Software cost estimation

Objectives

- To introduce the fundamentals of software costing and pricing

- To describe three metrics for software productivity assessment

- To explain why different techniques should be used for software estimation

- To describe the principles of the COCOMO 2 algorithmic cost estimation model

Topics covered

- Software productivity

- Estimation techniques

- Algorithmic cost modelling

- Project duration and staffing

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?

- Project estimation and scheduling are interleaved management activities.

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, heating, lighting.

  - Costs of networking and communications.

  - Costs of shared facilities (e.g library, staff restaurant, etc.).

Costing and pricing

- Estimates are made to discover the cost, to the developer, of producing a software system.

- There is not a simple relationship between the development cost and the price charged to the customer.

- Broader organisational, economic, political and business considerations influence the price charged.

Software pricing factors

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 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 measure 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.

- Essentially, we want to measure useful functionality produced per time unit.

Productivity measures

- Size related measures based on some output from the software process. This may be ***lines of delivered source code***, object code instructions, etc.

- 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.

Measurement problems

- Estimating the size of the measure (e.g. how many function points).

- Estimating the total number of programmer months that have elapsed.

- Estimating contractor productivity (e.g. documentation team) and incorporating this estimate in overall estimate.

Lines of code

- 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?

- 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 code to implement in a lower-level language than in a high-level language.

- The more verbose (wordy) the programmer, the higher the productivity

  - Measures of productivity based on lines of code 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	10weeks	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

  - external inputs and outputs;

  - user interactions;

  - external interfaces;

  - files used by the system.

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

- UFC: unadjusted function-point count

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

Function points

- The 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 FP for a given language

  - LOC = AVC * number of function points;
  - AVC (avg. # of LOC) is a language-dependent factor varying from 200-300 for assemble language to 2-40 for a 4GL;

- FPs are very subjective. They depend on the estimator

  - Automatic function-point counting is impossible.

Object points

- Object points (alternatively named application 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 program modules that must be developed to supplement the database code;

Object point estimation

- Object points are easier to estimate from a specification than function points as they are simply concerned with screens, reports and programming language modules.

- They can therefore be estimated at a fairly early point in the development process.

- At this stage, it is very difficult to estimate the number of lines of code in a system.

Productivity estimates

- Real-time embedded systems, 40-160 LOC/P-month.

- Systems programs , 150-400 LOC/P-month.

- Commercial applications, 200-900 LOC/P-month.

- In object points, productivity has been measured between 4 and 50 object points/month depending on tool support and developer capability.

Factors affecting productivity

Application domain experience	Knowledge of the application domain is essential for effective software development. Engineers who already understand a domain are likely to be the most productive.
Process quality	The development process used can have a significant effect on productivity..
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, configuration management systems, etc. can improve productivity.
Working environment	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.

- It is not clear how productivity/quality metrics are related.

- If requirements are constantly changing then an approach based on counting lines of code is not meaningful as the program itself is not static;

Estimation techniques

- There is no simple way to make an accurate estimate of the effort required to develop a software system

  - Initial estimates are 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.

Changing technologies

- Changing technologies may mean that previous estimating experience does not carry over to new systems
    - Distributed object systems rather than mainframe systems;
    - Use of web services;
    - Use of ERP (enterprise resource planning) or database-centred systems;
    - Use of off-the-shelf software;
    - Development for and with reuse;
    - Development using scripting languages;
    - The use of CASE tools and program generators.

Estimation techniques

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

Estimation techniques

Algorithmic cost modelling	A model based on historical cost information that relates some software metric (usually its size) to the project cost is used. 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

- The 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

- Any of these approaches may be used top-down or bottom-up.

- Top-down
  - Start at the system level and assess the overall system functionality and how this is delivered through sub-systems.

- 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.

- 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.

Estimation methods

- Each method has strengths and weaknesses.

- Estimation should be based on several methods.

- If these do not return approximately the same result, then you have insufficient information available to make an estimate.

- 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.

Pricing to win

- This approach may seem unethical and un-businesslike.

- 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.

Algorithmic cost modelling

- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers:

  - Effort $= A \times Size^B \times M$

  - 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.

