CHAPTER 9

PLANNING AND ESTIMATING

1

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 postdelivery maintenance
  - Initial planning is not enough
  - Planning must proceed throughout the project
  - The earliest possible time that detailed planning can take place is after the specifications are complete

2

9.1 Planning and the Software Process

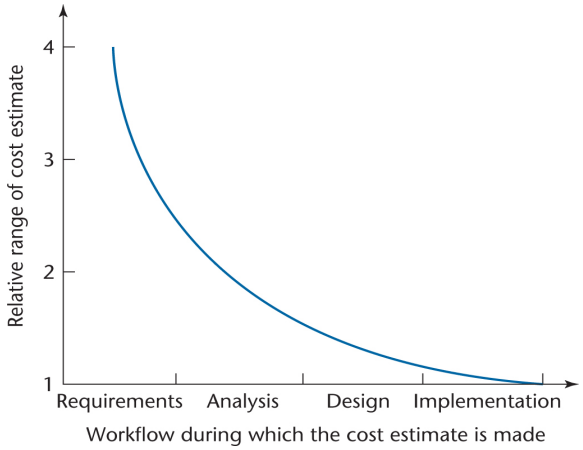

Figure 9.1

- The accuracy of estimation increases as the process proceeds

3

Planning and the Software Process

- Example
  - Cost estimate of $1 million during the requirements workflow
    - Likely actual cost is in the range ($0.25M, $4M)
  - Cost estimate of $1 million at the end of the requirements workflow
    - Likely actual cost is in the range ($0.5M, $2M)
  - Cost estimate of $1 million at the end of the analysis workflow (earliest appropriate time)
    - Likely actual cost is in the range ($0.67M, $1.5M)

4

Planning and the Software Process

- These four points are shown in the *cone of uncertainty*

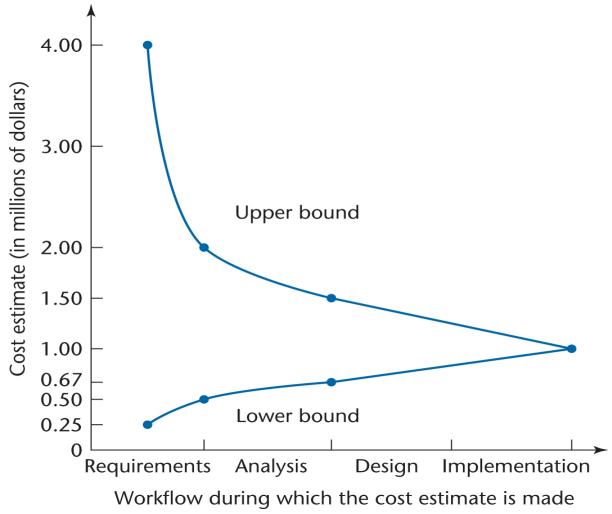

Figure 9.2

5

Planning and the Software Process

- This model is old (1976)
  - Estimating techniques have improved
  - But the shape of the curve is likely to be similar

6

9.2 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

7

Human Factors

- Sackman (1968) measured differences of up to 28 to 1 between pairs of programmers
  - He compared matched pairs of programmers with respect to
    - Product size
    - Product execution time
    - Development time
    - Coding time
    - Debugging time
- Critical staff members may resign during the project

8

9.2.1 Metrics for the Size of a Product

- Lines of code (LOC, KDSI, KLOC)
- FFP (see page 273 & 274)
- Function Points
- COCOMO

9

Lines of Code (LOC)

- Alternate metric
  - Thousand delivered source instructions (KDSI)
- Source code is only a small part of the total software effort
- Different languages lead to different lengths of code
- LOC is not defined for nonprocedural languages (like LISP)

10

Lines of Code

- 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
- A report, screen, or GUI generator can generate thousands of lines of code in minutes

11

Lines of Code

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

12

Metrics for the Size of a Product

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

13

Metrics for the Size of a Product

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

14

FFP Metric

- For cost estimation of medium-scale data processing products
- The three basic structural elements of data processing products
  - **F**iles
  - **F**lows
  - **P**rocesses

