Software cost estimation

- Predicting the resources required for a software development process

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

Topics covered

- 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 and interleaved management activities

Importance of cost estimation

Estimating the costs of a software project is important for:

- *Software project personnel*: are the the proposed budget and schedule realistic ?
- *Software analysts*: are the hardware-software tradeoff analyses realistic ?
- *Project managers*: how much time and effort should each software phase and activity take ?

Incorrect estimates can cause:

- Problems for the users who have to revise their plans
- Problems for the project staff who has to work harder to compensate problems caused by others
- Moving money destined for other purposes to the project with consequent ripple effects
- Moving to the project resources from other projects thus causing delays in the schedule

Software cost components

- Hardware and software costs
- Travel and training costs
- Consumables
- Effort costs (the dominant factor in most projects)
  - 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.)
  - costs of support staff (secretary, administrative, technical)

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

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

- 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

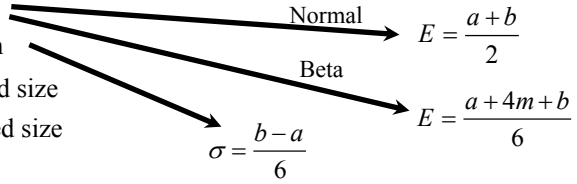
- Size related measures based on some output from the software process. This may be lines of delivered source code, object code instructions, etc.
- Function-related measures based on an estimate of the functionality of the delivered software. Function-points are the best known of this type of measure

Measurement problems

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

Estimating the size of software

- E: expected size
- σ: standard deviation
- a: minimum estimated size
- b: maximum estimated size
- m: likely size



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?
- Assumes 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 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

High and low level languages

Low-level language

High-level language

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	8 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
- The function point count is computed by multiplying each raw count by the weight and summing all values

Function points

- Function point count 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 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

Function Points

- 5 functional characteristics
  - *Number of input*
  - *Number of output*
  - *Number of queries*
  - *Number of files*
  - *Number of external interfaces*

	weight	
Simple	*Intermediate*	*Complex*
3	4	6
4	5	7
3	4	6
7	10	15
5	7	10

$$VP_i = C_i \times p_i$$

$$FP = \sum_{i=1}^{5}(VP_i) \times \left[0{,}65 + 0{,}01 \times \sum_{j=1}^{14} F_i\right]$$

14 adjustment parameters *Fi* varying from 0 to 5 depending on their influence

Data communications	On-line update
Distributed functions	Complex processing
Performances	Reusability
Heavily used configuration	Installation ease
Transaction rate	Operational ease
On-line data entry	Multiple site
End user efficiency	Facilitating change

0 *No incidence*
1 *Scarce incidence*
2 *Moderate incidence*
3 *Intermediate incidence*
4 *Meaningful incidence*
5 *Essential incidence*

Object points

- 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

Object point estimation

- Object points are easier to estimate from a specification than function points as they are simply concerned with screens, reports and 3GL modules
- They can therefore be estimated at an 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, 200-800 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

Factor	Description
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. 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 22, 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 change is constant then an approach based on counting lines of code 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 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

Estimation techniques

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

Estimates

- *Effort* (*MM*) in man-months
- *Development schedule* (*TDEV*) in months
- *Productivity* (Size/MM)
- *average staffing FSP = MM/TDEV*

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.
  - Group consensus techniques: *Delphi* (The Rand Corporation)
  - *Wideband Delphi* technique (Boehm *et al.*)
- Advantages: Relatively cheap estimation method. Can be accurate if experts have direct experience of similar systems
- Disadvantages: Very inaccurate if there are no experts!