- The 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

- 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
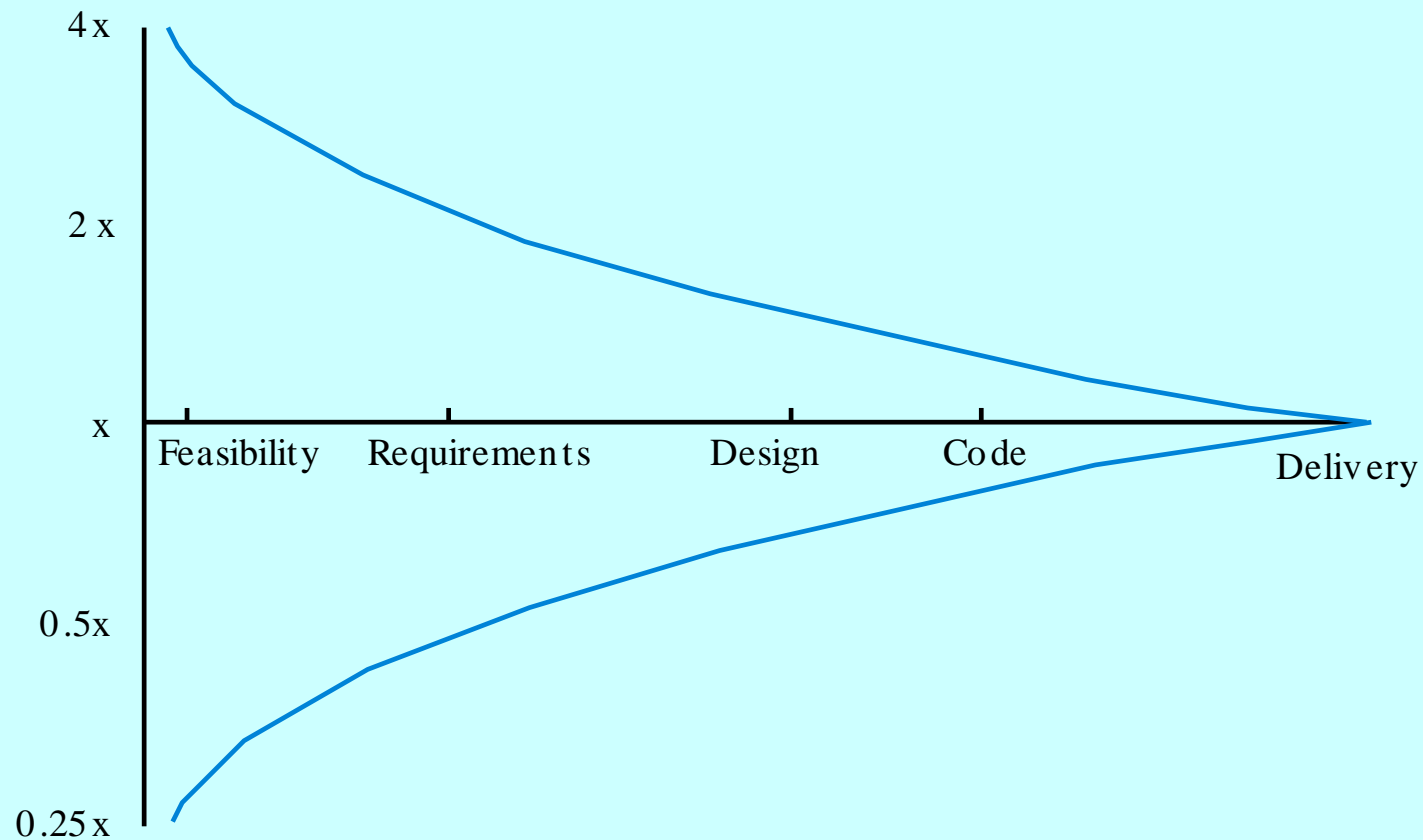
The COCOMO model

- An empirical model based on project experience.

- Well-documented, 'independent' model which is not tied to a specific software vendor.

- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.

- COCOMO 2 takes into account different approaches to software development, reuse, etc.

