	# of external interf	1	2	3	2	7	14
7	UFC: Unadjusted Function Count						342
8			Complexity adjustment factor				1.17
9						FP	400

Example 3: FP Approach (cont.) Complexity Factor

Complexity factor: Fi	value=0	value=1	value=2	value=3	value=4	value=5	Fi
Backup and recovery	0	0	0	0	1	0	4
Data communication	0	0	1	0	0	0	2
Distributed processing functions	0	0	0	0	0	0	0
Is performance critical?	0	0	0	0	1	0	4
Existing operating environment	0	0	0	1	0	0	3
On-line data entry	0	0	0	0	1	0	4
Input transaction built over multiple screens	0	0	0	0	0	1	5
Master files updated on-line	0	0	0	1	0	0	3
Complexity of inputs, outputs, files, inquiries	0	0	0	0	0	1	5
Complexity of processing	0	0	0	0	0	1	5
Code design for re-use	0	0	0	0	1	0	4
Are conversion/installation included in design?	0	0	0	1	0	0	3
Multiple installations	0	0	0	0	0	1	5
Application designed to facilitate change by the user	0	0	0	0	0	1	5
						Sigma (F)	52
Complexity adjustment factor	0.65 + 0.01 * Sigma (F) =			1.17			

Example 3: FP Approach (cont.)

Assuming $\sum_i F_i = 52$

$$FP = UFC * [\ 0.65 + 0.01 * \sum_i F_i\ ]$$

$$FP = 342 * 1.17 = 400$$

Complexity adjustment factor = 1.17

Example 4: FP Approach (cont.)

Assuming

- Estimated FP = 401
- Organisation average productivity (similar project type) = 6.5 FP/p-m (person-month)
- Burdened labour rate = 8000 $/p-m

Then

- Estimated effort = 401/6.5 = (61.65) = 62 p-m
- Cost per FP = 8000/6.5 = 1231 $/FP
- Project cost = 8000 * 62 = 496000 $

Object Points (for 4GLs)

- Object points are an alternative function-related measure to function points when 4Gls or similar languages are used for development
- Object points are NOT the same as 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 3GL modules that must be developed to supplement the 4GL code
  - C:\Software_Eng\Cocomo\Software Measurement Page, COCOMO II, object points.htm

Object Points – Weighting

Object Type	Simple	Meduim	Difficult
Screen	1	2	3
Report	2	5	8
Each 3GL module	10	10	10

Object Points – Weighting (cont.)

- ***srvr***: number of server data tables used with screen/report
- **clnt:** number of client data tables used with screen/report

For Screens				For Reports			
Number of Views contained	# and source of data tables			Number of Sections contained	# and source of data tables		
	Total < 4 (< 2 srvr < 3 clnt)	Total < 8 (2/3 srvr 3-5 clnt)	Total 8+ (> 3 srvr > 5 clnt)		Total < 4 (< 2 srvr < 3 clnt)	Total < 8 (2/3 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

Object Point Estimation

- Object points are easier to estimate from a specification than function points
  - simply concerned with screens, reports and 3GL modules
- At an early point in the development process:
  - Object points can be easily estimated
  - It is very difficult to estimate the number of lines of code in a system

Productivity Estimates

- LOC productivity
  - Real-time embedded systems, 40-160 LOC/P-month
  - Systems programs , 150-400 LOC/P-month
  - Commercial applications, 200-800 LOC/P-month
- Object points productivity
  - measured 4 - 50 object points/person-month
  - depends on tool support and developer capability

Developer's experience and Capability / ICASE maturity and capability	Very low	Low	Nominal	High	Very high
PROD: Productivity Object-point per person-month	4	7	13	25	50

Object Point Effort Estimation

- Effort in p-m = NOP / PROD
  - NOP = number of OP of the system
  - Example: An application contains 840 OP (NOP=840) & Productivity is very high (= 50)
  - Then, Effort = 840/50 = (16.8) = 17 p-m

Adjustment for % of Reuse

- Adjusted NOP = NOP * (1 - % reuse / 100)
- Example:
  - An application contains 840 OP, of which 20% can be supplied by existing components.

Adjusted NOP = 840 * (1 – 20/100) = 672 OP

Adjusted effort = 672/50 = (13.4) = 14 p-m

Factors affecting productivity

Factor	Description
Application domain experience	Knowledge of the application domain is essential for effective software development. Engineers whoalready understand a domain are likely to be the most productive.
Process quality	The development process used can have a significant effect on productivity. This is covered in Chapter 31.
Project size	The larger a project, the more time required for team communications. Less time is available for development so individual productivity is reduced.
Technology support	Good support technology such as CASE tools, supportive configuration management systems, etc. can improve productivity.
Working environment	As discussed in Chapter 28, a quiet working environment with private work areas contributes to improved productivity.