Parkinson's Law

- The project costs whatever resources are available (*Work expands to fill the available volume*)
- Advantages: No overspend
- Disadvantages: System is usually unfinished

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

Estimation by analogy

- 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

Algorithmic cost modelling

- A formulaic approach based on historical cost information and which is generally based on the size of the software
- Discussed later in this chapter

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

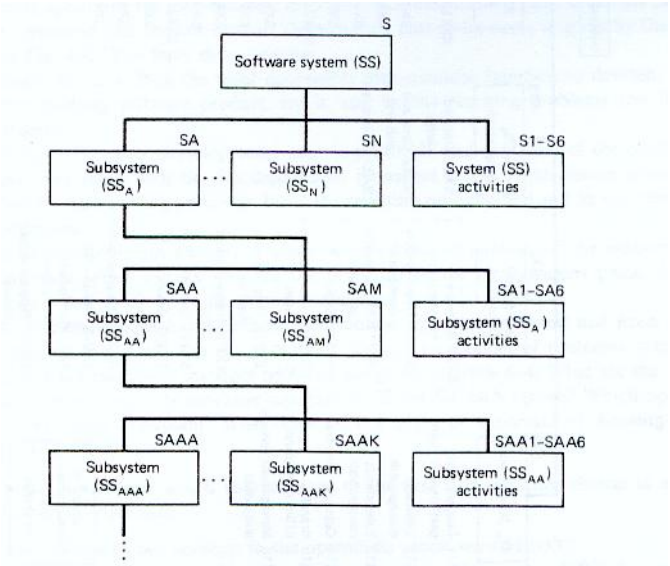
Bottom-up estimation

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

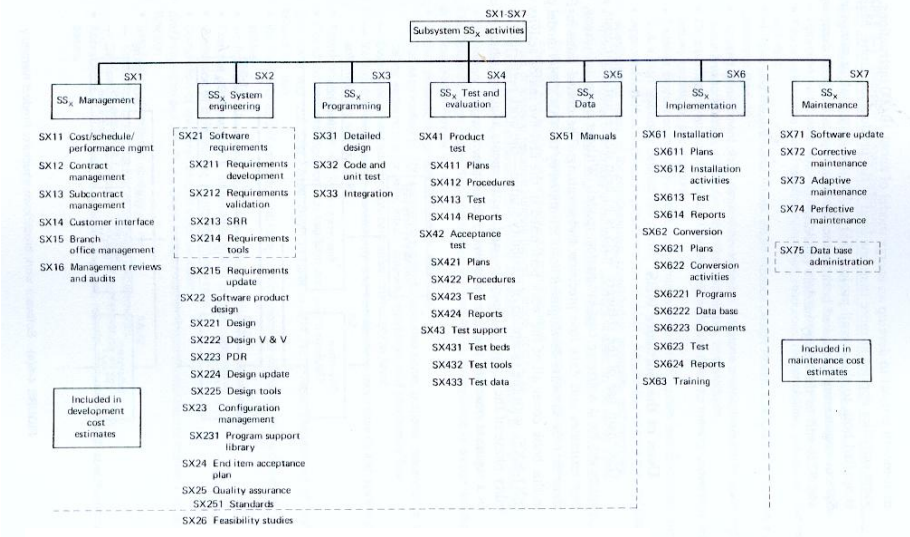
Work Breakdown Structure

- Two interconnected hierarchies
  - *Product Hierachy*. Decomposition of the system into software components
  - *Activity Hierarchy*. Indicates the activities related to a software component
- Utility
  - To define the costs to be estimated by the model
  - As basis to collect and produce reports of software costs
    - *A budget and a number of resources can be allocated to each element. This can be used to produce reports related to how much time and effort the different activities have consumed.*
    - *These data can be processed by a project information management system that enables the project manager to understand and control the time and effort consumed against the available budget*

WBS: Product hierarchy



WBS: Activity hierarchy



Estimation methods

- Each method has strengths and weaknesses
- 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

Experience-based estimates

- Estimating is primarily experience-based
- 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

Pricing to win

- This approach may seem unethical and unbusinesslike
- 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

Estimation by analogy

- Based on reusing the experience made in previous projects
  - Evaluates the analogy of the new project with similar past projects
- Applicable when:
  - No methodology has been adopted yet for cost estimation
  - A quick, simple, and reasonable accurate macro-estimates is required
  - Information from previous projects exists but no detailed historical database is available
- Requirements for the application:
  - Size and purposes of the new project are similar to that of the reference project
  - Size and purpose of the new product are similar to that of the reference product
  - Methods of work of the new project are similar to that of the reference project
  - Availability of detailed information or of people who remember the work on the reference project in a very accurate way
    - » in general any source of information + estimation experience