COCOMO 81

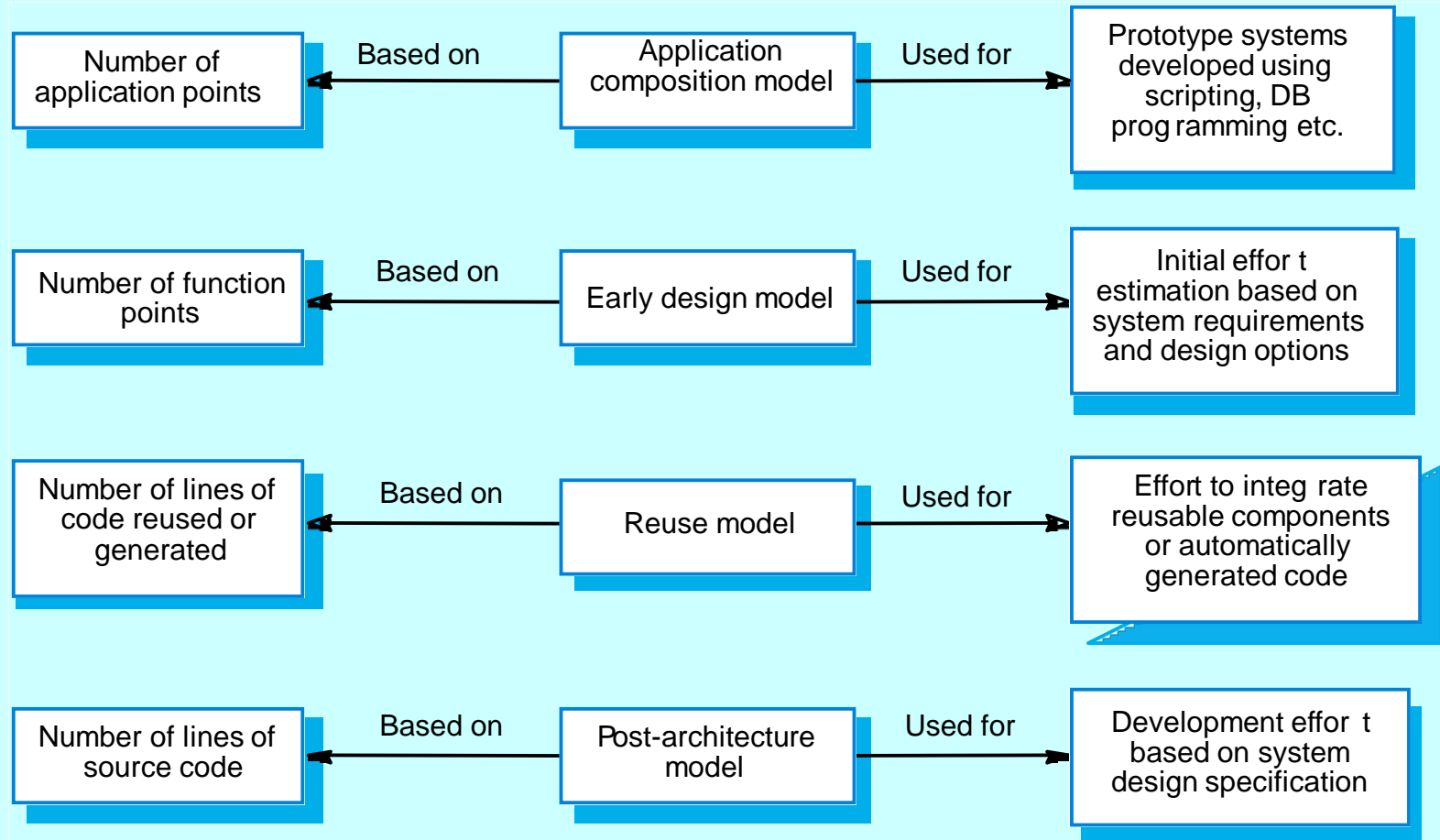| Project complexity	Formula	Description
Simple	$PM = 2.4 \, (KDSI)^{1.05} \times M$	Well-understood applications developed by small teams.
Moderate	$PM = 3.0 \, (KDSI)^{1.12} \times M$	More complex projects where team members may have limited experience of related systems.
Embedded	$PM = 3.6 \, (KDSI)^{1.20}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

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.

- Since its formulation, there have been many changes in software engineering practice and COCOMO 2 is designed to accommodate different approaches to software development.

COCOMO 2 models

- COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.

- The sub-models in COCOMO 2 are:

    - Application composition model. Used when software is composed from existing parts.

    - Early design model. Used when requirements are available but design has not yet started.

    - Reuse model. Used to compute the effort of integrating reusable components.

    - Post-architecture model. Used once the system architecture has been designed and more information about the system is available.

Use of COCOMO 2 models

Number of application points	← Based on	Application composition model	Used for →	Prototype systems developed using scripting, DB programming etc.
Number of function points	← Based on	Early design model	Used for →	Initial effort estimation based on system requirements and design options
Number of lines of code reused or generated	← Based on	Reuse model	Used for →	Effort to integrate reusable components or automatically generated code
Number of lines of source code	← Based on	Post-architecture model	Used for →	Development effort based on system design specification

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 $\times$ (1 - %reuse/100)) / PROD

  - PM is the effort in person-months, NAP is the number of application points and PROD is the productivity.

Object point productivity

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

Early design model

- Estimates can be made after the requirements have been agreed.

- Based on a standard formula for algorithmic models

    - $PM = A \times Size^B \times M$ where

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

    - 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.

Multipliers

- Multipliers reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.

  - RELY - 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

- 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. An effort estimate (PM) is computed.

    - White-box reuse where code is modified. A size estimate equivalent to the number of lines of new source code is computed. This then adjusts the size estimate for new code.

Reuse model estimates 1

- 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.

Reuse model estimates 2

- When code has to be understood and integrated:

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

    - ASLOC and AT as before.

    - 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.

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;

  - Estimate of equivalent number of lines of new code computed using the reuse model;

  - An estimate of the number of lines of code that have to be modified according to requirements changes.

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.