Quality and Productivity

- All metrics based on **volume/unit** time are flawed because they do not take quality into account
- Productivity may generally be increased at the cost of quality
- If change is constant, then an approach based on *counting lines of code* (LOC) is not meaningful

Estimation techniques

- There is no simple way to make an accurate estimate of the effort required to develop a software system:
  - Initial estimates may be based on inadequate information in a user requirements definition
  - The software may run on unfamiliar computers or use new technology
  - The people in the project may be unknown
- Project cost estimates may be self-fulfilling
  - The estimate defines the budget and the product is adjusted to meet the budget

Estimation techniques

- Algorithmic cost modelling
- Expert judgement
- Estimation by analogy
- Parkinson's Law
- Pricing to win

Algorithmic code modelling

- A formula – empirical relation:
  - based on historical cost information and which is generally based on the size of the software
- The formulae used in a formal model arise from the analysis of historical data.

Expert Judgement

- One or more experts in both software development and the application domain use their experience to predict software costs. Process iterates until some consensus is reached.
- Advantages: Relatively cheap estimation method. Can be accurate if experts have direct experience of similar systems
- Disadvantages: Very inaccurate if there are no experts!

Estimation by Analogy

- Experience-based Estimates
- The cost of a project is computed by comparing the project to a similar project in the same application domain
- Advantages: Accurate if project data available
- Disadvantages: Impossible if no comparable project has been tackled. Needs systematically maintained cost database

Estimation by Analogy : Problems

- However, new methods and technologies may make estimating based on experience inaccurate:
  - Object oriented rather than function-oriented development
  - Client-server systems rather than mainframe systems
  - Off the shelf components
  - Component-based software engineering
  - CASE tools and program generators

Parkinson's Law

- “The project costs whatever resources are available”
  (Resources are defined by the software house)
- Advantages: No overspend
- Disadvantages: System is usually unfinished
  - The work is contracted to fit the budget available: by reducing functionality, quality

Pricing to Win

- The project costs whatever the customer budget is.
- Advantages: You get the contract
- Disadvantages:
  - The probability that the customer gets the system he/she wants is small.
  - Costs do not accurately reflect the work required

Pricing to Win

- This approach may seem unethical and unbusiness like
- However, when detailed information is lacking it may be the only appropriate strategy
- The project cost is agreed on the basis of an outline proposal and the development is constrained by that cost
- A detailed specification may be negotiated or an evolutionary approach used for system development

Top-down and Bottom-up Estimation

- Top-down
  - Start at the system level and assess the overall system functionality

- Bottom-up
  - Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate

Top-down Estimation

- Usable *without* knowledge of the system architecture and the components that might be part of the system
- Takes into account costs such as integration, configuration management and documentation
- Can underestimate the cost of solving difficult low-level technical problems

Bottom-up estimation

- Usable when
  - the architecture of the system is known and
  - components identified
- Accurate method if the system has been designed in detail
- May underestimate costs of system level activities such as integration and documentation

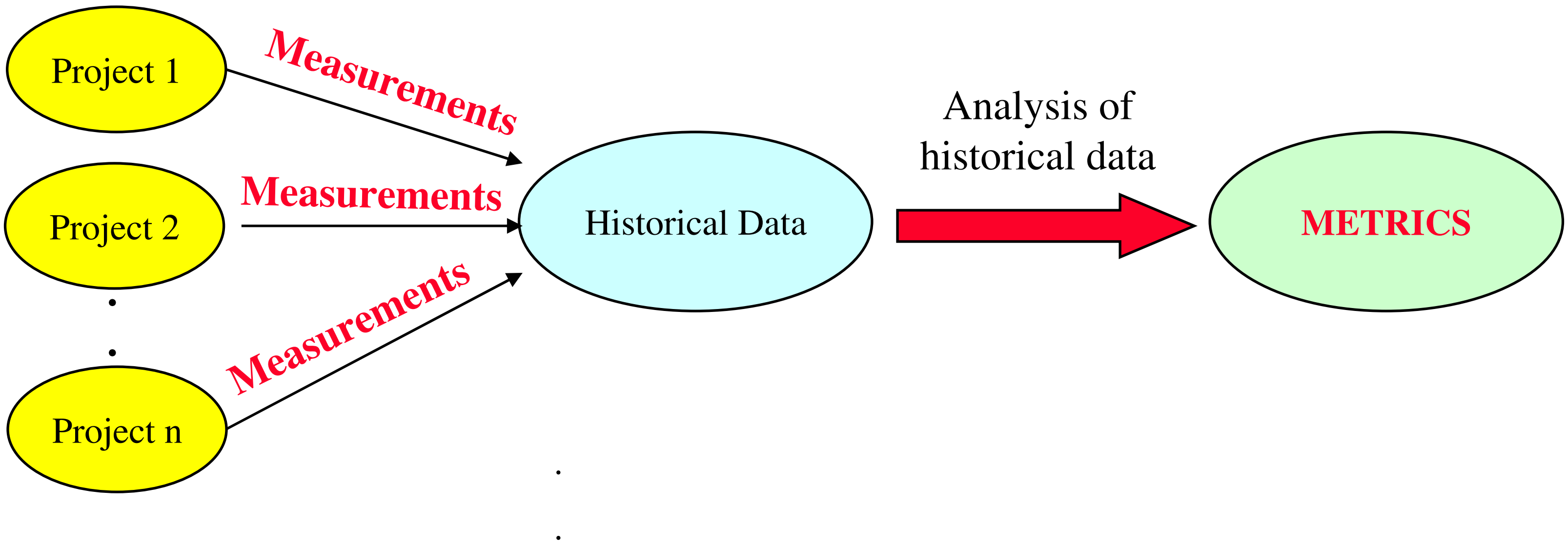
Estimation Methods

- S/W project estimation should be based on several methods
- If these do not return approximately the same result, there is insufficient information available
- Some action should be taken to find out more in order to make more accurate estimates
- Pricing to win is sometimes the only applicable method