Estimation by analogy

- The estimates is made by evaluating the expected quantity of work (man-months) and the development time (months)
  - Application to the costs of the reference project of multiplicative coefficients (*modifiers*) taking into account the differences of given *characteristics* with respect to the reference project

$$m_i = 1 \pm a_i$$

$$C = \prod_i m_i$$

$$M = M_{rif} \times C$$

- Essential requirements in the choice of the reference project:
  - Knowledge of which characteristics are similar or different
  - Knowledge of the quality of available effort data

Consider only the characteristics directly related to the software; peripheral characteristics that can be estimated directly should not be taken into account

Some Characteristics

- *The system and its structure*
  - Type and usage of the system
  - Type and size of the data base
  - Number and type of the system interfaces
- *The user of the software and its environment*
  - Number of users
  - Number of terminals
  - Number of user locations and geographical distribution
- *The construction of the system and the proposed schedule*
  - Hardware configuration
  - Third-party software
  - Project approach and system structure
  - Manpower organization and workload
  - Skills of the development and project team
  - Used software tools
  - Proposed schedule
- *The system implementation and the operating environment*
  - Target hardware access
  - System location
  - User location
  - Installation and testing requirements

An example

	estimate	*modifier*
System characteristics more complex; about 15%	+15%	1.15
More menus and screens; about 10%	+10%	1.10
Structure of data base file less complex; about 5%	–5%	0.95
More system interfaces; about 20%	+20%	1.20
Use of DBMS less familiar; about 20%	+20%	1.20
More powerful development tools; about 10%	–10%	0.90
Development team with more experience; about 20%	–20%	0.80
Acceptance testing more rigorous; about 5%	+5%	1.05

$$C = 1.15 \times 1.10 \times 0.95 \times 1.20 \times 1.20 \times 0.90 \times 0.80 \times 1.05 = 1.31$$

In theory no limitations to the number of modifiers that can be used.

In practice no more than 15 modifiers should be used in general and no more than 10 should be used if one of them has a percentage variation greater than 30%

Detailed Estimates

- Post-project reviews activities are needed to build a detailed data base
- Single modules can be used as references
  - Important when no single project exists that can be entirely used as reference and when parts of other projects are more suitable as reference
- Rules to follow:
  - Verify whether a modifier can be better applied to the system or to a module
  - Do not use a modifier with percentage variations greater than 30% or less than 5%
  - Pay attention to projects conducted by different development teams during time
  - Where possible, apply the modifiers to resource units (man-months) and then multiply by the unit cost
  - Where the differences are mainly additive (due to more work) do not use modifiers, but add another activity to the project

Algorithmic cost modelling

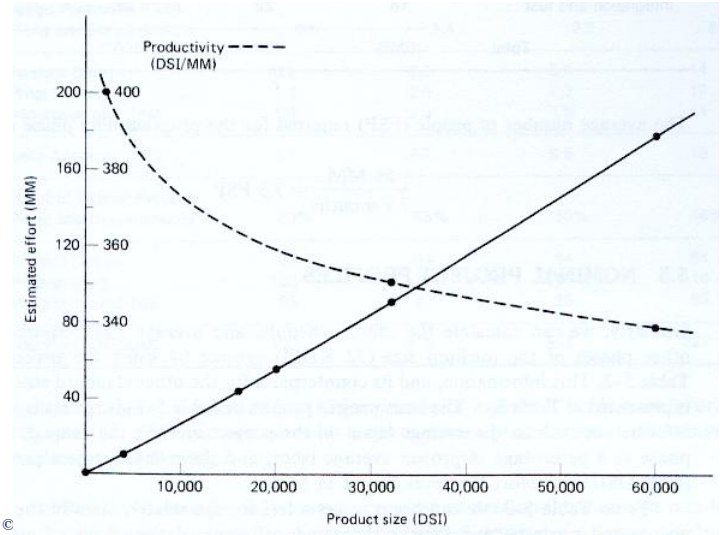
- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers
- COCOMO (COnstructive COst MOdel)
  - Effort = A ´ Size^{B} ´ M
  - A is an organisation-dependent constant, B reflects the disproportionate effort for large projects 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 basically similar but with different values for A, B and M

