Lecture15-16: Software Cost Estimation

EGCI341: WEEK10

Fundamental Estimation Questions

- How much effort is required to complete an activity?
- How much calendar time is needed to complete an activity?
- What is the total cost of an activity?

Software Cost Components

- Hardware and software costs
- Travel and training costs
- Effort costs (the dominant factor in most projects)
  - The salaries of engineers involved in the project
  - Social and insurance costs
- Effort costs must take overheads into account
  - Costs of building, air condition, lighting
  - Costs of networking and communications
  - Costs of shared facilities (e.g library, staff restaurant, etc.)

Software Pricing 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 organisation the opportunity to make a greater profit later. The experience gained may also help it develop new products.
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 can 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 smaller than normal profit or break even than to go out of business.

Software Productivity

- A measurement of the rate at which individual engineers involved in software development produce software and associated documentation
- Not quality-oriented although quality assurance is a factor in productivity assessment
- Measure the functionality produced per time unit

Productivity Measures

- Size-related measures based on output from the software process
  - This may be lines of delivered source code, object code instructions, etc.
- Function-related measures based on an estimation of the functionality of the delivered software
  - Function-points are the best known of this type of measure

Measurement Problems

- Estimating the size of the measurement
  - e.g. How many function points
- Estimating the total number of programmer's months that have elapsed
- Estimating contractor productivity (e.g. documentation team)

Lines of Code (LOC)

- What's a line of code?
  - The measurement was the first propose when programs were typed on cards with one line per card
  - How does this correspondent 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?
- This model assumes that there is a linear relationship between system size and volume of documentation

Productivity Comparisons

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

- The more verbose the programmer, the higher the productivity
  - Measures of productivity based on lines of codes suggest that programmers who write verbose code are more productive than programmers who write compact code

System Development Times

	Analysis	Design	Coding	Testing	Documentation
Assembly code	3 weeks	5 weeks	8 weeks	10 weeks	2 weeks
High-level language	3 weeks	5 weeks	4 weeks	6 weeks	2 weeks

	Size	Effort	Productivity
Assembly code	5000 lines	28 weeks	714 lines/month
High-level language	1500 lines	20 weeks	300 lines/month

Function Points

Based on a combination of program characteristics

1. External Inputs and Outputs (EI, EO)
2. User Interactions or External Queries (EQ)
3. External Interface files (EIF)
4. files used by the system or Internal Logical Files (ILF)

A weight is associated with each of these characteristics and the function point count is computed by multiplying each raw count by the weight and summing all values

- Three types of No. of elements : data element (DET), record element (RET), and file type reference (FTR)
- Weight or value adjustment factor (VAF)

$$\text{UFC} = \sum(\text{number of elements of given type}) \times (\text{weight})$$

Function Points (Cont.)

- Function point count is modified by complexity of the project
- FPs can be used to estimate LOC depending on the average number of LOC per Function Points (FP) for a given language
  - LOC = AVC * number of function points
  - AVC is a language-dependent factor varying from 200-300 for assemble language to 2-40 for a 4GL
  - AVC: the average number of lines of code
- FPs are very subjective
  - They depend on the estimator
  - Automatic function-point counting is impossible

Function Points (Cont.)

Complexity Level

- Low
- Average
- High

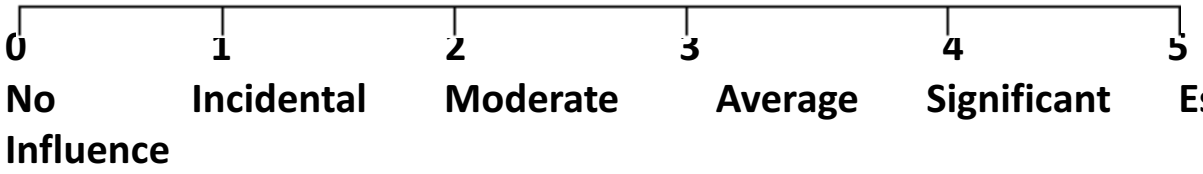
FP Complexity Level			
Internal Logical Files and External Interface Files			
Record Elements	**Data Element**		
	1-19	**20-50**	**51+**
1	Low	Low	Average
2-5	Low	Average	High
6+	Average	High	High
Internal External Output and External Inquiry			
File Types	**Data Element**		
	1-5	**6-19**	**20+**
0 or 1	Low	Low	Average
2-3	Low	Average	High
4+	Average	High	High
For External Input			
File Types	**Data Elements**		
	1-4	**5-15**	**16+**
0 or 1	Low	Low	Average
2-3	Low	Average	High
3+	Average	High	High