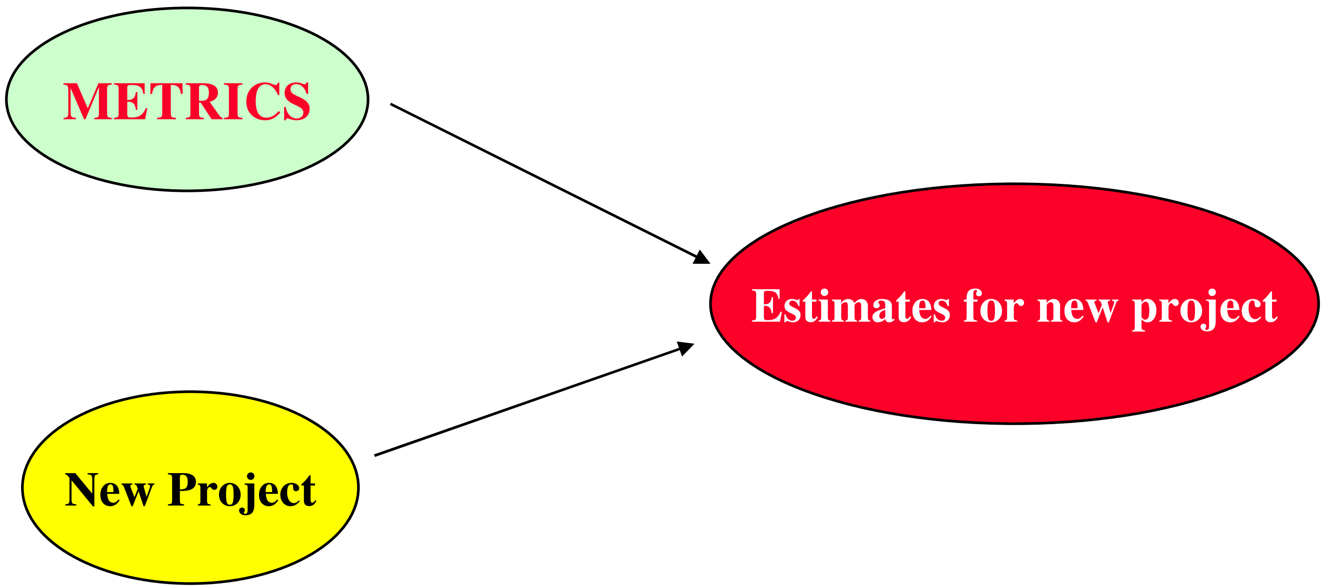
Algorithmic Cost Modelling

- Most of the work in the cost estimation field has focused on algorithmic cost modelling.
- Costs are analysed using mathematical formulas linking costs or inputs with METRICS to produce an estimated output.
- The formula is based on the analysis of historical data.
- The accuracy of the model can be improved by calibrating the model to your specific development environment, (which basically involves adjusting the weighting parameters of the metrics).

