PLANNING AND ESTIMATING

Prof. Ralph Laviste
College of Computer Studies
Cebu Institute of Technology University

Outline

- Planning and the software process
- Estimating duration and cost
- Software Size Metrics
- IEEE software project management plan
- Planning of testing
- Planning of object-oriented projects
- Training requirements
- Documentation standards

Planning and Estimating

- Before starting to build software, it is essential to plan the *entire* development effort *in detail*
- Planning continues during development and then maintenance
  - Initial planning is not enough
  - The earliest possible detailed planning is after the specification phase

Estimating Duration and Cost

- Accurate duration estimation is critical
- Accurate cost estimation is critical
  - Internal, external costs
- There are too many variables for accurate estimate of cost or duration

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 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.

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.

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.
ParkinsonÕs Law	ParkinsonÕsLaw 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 customerÕs budget and not on the software functionality.

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.

Metrics for the Size of a Product

- Lines of Code (LOC)
- Software Science
- FFP
- Function Points
- COCOMO

Lines of Code

- Lines of code (LOC), or
- Thousand delivered source instructions (KDSI)
  - Source code is only a small part of total software effort
  - Different languages $\Rightarrow$ different lengths of code
  - LOC not defined for nonprocedural languages (like LISP)
  - It is not clear how to count lines of code
    - Executable lines of code?
    - Data definitions ?
    - Comments?
    - JCL statements?
    - Changed/deleted lines?
  - Not everything written is delivered to the client

Lines of Code

- LOC is known when the product finished
- Estimation based on LOC is doubly dangerous
  - To start estimation process, LOC in finished product must be estimated
  - LOC estimate is then used to estimate the cost of the product — uncertain input to an uncertain cost estimator

Software Science

- Metrics based on number of operands, operators
- Limited predictive power—metrics can be computed only after the product has been implemented
- There are major doubts about the validity of Software Science

Metrics for the Size of a Product

- Metrics based on measurable quantities that can be determined early in software life cycle
  - FFP
  - Function Points

FFP Metric

- For cost estimation of medium-scale Data Processing systems
- The three basic structural elements of Data Processing systems
  - files, flows, and processes
- Given number of files (Fi), flows (Fl), processes (Pr)
  - Size (S), cost (C) given by

    $S = Fi + Fl + Pr$

    $C = b \times S$
- Constant b varies from organization to organization

FFP Metric

- Validity and reliability of FFP metric were demonstrated using a purposive sample
  - BUT, the metric was never extended to include databases

Function Points

- Based on number of inputs (Inp), outputs (Out), inquiries (Inq), master files (Maf), interfaces (Inf)
- For any product, size in “function points” is given by

$$FP = 4 \times Inp + 5 \times Out + 4 \times Inq + 10 \times Maf + 7 \times Inf$$

Function Points

1. Classify each component of product (Inp, Out, Inq, Maf, Inf) as simple, average, or complex.
   - Assign appropriate number of function points
   - Sum gives UFP (unadjusted function points)

	Level of Complexity		
Component	**Simple**	**Average**	**Complex**
Input item	3	4	6
Output item	4	5	7
Inquiry	3	4	6
Master file	7	10	15
Interface	5	7	10

Function Points

2. Compute technical complexity factor (TCF)
   - Assign value from 0 ("not present") to 5 ("strong influence throughout") to each of 14 factors
   - Add 14 numbers $\Rightarrow$ total degree of influence (DI)

     $$TCF = 0.65 + 0.01 \times DI$$

   - Technical complexity factor (TCF) lies between 0.65 and 1.35

1. Data communication
2. Distributed data processing
3. Performance criteria
4. Heavily utilized hardware
5. High transaction rates
6. Online data entry
7. End-user efficiency
8. Online updating
9. Complex computations
10. Reusability
11. Ease of installation
12. Ease of operation
13. Portability
14. Maintainability

Function Points

3. Number of function points (FP) given by

$$FP = UFP \times TCF$$

Analysis of Function Points

- Function points are usually better than KDSI
- Like FFP, maintenance can be inaccurately measured

Techniques of Cost Estimation

- Expert judgment by analogy
- Experts compare target product to completed products
  - Guesses can lead to hopelessly incorrect cost estimates
  - Experts may recollect completed products inaccurately
  - Human experts have biases
  - However, results of estimation by broad group of experts may be accurate

Techniques of Cost Estimation

- Bottom-up approach
- Break product into smaller components
  - Smaller components may be no easier to estimate
  - Process-level costs

