Software cost estimation

- Predicting the resourcesrequired for a software development process

Objectives

- To introduce the fundamentals of software costingand pricing
- To describe three metrics for software productivityassessment
- To explain why different techniques should beused for software estimation
- To describe the COCOMO 2 algorithmic costestimation model

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

Software cost components

- Hardware and software costs
- Travel and training costs
- 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.)

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

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

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

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 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
- 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

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

Algorithmic code 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

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

- 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

Parkinson's Law

- The project costs whatever resources are available
- 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

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

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

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 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

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	$PM = 2.4\ (KDSI)^{1.05} \times M$	Ω ελεινδερασοδ απλιχατιονσ δεσελοπεδ βψ σμαλτεαμισ.
Μοδερατε	$\Pi M = 3.0\ (K\Delta\Sigma I)^{1.12} \times M$	Μορε χομπλεξ προφχτσωηερε τεαμ μεμβερσ μαψηαπε λιμιτεδ εξπεριενχε οφ ρελατεδ σψστεμσ.
Εμβεδδεδ	$\Pi M = 3.6\ (K\Delta\Sigma I)^{1.20} \times M$	Χομπλεξ προφχτσωηερε τηε σοφτωαρε ισ παρτοφ α τρονγλψ χουπλεδ χομπλεξ οφηαρδωαρε σοφτωαρε, ρεγυλατορνσανδ οπερατιοναλπροχεδυρεσ

COCOMO 2 levels

- COCOMO 2 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
- Post-architecture level
  - Estimates based on lines of source code

Early prototyping level

- 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 × (1 - %reuse/100)) / PROD
  - PM is the effort in person-months, NOP is the number of object 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 level

- Estimates can be made after the requirements have been agreed
- Based on standard formula for algorithmic models
  - $PM = A \times Size^{B} \times M + PM_m$ where
  - $M = PERS \times RCPX \times RUSE \times PDIF \times PREX \times FCIL \times SCED$
  - $PM_m = (ASLOC \times (AT/100)) / ATPROD$
  - A = 2.5 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.
  - 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
- PM reflects the amount of automatically generated code

Post-architecture level

- 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 × (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.

The exponent term

- This depends on 5 scale factors (see next slide). Their sum/100 is added to 1.01
- Example
  - Precedenteness - 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

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.

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

Project 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

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
  - 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 costs

Option choice

- Option D (use more experienced staff) appears to be the best alternative
  - However, it has a high associated risk as expreienced 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 diving 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

- Factors affecting productivity include individual aptitude, domain experience, the development project, the project size, tool support and the working environment
- Different techniques of cost estimation should be used when estimating costs
- Software may be priced to gain a contract and the functionality adjusted to the price

Key points

- Algorithmic cost estimation is difficult because of the need to estimate attributes of the finished product
- The COCOMO model takes project, product, personnel and hardware attributes into account when predicting effort required
- Algorithmic cost models support quantitative option analysis
- The time to complete a project is not proportional to the number of people working on the project