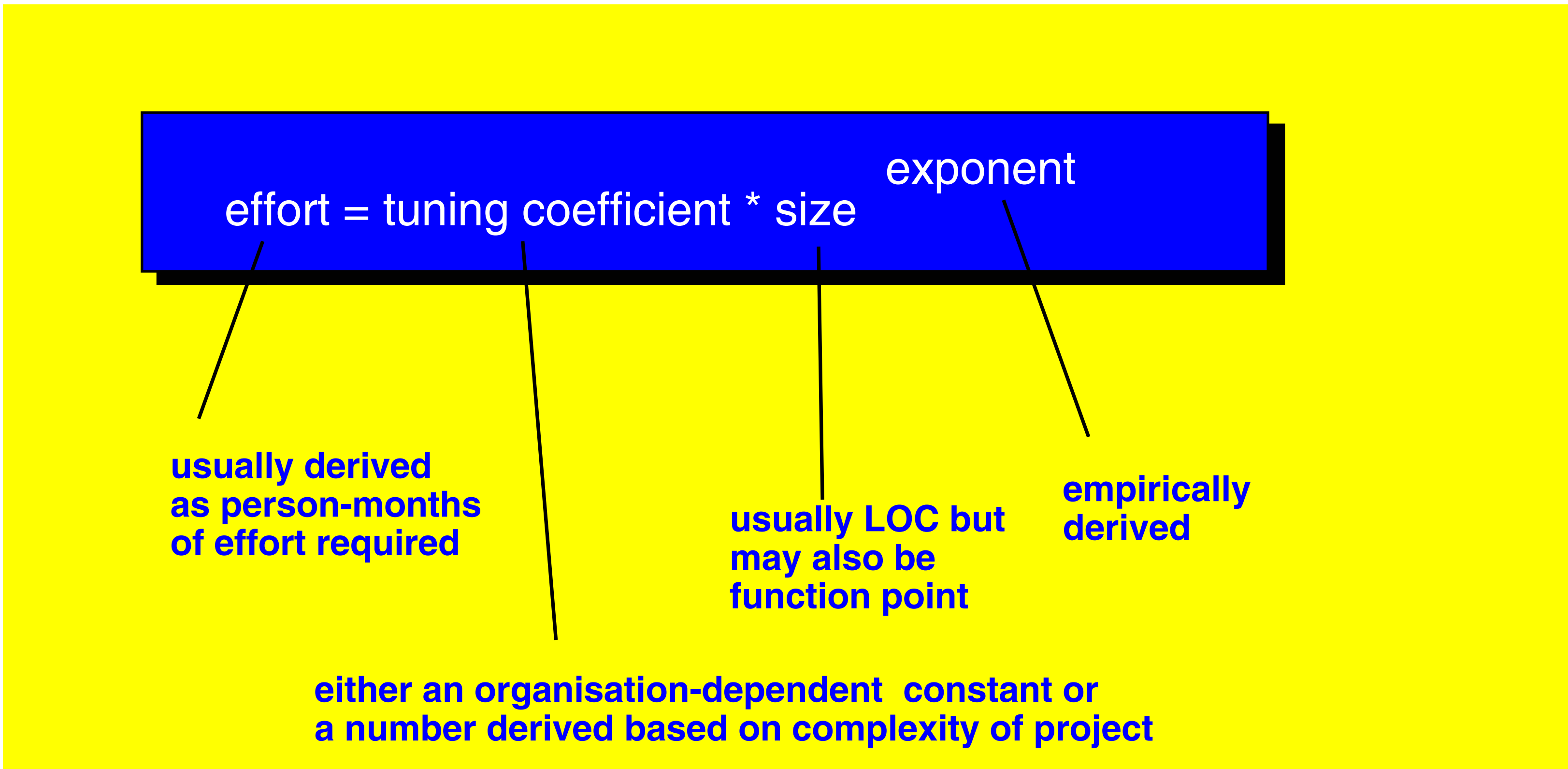
Building Metrics from measurements



New Project estimation using available Metrics



Empirical Estimation Models - Algorithmic Cost Modelling



Algorithmic Cost Modelling

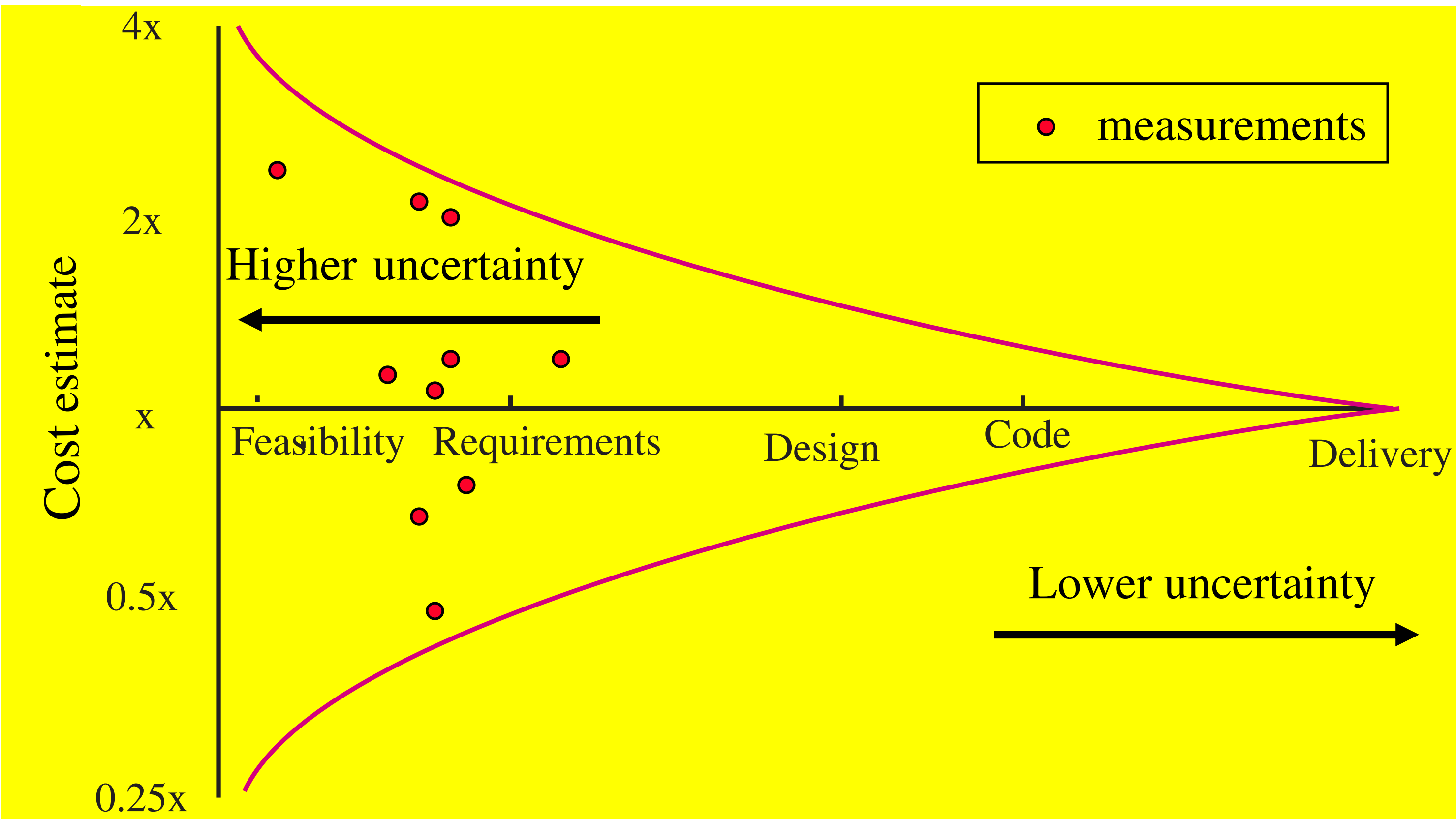
- $\text{Effort} = A \times \text{Size}^B \times M$
- A is an organisation-dependent constant
- B reflects the nonlinearity (disproportionate) effort for large projects
- M is a multiplier reflecting product, process and people attributes