Techniques of Cost Estimation

- Algorithmic models
- Metric used as input to model to compute cost, duration
  - An algorithmic model is unbiased, and superior to expert opinion
  - However, estimates are only as good as the underlying assumptions
- Examples
  - SLIM Model
  - Price S Model
  - COnstructive COst MOdel (COCOMO)
- COCOMO consists of three models
  - Macro-estimation model for product as a whole
  - Intermediate COCOMO
  - Micro-estimation model which treats product in detail

Intermediate COCOMO

1. Estimate length of product in KDSI

Intermediate COCOMO

2. Estimate product development mode (organic, semidetached, embedded)

- Example
  - Straightforward product (“organic mode”)

  Nominal effort = $3.2 \times (\text{KDSI})^{1.05}$ person-months

Intermediate COCOMO

3. Compute nominal effort

- Example
  - Organic product, est. 12,000 delivered source statements (12 KDSI)

  Nominal effort = $3.2 \times (12)^{1.05} = 43$ person-months

Intermediate COCOMO

4. Multiply nominal value by 15 software development cost multipliers

- Example
  - Product complexity multiplier
- Intermodule control and decision tables

Intermediate COCOMO

Cost Drivers	Rating					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliablity	0.75	0.88	1.00	1.15	1.40	
Database size		0.94	1.00	1.08	1.16	
Product complexity	0.70	0.85	1.00	1.15	1.30	1.65
Computer attributes						
Execution time constraint			1.00	1.11	1.30	1.66
Main storage constraint			1.00	1.06	1.21	1.56
Virtual machine volatility*		0.87	1.00	1.15	1.30	
Computer turnaround time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capabilities	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Programmer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience*	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Use of modern programming practices	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

*For a given software product, the underlying virtual machine is the complex of hardware and software (operating system, database management system) it calls on to accomplish its task.

- Software development effort multipliers

Intermediate COCOMO

- Example
  - Microprocessor-based communications processing software for electronic funds transfer network with high reliability, performance, development schedule, and interface requirements

1. Complex (“embedded”) mode

Intermediate COCOMO

2. Estimate: 10,000 delivered source instructions

Intermediate COCOMO

3. Nominal effort = $2.8 \times (10)^{1.20} = 44$ person-months

Intermediate COCOMO

- 4. Product of effort multipliers = 1.35, so estimated effort for project is
  - 1.35 $\times$ 44 = 59 person-months

Intermediate COCOMO

Cost Drivers	Situation	Rating	Effort Multiplier
Required software reliability	Serious financial consequences of software fault	High	1.15
Database size	20,000 bytes	Low	0.94
Product complexity	Communications processing	Very high	1.30
Execution time constraint	Will use 70% of available time	High	1.11
Main storage constraint	45K of 64K store (70%)	High	1.06
Virtrual machine volatility	Based on commercial microprocessor hardware	Nominal	1.00
Computer turnaround time	Two hour average turnaround time	Nominal	1.00
Analyst capabilities	Good senior analysts	High	0.86
Applications experience	Three years	Nominal	1.00
Programmer capability	Good senior programmers	High	0.86
Virtual machines experience	Six months	Low	1.10
Programming language experience	Twelve months	Nominal	1.00
Use of modern programming practices	Most techniques in use over one year	High	0.91
Use of software tools	At basic minicomputer tool level	Low	1.10
Required development schedule	Nine months	Nominal	1.00

Intermediate COCOMO

- Estimated effort for project (59 person-months) used as input for additional formulas for
  - Dollar costs
  - Development schedules
  - Phase and activity distributions
  - Computer costs
  - Annual maintenance costs
  - Related items

Intermediate COCOMO

- Intermediate COCOMO has been validated with respect to broad sample
- Actual values within 20% of predicted values about 68% of time
  - Intermediate COCOMO was most accurate estimation method of its time

COCOMO II

- 1995 extension to 1981 COCOMO that incorporates
  - Object orientation
  - Modern life-cycle models
  - Rapid prototyping
  - Fourth-generation languages
  - COTS software
- COCOMO II is far more complex than the first version

COCOMO II

- Three different models
  - Application composition model for early phases
    - Based on feature points (like function points)
  - Early design model
    - Based on function points
  - Post-architecture model
    - Based on function points or KDSI

COCOMO II

- COCOMO Effort model is effort = a (size)^b
  - Intermediate COCOMO
    - Three values for (a, b)
  - COCOMO II
    - b varies depending on values of certain parameters
- COCOMO II supports reuse

COCOMO II

- COCOMO II has 17 multiplicative cost drivers (was 15)
  - Seven are new
- It is too soon for results regarding
  - Accuracy of COCOMO II
  - Extent of improvement (if any) over Intermediate COCOMO

Tracking Duration and Cost Estimates

- Whatever estimation method used, careful tracking is vital
- Components of a software project management plan (SPMP)
  - Work to be done
  - Resources with which to do it
  - Money to pay for it