Function Point Calculation

Complexity-Weight					
Function Type	**Count**	**Weight-Factor**			**Total**
		Low	**Average**	**High**	
1. Internal Logical File (ILF)	___ x	7	10	15	
2. External Interface File (EIF)	___ x	5	7	10	
3. External Input (EI)	___ x	3	4	6	
4. External Output (EO)	___ x	4	5	7	
5. External Inquiry (EQ)	___ x	3	4	6	
UFC				**Sum Total =**	
Complexity Multiplier (VAF)				**VAF = 0.65+(0.01xTDI) =**	
Function Points =				**FP = UFCxVAF =**	

Degree of Influence

To calculate DI, the following factors are rated on a scale of 0 to 5:



S/N	Question	Rating
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	Will the system run in an existing heavily utilized operational environment?	
6	Does the system require online data entry?	
7	Does the online data entry require the input transaction to be built over multiple screens or operations?	
8	Are the master files updated online?	
9	Are the inputs, outputs, files, or inquires complex?	
10	Is the internal processing complex?	
11	Is the code designed to be reusable?	
12	Are conversion and installation included in the design?	
13	Is the system designed for multiple installations in different organizations?	
14	Is the application designed to facilitate change and ease of use by the user?	
	Total DI (TDI) =	Σ

Average Number of Lines of Code (AVC)

Lines of Codes: LOC = AVC * FP

Language	SLOC/FP
Access	38
C	128
C++	55
HTML	15
JAVA	53
Perl	27
Visual C++	34

(From Capers Jones, 1996)

Example: Lines of Code

There are 4 subsystems in this project

Given for each subsystem:

- size in LOC
- productivity metric: LOC/P-month
- Cost metric: $/LOC

Calculate the system total cost in $ and effort in months

Functions	Estimated LOC	LOC/P-month	$/LOC	Cost	Effort
UICF	2340	315	14	**32760**	**7.4**
2DGA	5380	220	20		
DSM	3350	240	18		
PCF	2140	140	28		
Total					

Object Points

- Object points (alternatively named application points) are an alternative function-related measurement to function points
- Object points are NOT the same as object classes
- The number of object points in a program is a weighted estimation of:
  - Number of separate screens that are displayed
  - Number of reports that are produced by the system
  - Number of program modules that must be developed to supplement the database code

Productivity Estimates

- Real-time embedded systems, 40-160 LOC/P-month
  - (**LOC/P-month**: Lines of code per person-month)
- System programs , 150-400 LOC/P-month
- Commercial applications, 200-900 LOC/P-month
- Object points:
  - Productivity has been measured between 4 and 50 object points/month
  - Depend on tool's supports and developer capability

Factors Affecting Productivity

- Application Domain Experience
- Process Quality
- Project Size
- Technology Support
- Working Environment

Estimation Techniques

Technique	Description
Algorithmic cost modelling	A model is developed using historical cost information that relates some software metric (usually its size) to the project cost. An estimate is made of that metric and the model predicts the effort required.
Expert judgement	Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached.
Estimation by analogy	This technique is applicable when other projects in the same application domain have been completed. The cost of a new project is estimated by analogy with these completed projects. Myers (Myers, 1989) gives a very clear description of this approach.
Parkinson's Law	Parkinson's Law states that work expands to fill the time available. The cost is determined by available resources rather than by objective assessment. If the software has to be delivered in 12 months and 5 people are available, the effort required is estimated to be 60 person-months.
Pricing to win	The software cost is estimated to be whatever the customer has available to spend on the project. The estimated effort depends on the customer's budget and not on the software functionality.