Example: Basic COCOMO estimates for organic mode projects

Project Profile	DSI	MM	DSI/MM	TDEV	FSP
• *Small*	2,000	5.0	400	4.6	1.1
• *Intermediate*	8,000	21.3	376	8.0	2.7
• *Medium*	32,000	91.0	352	14.0	6.5
• *Large*	128,000	392.0	327	24.0	16.0

- DSI: Delivered Source Instructions
- MM: Man-Month (Effort)
- TDEV: Development Time
- FSP: Full-time equivalent Software Personnel (Average Staffing)

Basic COCOMO Effort estimates (Organic mode)

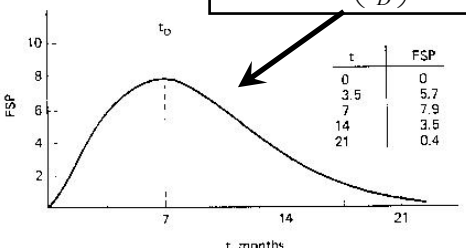


Diseconomy of Scale

- *Effort is not proportional to the size of the project (growth is more than linear) … Why ?*
- Greater effort in the software design phase to develop module specifications to suppport parallel development of more programmers
- Greater effort to verify and validate larger requirements and software specifications
- More time for communication and to resolve interface problems among programmers
- More integration testing effort
- More system testing effort
- More management effort

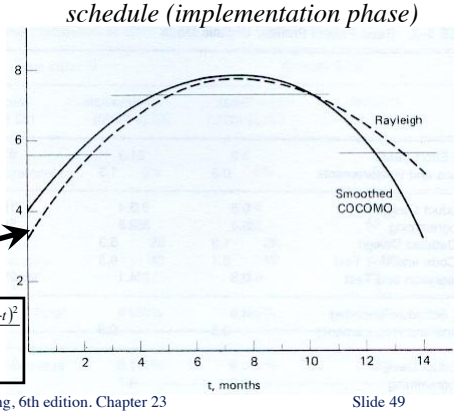
Distribution of effort among the phases of a project

Rayleigh curve $FSP = MM \cdot \left(\frac{t}{t_D^2}\right) \cdot e^{-\frac{t^2}{2 \cdot t_D^2}}$



t	FSP
0	0
3.5	5.7
7	7.9
14	3.5
21	0.4

- *An approximation of the Rayleigh curve already used to model brain intensive team work*
- *The peak is on the half of the schedule (implementation phase)*



$$FSP = MM \cdot \left(\frac{0.15 \cdot TDEV + 0.7 \cdot t}{0.25 \cdot (TDEV)^2}\right) \cdot e^{-\frac{(0.15 \cdot TDEV + 0.7 \cdot t)^2}{0.5 (TDEV)^2}}$$

Other algorithmic cost models

- Algorithmic models
  - Linear models (System Development Corporation) $\text{Effort} = a_0 + a_1 x_1 + \cdots + a_n x_n$
  - Multiplicative models (*IBM Walston-Felix*, *Doty*) $\text{Effort} = a_0 a_1^{x_1} a_2^{x_2} \cdots a_n^{x_n}$
  - Analytic models
    - *Halstead* $\text{Effort} = \frac{\eta_1 N_2 N \log_2 \eta}{2 S \eta_2}$
    - *Putnam* $S_s = C_k K^{1/3} t_d^{4/3} \Leftrightarrow K = \frac{L^3}{C_k^3 t_d^4}$
    - *Bailey-Basili* $M_{N\text{om}} = 3.5 + 0.73 S^{1.16}$
  - Tabular models: *Aron*, *Wolverton*, *Boeing* (Black *et al.*)
- Composite models: *RCA PRICE S* (Freiman, Park), *Putnam SLIM* (Putnam, Fitzsimmons), *TRW SCEP* (Boehm, Wolverton), *COCOMO* (Boehm)

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