Resources

- Resources needed for software development:
  - People
  - Hardware
  - Support software

Work Categories

- Project function
  - Work carried on throughout project
  - Examples:

  project management, quality control
- Activity
  - Work that relates to a specific phase
  - Major unit of work
  - With precise beginning and ending dates
  - That consumes *resources*, and
  - Results in *work products* like budget, design, schedules, source code, or users' manual
- Task
  - An activity comprises a set of *tasks* (the smallest unit of work subject to management accountability)

Completion of Work Products

- Milestone: Date on which the work product is to be completed
- It must first pass *reviews* performed by
  - Fellow team members
  - Management
  - Client
- Once the work product has been reviewed and agreed upon, it becomes a *baseline*

Work Package

- Work product, *plus*
  - Staffing requirements
  - Duration
  - Resources
  - Name of responsible individual
  - Acceptance criteria for work product
- Money
  - Vital component of plan
  - Detailed budget must be worked out as a function of time
  - Money must be allocated, as function of time, to
    - Project functions
    - Activities

How to Plan Software Development

- State problem clearly (Specification Phase)
- Determine viable solution strategies (Specification Phase)
- Should client be advised to computerize?
  - Cost–benefit analysis
- If so, which viable solution strategy? Decide by
  - Minimizing total cost to client, or
  - Maximizing total return on investments, or
  - Other methods
- Develop SPMP for product as whole

Software Project Management Plan (SPMP)

- Determine work units
- Estimate resources required
- Draw up budget
- Come up with detailed timetable

Framework for SPMP

- IEEE Standard 1058.1
  - Standard widely agreed upon
  - Designed for use with all types of software product
  - Advantages of standardization

IEEE SPMP

1. Introduction
   - 1.1 Project Overview
   - 1.2 Project Deliverables
   - 1.3 Evolution of the Software Project Management Plan
   - 1.4 Reference Materials
   - 1.5 Definitions and Acronyms
2. Project Organization
   - 2.1 Process Model
   - 2.2 Organizational Structure
   - 2.3 Organizational Boundaries and Interfaces
   - 2.4 Project Responsibilities
3. Managerial Process
   - 3.1 Management Objectives and Priorities
   - 3.2 Assumptions, Dependencies, and Constraints
   - 3.3 Risk Management
   - 3.4 Monitoring and Controlling Mechanisms
   - 3.5 Staffing Plan
4. Technical Process
   - 4.1 Methods, Tools, and Techniques
   - 4.2 Software Documentation
   - 4.3 Project Support Functions
5. Work Packages, Schedule, and Budget
   - 5.1 Work Packages
   - 5.2 Dependencies
   - 5.3 Resources Requirements
   - 5.4 Budget and Resource Allocation
   - 5.5 Schedule

Additional components

Planning of Testing

- SPMP must explicitly state what testing is to be done
  - Traceability
  - All black box test cases must be drawn up as soon as possible after specifications are complete

Planning of Object-Oriented Projects

- Object-oriented product consists of *largely* independent pieces
- Planning is *somewhat* easier
- Whole is more than the sum of its parts
- Use COCOMO II (or modify intermediate COCOMO estimators)

Planning of Object-Oriented Projects

- However, reuse destroys everything
  - Reuse of existing components during development
  - Production of components for future reuse
- These work in opposite directions
- Newer data: savings outweigh costs

Training Requirements

- “We don’t need to worry about training until the product is finished, and then we can train the user ”
  - Training is generally needed by the members of the development group, starting with training in software planning
  - New software development method necessitates training for every member of the group
  - Introduction of hardware or software tools of any sort necessitates training
  - Programmers may need training in the operating system, implementation language
  - Documentation preparation training may be needed
  - Computer operators require training

Documentation Standards

- How much documentation is generated by a product?
  - IBM internal commercial product (50 KDSI)
    - 28 pages of documentation per KDSI
  - Commercial software product of same size
    - 66 pages per KDSI
  - IMS/360 Version 2.3 (about 166 KDSI)
    - 157 pages of documentation per KDSI
  - (TRW) For every 100 hours spent on coding activities, 150–200 hours were spent on documentation-related activities

Types of Documentation

- Planning
- Control
- Financial
- Technical
- Source code
- Comments within source code

Documentation Standards

- Reduce misunderstandings between team members
- Aid SQA
- Only new employees have to learn standards
- Standards assist maintenance programmers
- Standardization is important for user manuals

CASE Tools for the Planning Phase

- Word processor, spreadsheet
- Automated intermediate COCOMO/COCOMO II
- Management tools assist with planning and monitoring
  - MacProject, Microsoft Project

Testing during the Planning Phase

- Must check SPMP as a whole
- Pay particular attention to duration and cost estimates