- Most commonly used product attribute for cost estimation is code size (LOC)
- Most models are basically similar but with different values for A, B and M

Estimation Accuracy

- The size of a software system can only be known accurately when it is finished
- Several factors influence the final size
  - Use of COTS and components
  - Programming language
  - Distribution of system
- As the development process progresses then the size estimate becomes more accurate

Estimate Uncertainty



The COCOMO Cost model
Constructive Cost Model

- An empirical model based on project experience
- COCOMO'81 is derived from the analysis of 63 software projects in 1981.
- Well-documented, ‘independent’ model which is not tied to a specific software vendor
- COCOMO II (2000) takes into account different approaches to software development, reuse, etc.

COCOMO 81

Project complexity	Formula	Description
Simple (Organic)	$PM = 2.4\ (KDSI)^{1.05} \times M$	Well-understood applications developed by small teams.
Moderate (Semi-detached)	$PM = 3.0\ (KDSI)^{1.12} \times M$	More complex projects where team members may have limited experience of related systems.
Embedded	$PM = 3.6\ (KDSI)^{1.20} \times M$	Complex projects where the software is part of a strongly coupled complex of hardware, software, regulations and operational procedures.

M: multiplier similar as for COCOMO II, based on 15 cost drivers
KDSI: Thousands of Delivered Source Instructions (KLOC)

Metrics: Parameters calculations

- Least Squares method – Curve fitting

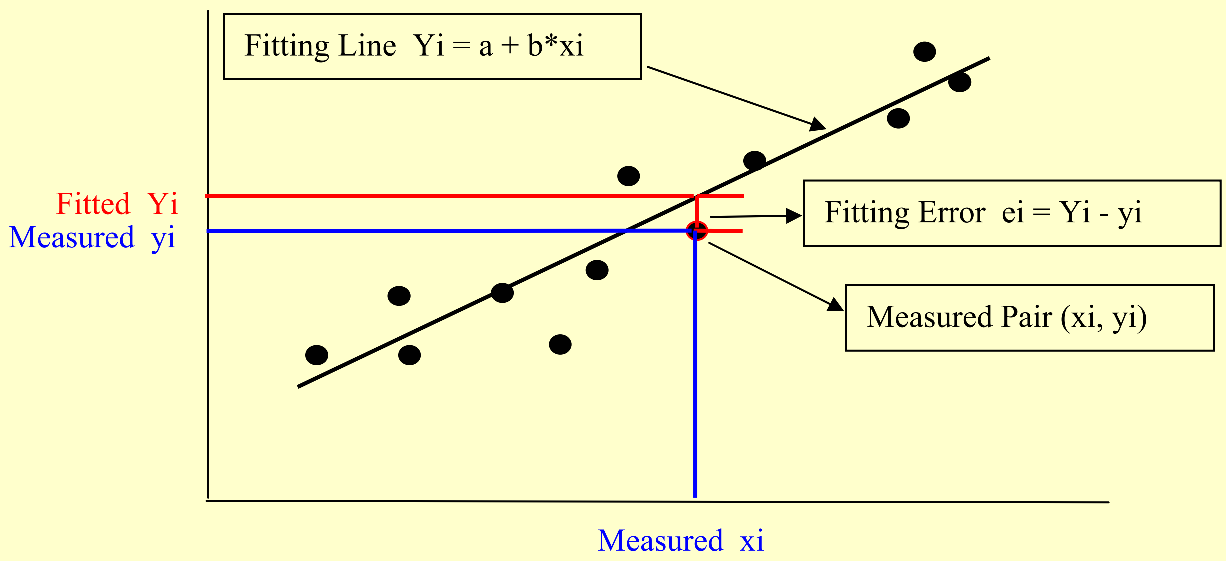
- **Given**: n measurements of pairs (xi, yi)
- **Required**: Best fit of measurements to get metrics parameters