Pricing to Win

- Project costs whatever the customer has to spend on it
- Advantages:
  - You get the contract
- Disadvantages:
  - The probability that the customer gets the system he or she wants is small
  - Costs do not accurately reflect the work required

Top-Down and Bottom-Up Estimation

- Top-down
  - Start at the system level and assess overall system functionality and how this is delivered through subsystems
- Bottom-up
  - Start at the component level and estimate the effort required for each component
  - Add these efforts to reach a final estimation

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
- It may 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
- This can be an accurate method if the system has been designed in detail
- It may underestimate the costs of system level activities such as integration and documentation

Algorithmic Cost Modelling

- 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 is an organisation-dependent constant,
  - B reflects the disproportionate effort for large projects and,
  - M is a multiplier reflecting product, process and people attributes
- Most commonly used product attribute for cost estimation is code size
- Most models are similar but they use different values for A, B and M

Estimation Accuracy

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

COCOMO Model

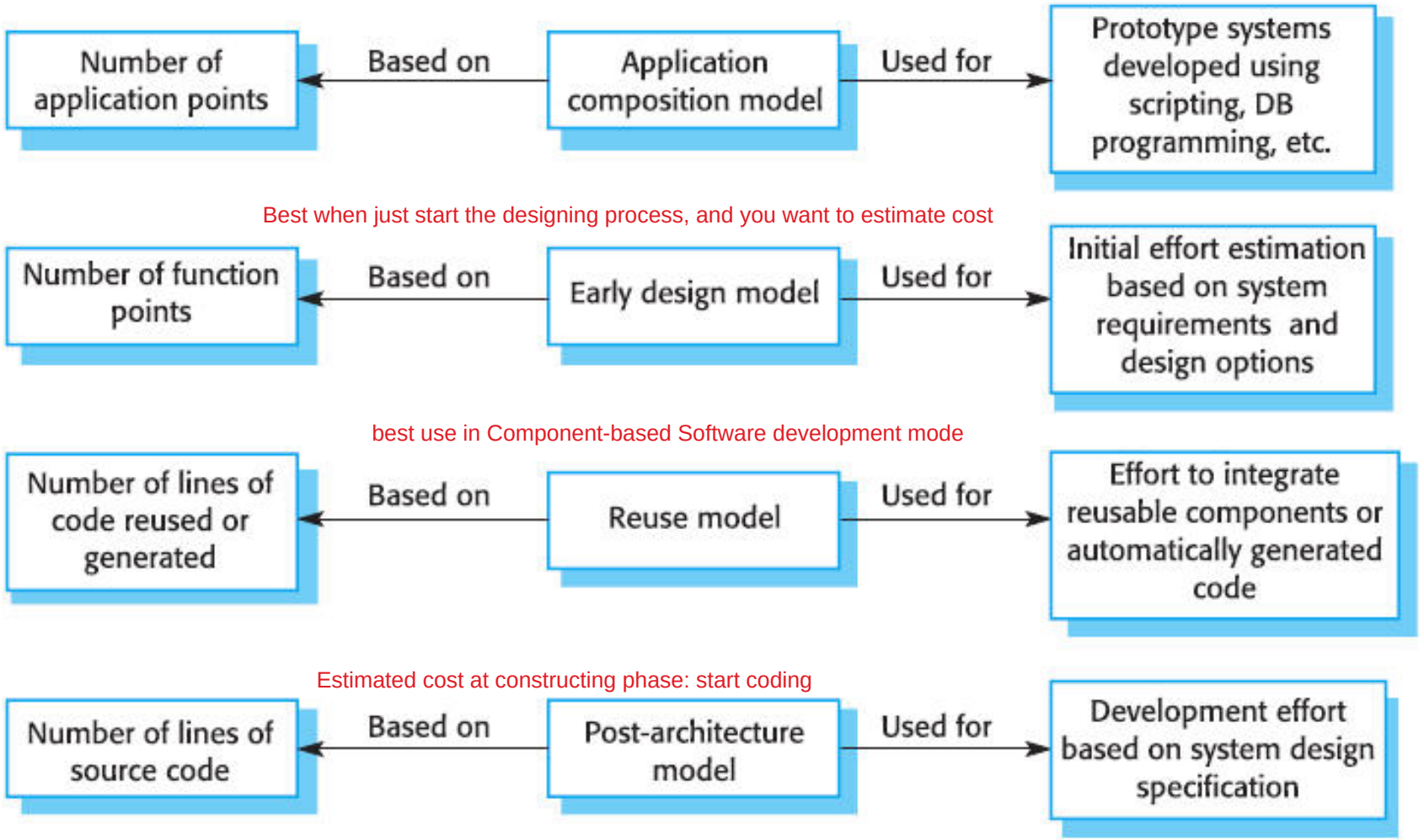
- Constructive Cost Model (COCOMO)
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2
- COCOMO 81 was developed with the assumption that a waterfall process would be used and that all software would be developed from scratch
- There have been many changes in software engineering practice
  - COCOMO 2 is designed to accommodate different approaches to software development

