Software Project Management

Chapter Five

Software effort estimation

What makes a successful project?

Delivering:

- agreed functionality
- on time at the agreed cost
- with the required quality

Stages:

1. Set targets
2. Attempt to achieve targets

BUT what if the targets are not achievable?

Some problems with estimating

- Subjective nature of much of estimating
  - It may be difficult to produce evidence to support your precise target
- Political pressures
  - Managers may wish to reduce estimated costs in order to win support for acceptance of a project proposal
- Changing technologies
  - these bring uncertainties, especially in the early days when there is a ‘learning curve’
- Projects differ
  - Experience on one project may not be applicable to another

Over and under-estimating

- Parkinson's Law: 'Work expands to fill the time available'
- An over-estimate is likely to cause project to take longer than it would otherwise
- Weinberg's Zeroth Law of reliability: 'a software project that does not have to meet a reliability requirement can meet any other requirement'

Basis for successful estimating

- Information about past projects
  - Need to collect performance details about past project: how big were they? How much effort/time did they need?
- Need to be able to measure the amount of work involved
  - Traditional size measurement for software is ‘lines of code’ – but this can have problems

A taxonomy of estimating methods

- Bottom-up - activity based, analytical
- Parametric or algorithmic models e.g. function points
- Expert opinion - just guessing?
- Analogy - case-based, comparative
- Parkinson and ‘price to win’

Parameters to be Estimated

- Size is a fundamental measure of work
- Based on the estimated size, two parameters are estimated:
  - Effort
  - Duration
- Effort is measured in person-months:
  - One person-month is the effort an individual can typically put in a month.

Person-Month

- Suppose a project is estimated to take 300 person-months to develop:
  - Is one person working for 30 days same as 30 persons working for 1 day?
  - Yes/No? why?
- How many hours is a man month?
  - Default Value: 152 hours per month
  - 19 days at 8 hours per day.

Mythical Man-Month

- "Cost varies as product of men and months, progress does not."
  - Hence the man-month as a unit for measuring the size of job is a dangerous and deceptive myth.
- The myth of additional manpower
  - Brooks Law: "Adding manpower to a late project makes it later"

Mythical Man-Month

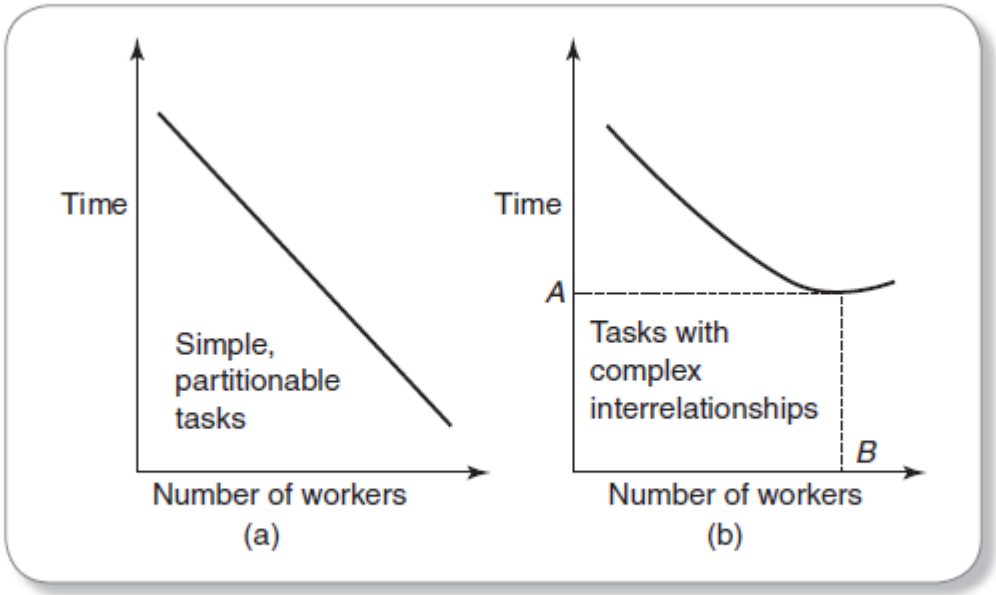


FIGURE 5.2 Impact of addition of workers on the completion time for various types of projects

For tasks with complex interrelationship, addition of manpower to a late project does not help.

Measure of Work

- The project size is a measure of the problem complexity in terms of the effort and time required to develop the product.
- Two metrics are used to measure project size:
  - Source Lines of Code (SLOC)
  - Function point (FP)
- FP is now-a-days favoured over SLOC:
  - Because of the many shortcomings of SLOC.