- Assume: A linear relation between measured pairs:
  $Y = a + b x$

  Other relations may be assumed as quadratic 'or higher':
  $Y = a + b x + c x*x$, …

- Get metrics parameters a, b that best fit the measurements

How to get parameters a, b



How to get parameters a, b

- $ei = Yi – yi = a + b*xi - yi$

- For all measurements get S as:

  S is the sum mover n measurements of squared values of ei

- $S = \Sigma (ei)^2 = \Sigma (a + b*xi - yi)^2$
- $S = S(a, b)$

How to get parameters a, b

- Best fitting when S is minimum
- S is minimum when both the partial derivatives of S with respect to a and b are zero.
- This leads to 2 equations in a and b.
- Solve and get a and b.

COCOMO II

- COCOMO II is a 3-level model that allows increasingly detailed estimates to be prepared as development progresses
  - Early prototyping level
    - Estimates based on object points and a simple formula is used for effort estimation
  - Early design level
    - Estimates based on function points that are then translated to LOC
    - Includes 7 cost drivers
  - Post-architecture level
    - Estimates based on lines of source code **or** function point
    - Includes 17 cost drivers
- Five scale factors replace COCOMO 81 ratings (organic, semi-detached, and embedded)

Early prototyping level - COCOMO II

- Suitable for projects built using modern GUI-builder tools
  - Based on Object Points
- Supports prototyping projects and projects where there is extensive reuse
- Based on standard estimates of developer productivity in object points/month
- Takes CASE tool use into account
- Formula is
  - $PM = (NOP \times (1 - \%reuse/100)) / PROD$
  - PM is the effort in person-months, NOP is the number of object points and PROD is the productivity

Early Design Level: 7 cost drivers - COCOMO II

- Estimates can be made after the requirements have been agreed
- Based on standard formula for algorithmic models

Effort for Manually developed code

$$PM = \underbrace{A \times Size^{B} \times M}_{} + PM_m$$

Effort for Manual adaptation of Automatically generated code

- $M = PERS \times RCPX \times RUSE \times PDIF \times PREX \times FCIL \times SCED$
  - A = 2.5 in initial calibration,
  - Size: manually developed code in **KLOC**
  - Exponent **B**
    - varies from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity.
    - B is calculated using a Scale Factor based on 5 exponent drivers
  - PM_m: represents manual adaptation for automatically generated code

PM_m : Manual Adaptation for Automatically Generated Code ..

$$PM_m = (ASLOC \times (AT/100)) / ATPROD$$

- Used when big % of code is generated automatically
- ASLOC :Size of adapted components
- ATPROD: Productivity of the engineer integrating the adapted code (app. 2400 source statements per month)
- AT: % of adapted code (that is automatically generated)

COCOMO II Early Design Stage Effort Multipliers: 7 cost drivers

- Multipliers reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.
  - 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 Exponent B
Scale Factor(SF) - COCOMO II

- Exponent **B** for effort calculation
- B = 1.01 + 0.01 x sum [SF (i)] , i=1,…, 5
  - SF = Scale Factor
- Each SF is rated on 6-point scale (ranging from 0 to 5) :
  - very low (5), low (4), nominal (3), high (2), very high (1), extra high (0)
- 5 Scale Factor (exponent drivers)
  - Precedenteness
  - Development flexibility
  - Architecture/risk resolution
  - Team cohesion
  - Process maturity

  - Ex: 20 KLOC ^ 1.26 / 20 KLOC ^ 1.01 = 43.58/20.6 = 2.11

Exponent scale factors - COCOMO II

Scale factor	Explanation
Precedentedness	Reflects the previous experience of the organisation with this type of project. Very low means no previous experience, Extra high means that the organisation is completely familiar with this application domain.
Development flexibility	Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; Extra high means that the client only sets general goals.
Architecture/risk resolution	Reflects the extent of risk analysis carried out. Very low means little analysis, Extra high means a complete a thorough risk analysis.
Team cohesion	Reflects how well the development team know each other and work together. Very low means very difficult interactions, Extra high means an integrated and effective team with no communication problems.
Process maturity	Reflects the process maturity of the organisation. The computation of this value depends on the CMM Maturity Questionnaire but an estimate can be achieved by subtracting the CMM process maturity level from 5.