15

FFP Metric (page 273)

- Given the number of files (*Fi*), flows (*Fl*), and processes (*Pr*)
  - The size (*S*), cost (*C*) are given by

$$S = Fi + Fl + Pr$$

$$C = b \times S$$

- The constant *b* (efficiency or productivity) varies from organization to organization

16

FFP Metric

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

17

Function Points (273 & 274)

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

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

- This is an oversimplification of a 3-step process
- [UFP>unadjusted function points]

18

Function Points

- Step 1. Classify each component of the product (*Inp*, *Out*, *Inq*, *Maf*, *Inf*) as simple, average, or complex
  - Assign the appropriate number of function points
  - The 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

Figure 9.3

19

Function Points

- Step 2. Compute the technical complexity factor (*TCF*)
  - Assign a value from 0 ("not present") to 5 ("strong influence throughout") to each of 14 factors such as transaction rates, portability

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

Figure 9.4

20

Function Points

- Add the 14 numbers
  - This gives the total degree of influence (*DI*)

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

- The technical complexity factor (*TCF*) lies between 0.65 and 1.35

21

Function Points

- Step 3. The number of function points (*FP*) is then given by

$$FP = UFP \times TCF$$

The real function points!

22

Analysis of Function Points

- Function points are usually better than KDSI — but there are some problems
- "Errors in excess of 800% counting KDSI, but *only* 200% in counting function points" [Jones, 1987]

23

Analysis of ~~Function Points~~ KDSI

- We obtain nonsensical results from metrics
  - KDSI per person month and
  - Cost per source statement

	Assembler Version	Ada Version
Source code size	70 KDSI	25 KDSI
Development costs	$1,043,000	$590,000
KDSI per person-month	0.335	0.211
Cost per source statement	$14.90	$23.60
Function points per person-month	1.65	2.92
Cost per function point	$3,023	$1,170

Figure 9.5

- Cost per function point is meaningful

24

Analysis of Function Points

- Like FFP, maintenance can be inaccurately measured
- It is possible to make major changes without changing
  - The number of files, flows, and processes; or
  - The number of inputs, outputs, inquiries, master files, and interfaces
- In theory, it is possible to change every line of code with changing the number of lines of code

25

Mk II Function Points

- This metric was put forward to compute *UFP* more accurately
- We decompose software into component transactions, each consisting of input, process, and output
- Mark II function points are widely used all over the world

26

9.2.2 Techniques of Cost Estimation

- Expert judgment by analogy
- Bottom-up approach
- Algorithmic cost estimation models

27

Expert Judgment by Analogy

- Experts compare the target product to completed products
  - Guesses can lead to hopelessly incorrect cost estimates
  - Experts may recollect completed products inaccurately
  - Human experts have biases
  - However, the results of estimation by a broad group of experts may be accurate
- The Delphi technique is sometimes needed to achieve consensus

28

Bottom-up Approach

- Break the product into smaller components
  - The smaller components may be no easier to estimate
  - However, there are process-level costs
- When using the object-oriented paradigm
  - The independence of the classes assists here
  - However, the interactions among the classes complicate the estimation process

29

Algorithmic Cost Estimation Models

- A metric is used as an input to a model to compute cost and duration
  - An algorithmic model is unbiased, and therefore superior to expert opinion
  - However, estimates are only as good as the underlying assumptions
- Examples
  - SLIM Model
  - Price S Model
  - COnstructive COst MOdel (COCOMO)

30

9.2.3 Intermediate COCOMO

- COCOMO consists of three models
  - A macro-estimation model for the product as a whole
  - Intermediate COCOMO
  - A micro-estimation model that treats the product in detail
- We examine intermediate COCOMO

31

Intermediate COCOMO

- Step 1. Estimate the length of the product in KDSI
- Figure out the function points
- Pick your language
- http://www.cs.bsu.edu/homepages/dmz/cs697/langtbl.htm
- Multiply by ***AVERAGE SOURCE STATEMENTS PER FUNCTION POINT***
- Convert to KDSI (Thousands of instructions)