COCOMO 2 Models

Submodels in COCOMO 2 are:

1. Application composition model
   - Used when software is composed from existing parts
2. Early design model
   - Used when requirements are available but design has not yet started
3. Reuse model
   - Used to compute the effort of integrating reusable components
4. Post-architecture model
   - Used once the system architecture has been designed and more information about the system is available

COCOMO 2 Models (Cont.)



Application Composition Model

- Supports prototyping projects and projects where there is extensive reuse
- Based on standard estimates of developer productivity in application (object) points/month
- Takes CASE tool use into account
- Formula is: PM = (NAP x (1 - %reuse/100)) / PROD
  - **PM** is the effort in person-months
  - **NAP** or **NOP** is the number of application points or object points
  - **PROD** is the productivity

PM = 8 * (0.8)/7

Object Point: Complexity Level and Weighting

For Screen				For Report			
No. of Views	# and source of data tables			Number of Section	# and source of data tables		
	Total <4 (<2 servers, <3 clients)	Total<8 (2-3 servers, 3-5 clients)	Total 8+ (>3 servers, >5 clients)		Total<4 (<2 server, 3 clients)	Total<8 (2-3 servers, 3-5 clients)	Total 8+ (>3 servers, >5 clients)
<3	simple	simple	Medium	**0-1**	simple	simple	Medium
3-7	simple	medium	difficult	**2-3**	simple	medium	difficult
8+	medium	difficult	difficult	**4+**	medium	difficult	difficult

Object Type	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
Each 3GL Module	10	10	10

Example: Object Counts

Object counts:

2 simple screens x 1	= 2
3 medium screens x 2	= 6
Total Objects	**= 8**

Object Point Productivity

Developer's experience and capability	Very low	Low	Nominal	High	Very high
CASE maturity and capability	Very low	Low	Nominal	High	Very high
PROD (NOP/month)	4	7	13	25	50

PM = 2.9 * 150^1.16 * 24 = 23274.22

B = 1.26

Early Design Model

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

  PM = A × Size^{B} × M where
  - **M** = PERS × RCPX × RUSE × PDIF × PREX × FCIL × SCED;
  - **A** = 2.94 in initial calibration,
  - **Size** in KLOC
  - **B** varies from 1.1 to 1.24
    - Depending on novelty of the project, development flexibility, risk management approaches and the process maturity

A is constant but it can change from team to team from CMMI test, reflect performance of the team
size = 1000 line of codes
B = exponent number