Example: Exponent B calculations using Scale Factor

Given:

- Precedenteness - new project – rated low SF(1) = 4
- Development flexibility - no client involvement – rated Very high - SF(2) = 1
- Architecture/risk resolution - No risk analysis – rated Very Low - SF(3) = 5
- Team cohesion - new team - nominal - SF(4) = 3
- Process maturity - some control - nominal - SF(5) = 3

Then:

- Exponent B =1.17

Post-architecture stage - COCOMO II

- Uses same formula as early design estimates
- Estimate of size is adjusted to take into account
  - Requirements volatility: Rework required to support change
  - Extent of possible reuse: Reuse is non-linear and has associated costs so this is not a simple reduction in LOC
  - $ESLOC = ASLOC \times (AA + SU + 0.4DM + 0.3CM + 0.3IM)/100$
    - ESLOC is equivalent number of lines of new code. ASLOC is the number of lines of reusable code which must be modified, DM is the percentage of design modified, CM is the percentage of the code that is modified , IM is the percentage of the original integration effort required for integrating the reused software.
    - SU is a factor based on the cost of software understanding, AA is a factor which reflects the initial assessment costs of deciding if software may be reused.

COCOMO II Post Architecture Effort Multipliers (17 multipliers)

- Product attributes (5 multipliers)
  - concerned with required characteristics of the software product being developed
- Computer attributes (3 multipliers)
  - constraints imposed on the software by the hardware platform
- Personnel attributes (6 multipliers)
  - multipliers that take the experience and capabilities of the people working on the project into account.
- Project attributes (3 multipliers)
  - concerned with the particular characteristics of the software development project

COCOMO II Post Architecture Effort Multipliers:17 cost drivers

Product attributes			
RELY	Required system reliability	DATA	Size of database used
CPLX	Complexity of system modules	RUSE	Required percentage of reusable components
DOCU	Extent of documentation required		
Computer attributes			
TIME	Execution time constraints	STOR	Memory constraints
PVOL	Volatility of development platform		
Personnel attributes			
ACAP	Capability of project analysts	PCAP	Programmer capability
PCON	Personnel continuity	AEXP	Analyst experience in project domain
PEXP	Programmer experience in project domain	LTEX	Language and tool experience
Project attributes			
TOOL	Use of software tools	SITE	Extent of multi-site working and quality of site communications
SCED	Development schedule compression		

Effects of cost drivers

Maximum & Minimum Data are from ref: Boehm, 1997

Exponent value System size (including factors for reuse and requirements volatility) **Initial COCOMO estimate without cost drivers (M=1)**	1.17 128, 000 DSI **730 person-months**
Reliability Complexity Memory constraint Tool use Schedule **Adjusted COCOMO estimate:**	Very high, multiplier = 1.39 Very high, multiplier = 1.3 High, multiplier = 1.21 Low, multiplier = 1.12 Accelerated, multiplier = 1.29 **2306 person-months**
Reliability Complexity Memory constraint Tool use Schedule **Adjusted COCOMO estimate:**	Very low, multiplier = 0.75 Very low, multiplier = 0.75 None, multiplier = 1 Very high, multiplier = 0.72 Normal, multiplier = 1 **295 person-months**

Maximum

Minimum

Effects of cost drivers (M = ?)

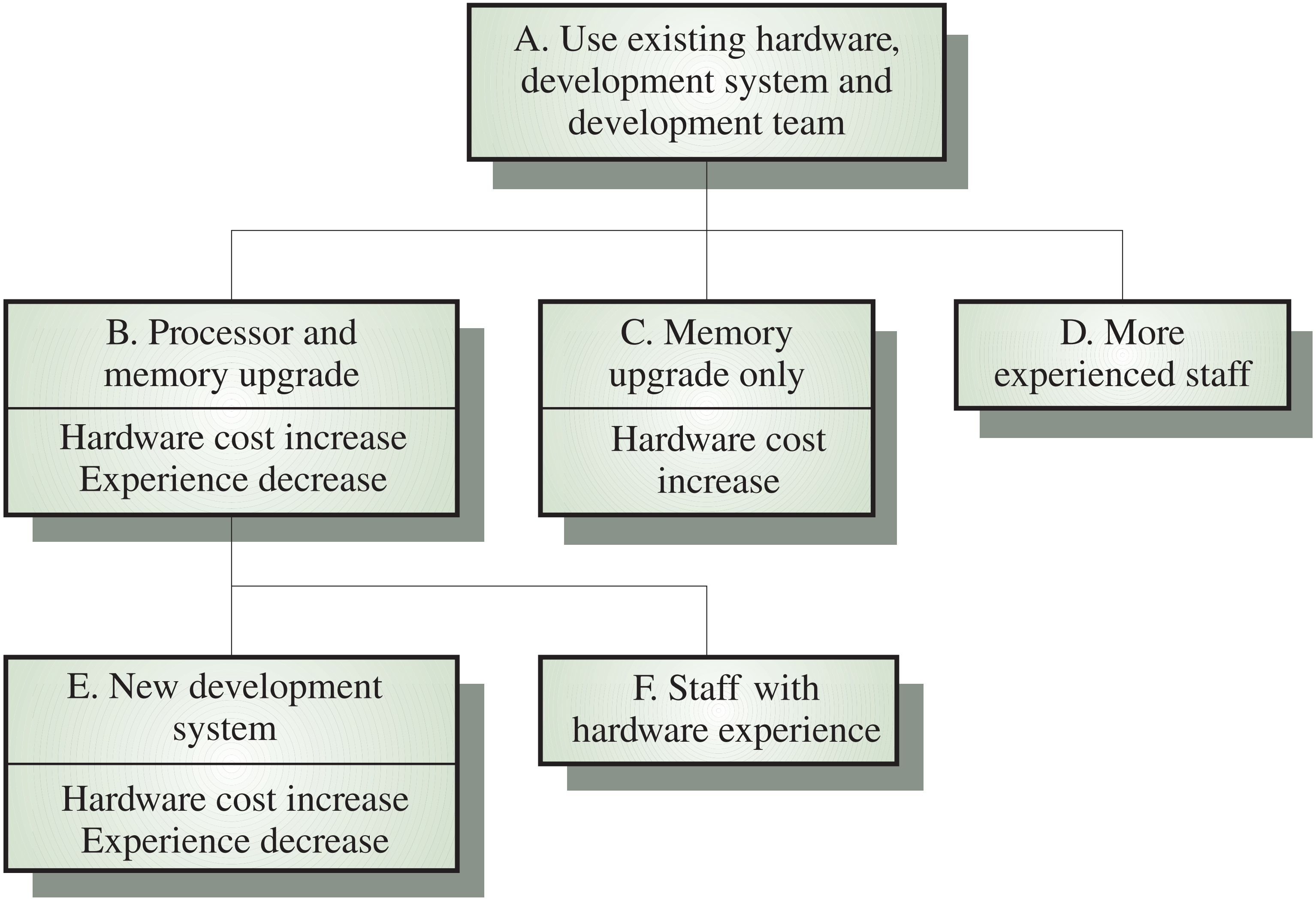
Maximum & Minimum Data are from ref: Boehm, 1997