  - Precedentedness - 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)

- Scale factor is therefore 1.17.

Exponent scale factors

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.

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.

Effects of cost drivers

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**

Project planning

- Algorithmic cost models provide a basis for project planning as they allow alternative strategies to be compared.

- Embedded spacecraft system – a RT 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
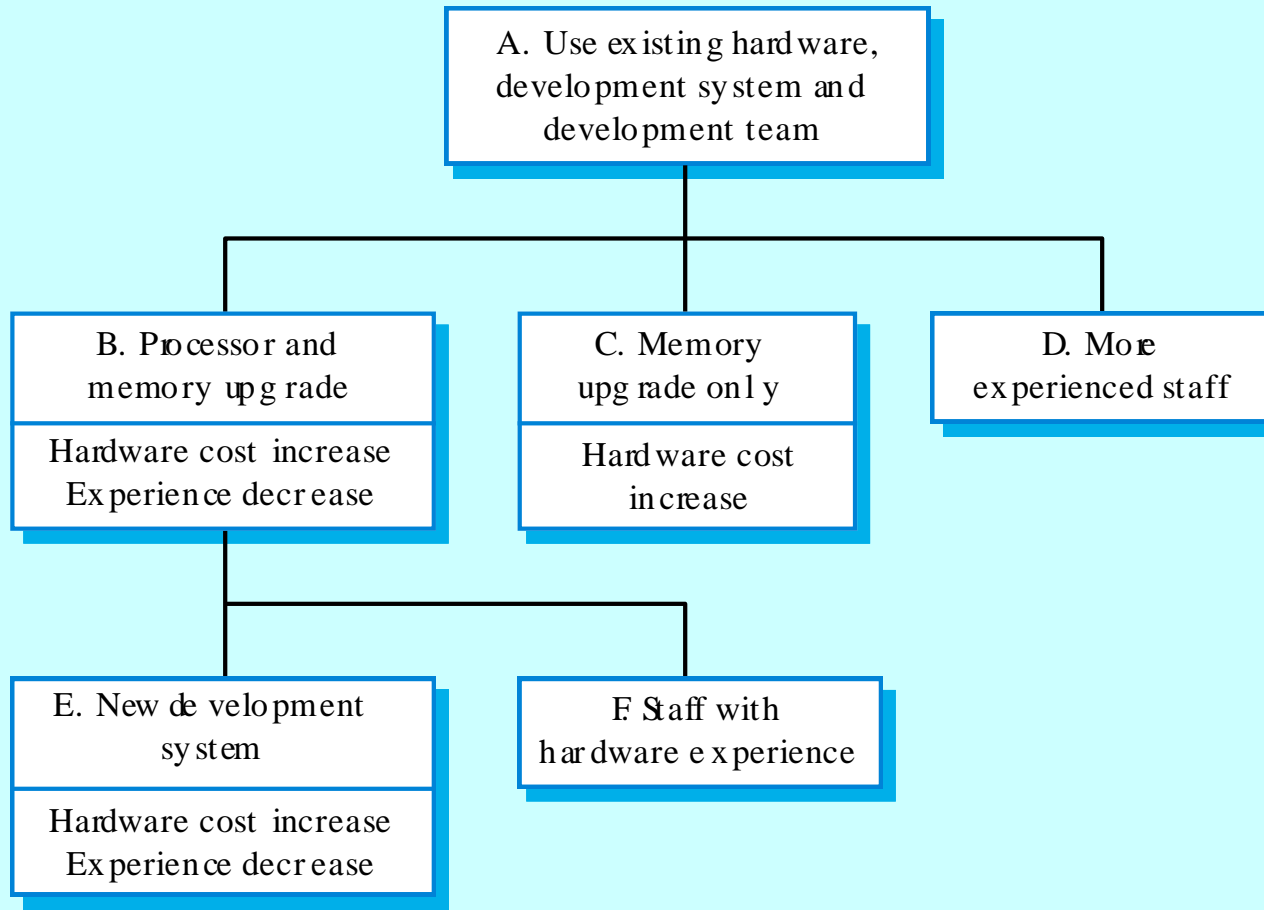
Management option costs

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*
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

- 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 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.

Staffing requirements

- Staff required can't be computed by dividing the development time by the required schedule.

- 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.

Key points

- There is not a simple relationship between the price charged for a system and its development costs.

- Factors affecting productivity include individual aptitude, domain experience, the development project, the project size, tool support and the working environment.

- Software may be priced to gain a contract and the functionality adjusted to the price.

Key points

- Different techniques of cost estimation should be used when estimating costs.

- The COCOMO model takes project, product, personnel and hardware attributes into account when predicting effort required.

- Algorithmic cost models support quantitative option analysis as they allow the costs of different options to be compared.

- The time to complete a project is not proportional to the number of people working on the project.