Multipliers Multiply all of this to get **M**

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

You estimate values for these attributes using a six-point scale where 1 corresponds to very low values and 6 corresponds to very high

The Exponent B : Scale Factor(SF)

This how we calculate B for early design

Exponent B for effort calculation

$$B = 1.01 + 0.01 \times \sum SF(i) \quad , i=1,\ldots, 5$$

- 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)

Five Scale Factor (exponent drivers)

1. Precedenteness
2. Development flexibility
3. Architecture/risk resolution
4. Team cohesion
5. Process maturity

Exponent Scale Factors

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 sets only general goals.
Architecture/risk resolution	Reflects the extent of risk analysis carried out. Very low means little analysis; Extra high means a complete and 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: The Exponent Term

- This depends on 5 scale factors (see next slide). Their sum/100 is added to 1.01
- A company takes on a project in a new domain. The client has not defined the process to be used and has not allowed time for risk analysis.

  The company has a CMM level 2 rating:

  - Precedence - new project (4)
  - Development flexibility - no client involvement - Very high (1)
  - Architecture/risk resolution - No risk analysis - V. Low (5)
  - Team cohesion - new team - nominal (3)
  - Process maturity - some control - nominal (3)

B value

Scale factor is therefore 1.17

Reuse Model

- Takes into account black-box code that is reused without change and code that has to be adapted to integrate it with new code
- There are two versions:
  - Black-box reuse where code is not modified
    - Effort estimation (PM) is computed
  - White-box reuse where code is modified
    - A size estimation equivalent to the number of lines of new source code is computed
    - This adjusts the size estimation for new code

Reuse Model Estimation

For generated code:

PM = (ASLOC * AT/100)/ATPROD

- ASLOC is the number of lines of generated code
- AT is the percentage of code automatically generated
- ATPROD is the productivity of engineers in integrating this code

For Generated code example
PM = (20,000 x 30/100) / 2400 = 2.5 person-months

Reuse Model Estimates 2

When code has to be understood and integrated:

ESLOC = ASLOC * (1-AT/100) * AAM

- ASLOC is the number of lines of generated code
- AT is the percentage of code automatically generated
- AAM is the adaptation adjustment multiplier computed from the costs of changing the reused code, the costs of understanding how to integrate the code and the costs of reuse decision making.
- AAM is the sum of three components (AAF+SU+AA)

Reuse Model Estimation (Cont.)

AAM is the sum of three components

- AAF (Adaptive Component) 0.4*DM + 0.3IM + 0.3*CM
- SU (Understanding Component) ranges from 50 for complex to 10 for well-written, object-oriented code
- AA (Assessment Component) varies from 0 to 8 depending on the amount of analysis effort required

Post-architecture Level

- Uses the same formula as the early design model but with 17 rather than 7 associated multipliers
- The code size is estimated as:
  - Number of lines of new code to be developed;
  - Estimation of equivalent number of lines of new code computed using the reuse model;
  - An estimation of the number of lines of code that have to be modified according to requirements changes