Exponent value System size (including factors for reuse and requirements volatility) **Initial COCOMO estimate without cost drivers (M=1)**	1.17 128, 000 DSI **730 person-months**
Reliability Complexity Memory constraint Tool use Schedule **Adjusted COCOMO estimate:** **$M = \prod (M_i) = 3.15$**	Very high, multiplier = 1.39 Very high, multiplier = 1.3 High, multiplier = 1.21 Low, multiplier = 1.12 Accelerated, multiplier = 1.29 **2306 person-months**
Reliability Complexity Memory constraint Tool use Schedule **Adjusted COCOMO estimate:** **$M = \prod (M_i) = 0.405$**	Very low, multiplier = 0.75 Very low, multiplier = 0.75 None, multiplier = 1 Very high, multiplier = 0.72 Normal, multiplier = 1 **295 person-months**

Project planning

- Algorithmic cost models provide a basis for project planning as they allow alternative strategies to be compared
- Embedded spacecraft system
  - Must be reliable
  - Must minimise weight (number of chips)
  - Multipliers on reliability and computer constraints > 1
- Cost components
  - Target hardware
  - Development platform
  - Effort required

Management options



Management options costs

Option	RELY	STOR	TIME	TOOLS	LTEX	Total effort	Software cost	Hardware cost	Total cost
A	1.39	1.06	1.11	0.86	1	63	949393	100000	1049393
B	1.39	1	1	1.12	1.22	88	1313550	120000	1402025
C	1.39	1	1.11	0.86	1	60	895653	105000	1000653
D	*1.39*	*1.06*	*1.11*	*0.86*	*0.84*	*51*	*769008*	*100000*	*897490*
E	1.39	1	1	0.72	1.22	56	844425	220000	1044159
F	1.39	1	1	1.12	0.84	57	851180	120000	1002706

Option choice

- Option D (use more experienced staff) appears to be the best alternative
  - However, it has a high associated risk as experienced staff may be difficult to find
- Option C (upgrade memory) has a lower cost saving but very low risk
- Overall, the model reveals the importance of staff experience in software development

Project duration and staffing - COCOMO II

- As well as effort estimation, managers must estimate the calendar time required to complete a project and when staff will be required
- Calendar time can be estimated using a COCOMO II formula
  - $TDEV = 3 \times (PM)^{(0.33+0.2*(B-1.01))}$
  - PM is the effort computation and B is the exponent computed as discussed above (B is 1 for the early prototyping model). This computation predicts the nominal schedule for the project
- The time required is independent of the number of people working on the project

Project duration and staffing Example

- Given:
  - Software development effort = 60 PM
  - Exponent B = 1.17

- Then:
  - Nominal schedule for the project (calendar time TDEV required to complete the project):

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

$$= 3 \times (PM)^{(0.36)}$$

$$= 13 \text{ months}$$

Staffing requirements

- Staff required can't be computed by diving the development time by the required schedule – Non linear relation ship
- The number of people working on a project varies depending on the phase of the project
- The more people who work on the project, the more total effort is usually required
- A very rapid build-up of people often correlates with schedule slippage

Use Case Points UCP

- Effort: person-month based on Use Case description.

- See file: Use_Case_Points.doc