32

Intermediate COCOMO

- Step 2. Estimate the product development mode (organic, semidetached, embedded)
- Easy, medium, and really hard

33

Intermediate COCOMO

- Step 3. Compute the nominal effort
- Nominal Effort = ai * $(KDSI)^{bi}$

- Example:
  - Organic product
  - 12,000 delivered source statements (12 KDSI) (estimated)

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

Software project	a_i	b_i
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

34

Intermediate COCOMO

- Step 4. Multiply the nominal value by 15 software development cost multipliers

	Rating					
Cost Drivers	**Very Low**	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
Product Attributes						
Required software reliability	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.

Figure 9.6

35

Intermediate COCOMO

- Example:
  - Microprocessor-based communications processing software for electronic funds transfer network with high reliability, performance, development schedule, and interface requirements
- Step 1. Estimate the length of the product
  - 10,000 delivered source instructions (10 KDSI)
- Step 2. Estimate the product development mode
  - Complex (“embedded”) mode

36

Intermediate COCOMO (contd)

- Step 3. Compute the nominal effort
  - Nominal effort = 2.8 x $(10)^{1.20}$ = 44 person-months

- Step 4. Multiply the nominal value by 15 software development cost multipliers
  - Product of effort multipliers = 1.35 (page)
  - Estimated effort for project is therefore 1.35 x 44 = 59 person-months

http://sunset.usc.edu/research/COCOMOII/cocomo81_pgm/cocomo81.html

37

Intermediate COCOMO

- Software development effort multipliers

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
Virtual machine volatility	Based on commercial microprocessor hardware	Nominal	1.00
Computer turnaround time	2 hour average turnaround time	Nominal	1.00
Analyst capabilities	Good senior analysts	High	0.86
Applications experience	3 years	Nominal	1.00
Programmer capability	Good senior programmers	High	0.86
Virtual machine experience	6 months	Low	1.10
Programming language experience	12 months	Nominal	1.00
Use of modern programming practices	Most techniques in use over 1 year	High	0.91
Use of software tools	At basic minicomputer tool level	Low	1.10
Required development schedule	9 months	Nominal	1.00

Figure 9.7

38

Intermediate COCOMO

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

39

Intermediate COCOMO

- Intermediate COCOMO has been validated with respect to a broad sample
- Actual values are within 20% of predicted values about 68% of the time
  - Intermediate COCOMO was the most accurate estimation method of its time
- Major problem
  - If the estimate of the number of lines of codes of the target product is incorrect, then everything is incorrect

40

9.2.4 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

41

9.2.5 Tracking Duration and Cost Estimates

- Whatever estimation method used, careful tracking is vital

45

9.3 Components of a Software Project Management Plan

- The work to be done
- The resources with which to do it
- The money to pay for it

46

Resources

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

47

Use of Resources Varies with Time

- Rayleigh curves accurately depict resource consumption
- The entire software development plan must be a function of time

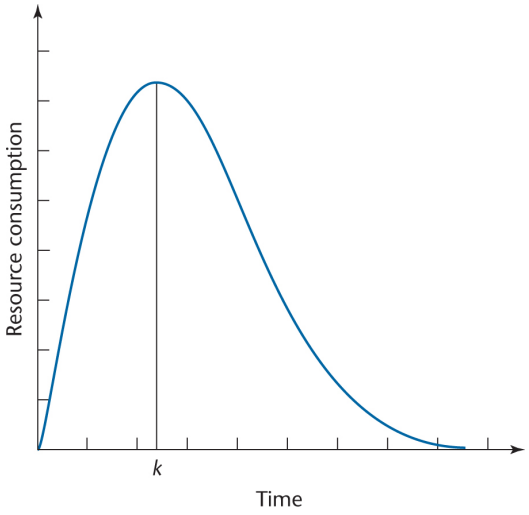

Figure 9.8

48

The End!

49