Multipliers

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

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

Base line

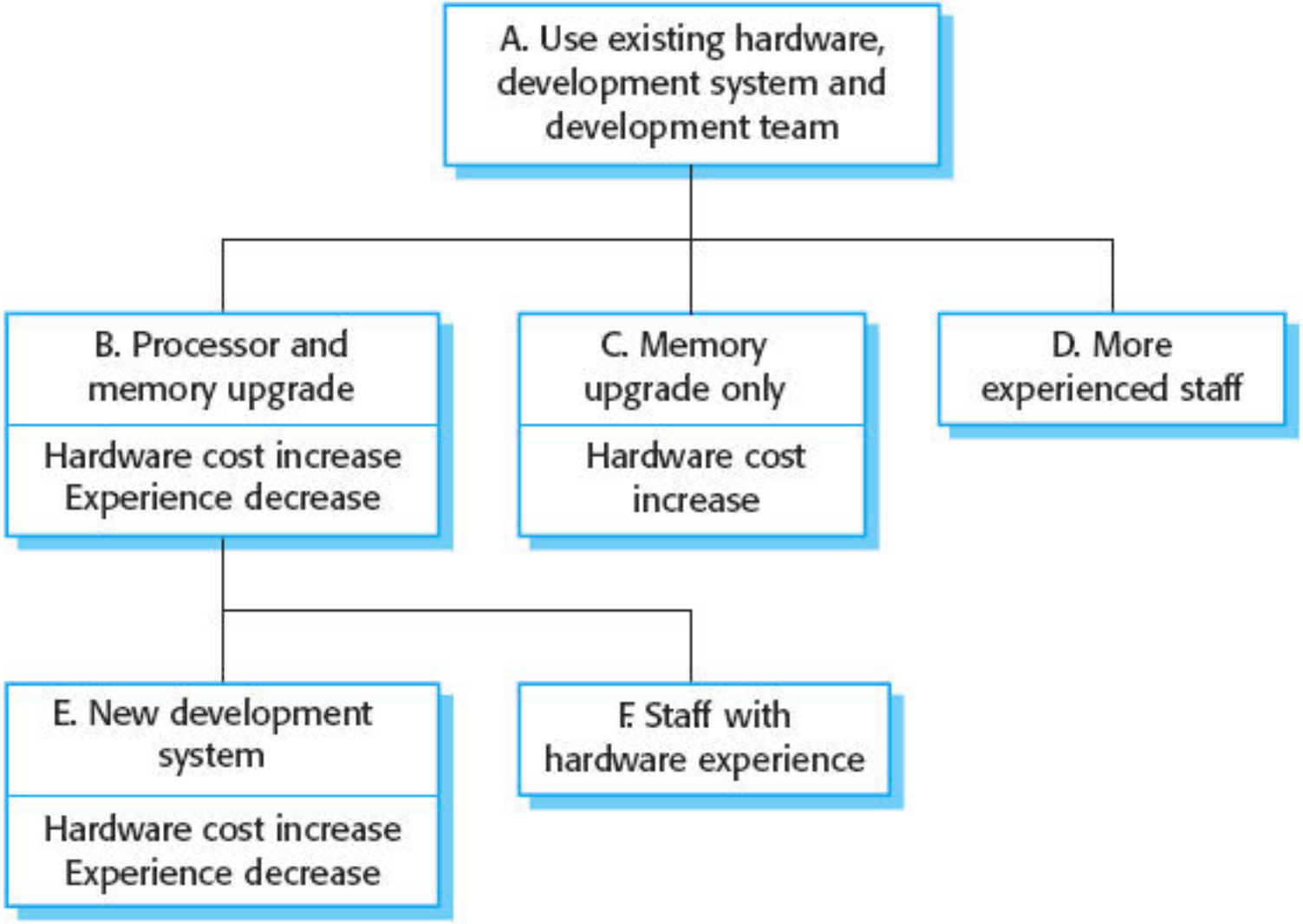
Exponent value	1.17
System size (including factors for reuse and requirements volatility)	128,000 DSI
Initial COCOMO estimate without cost drivers	**730 person-months**
Reliability	Very high, multiplier = 1.39
Complexity	Very high, multiplier = 1.3
Memory constraint	High, multiplier = 1.21
Tool use	Low,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

Lower cost

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
  - Development effort

Management Options



Management Option 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**
EX	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

p-month != real month, this is not take into consideration about number of people working, and productivity of them

Project Duration and Staff

- As well as effort estimation, m anagers must estimate the calendar time required to complete a project and when staff will be required
- Calendar time can be estimated using a COCOMO 2 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

use this early design model and post architecture model

Staffing Requirements

- Staff requirement cannot be computed by dividing the development time by the required schedule
- 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

Reference

- This set of slides and examples are modified from Ian Sommerville, Software Engineering 8th Edition, Addison-Wesley; 2007

Any Questions?

:O)

Thank you