Major Shortcomings of SLOC

- Difficult to estimate at start of a project
- Only a code measure
- Programmer-dependent
- Does not consider code complexity

Bottom-up versus top-down

- Bottom-up
  - use when no past project data
  - identify all tasks that have to be done – so quite time-consuming
  - use when you have no data about similar past projects
- Top-down
  - produce overall estimate based on project cost drivers
  - based on past project data
  - divide overall estimate between jobs to be done

Bottom-up estimating

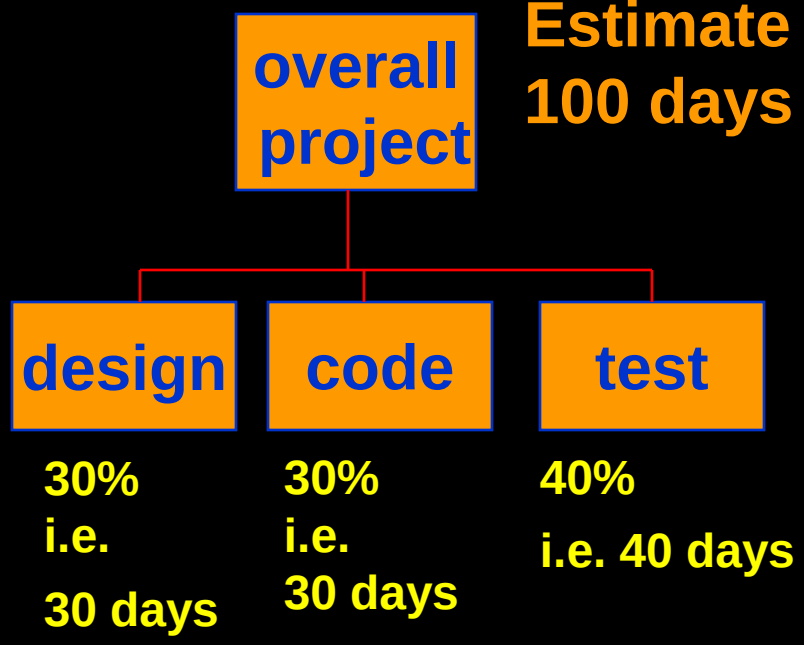
1. Break project into smaller and smaller components

[2. Stop when you get to what one person can do in one/two weeks]

3. Estimate costs for the lowest level activities

4. At each higher level calculate estimate by adding estimates for lower levels

Top-down estimates



- Produce overall estimate using effort driver(s)
- distribute proportions of overall estimate to components

Algorithmic/Parametric models

- COCOMO (lines of code) and function points examples of these
- Problem with COCOMO etc:

but what is desired is

Parametric models - the need for historical data

- simplistic model for an estimate

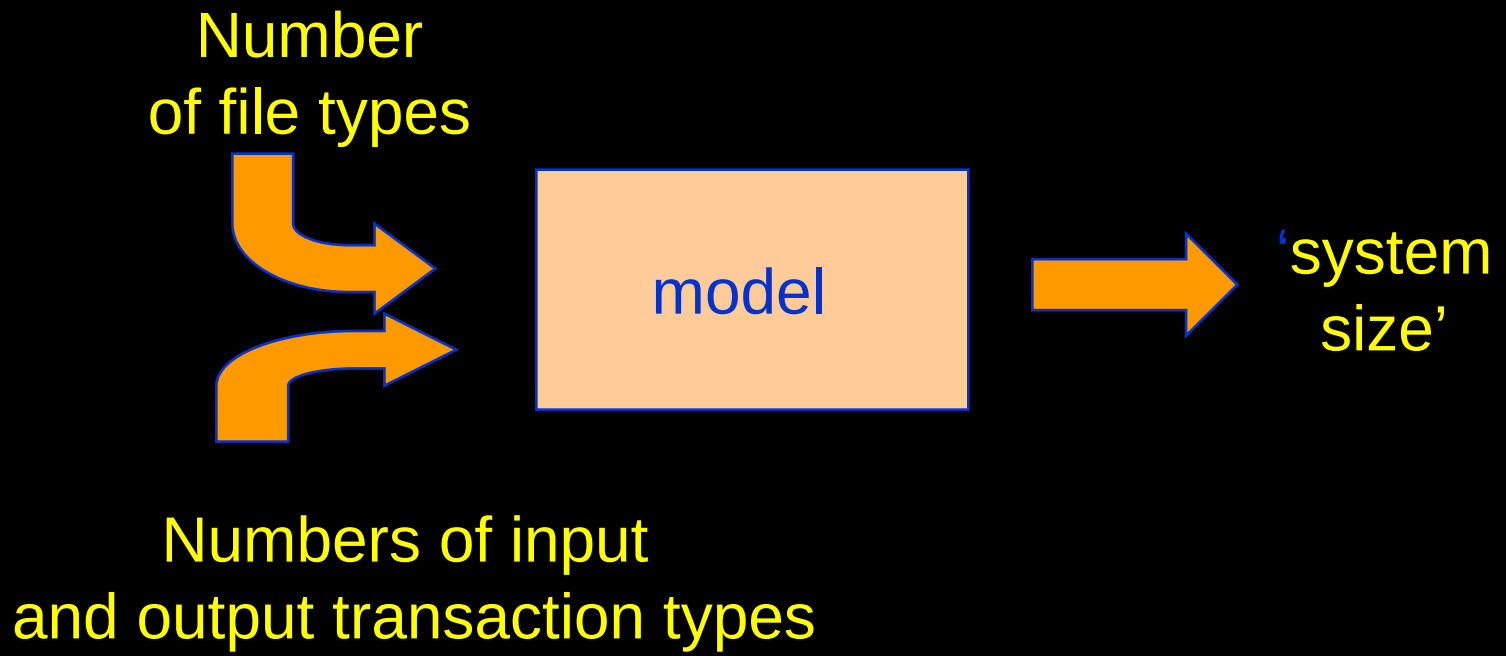
  estimated effort = (system size) / productivity
- e.g.

  system size = lines of code

  productivity = lines of code per day
- productivity = (system size) / effort
  - based on past projects

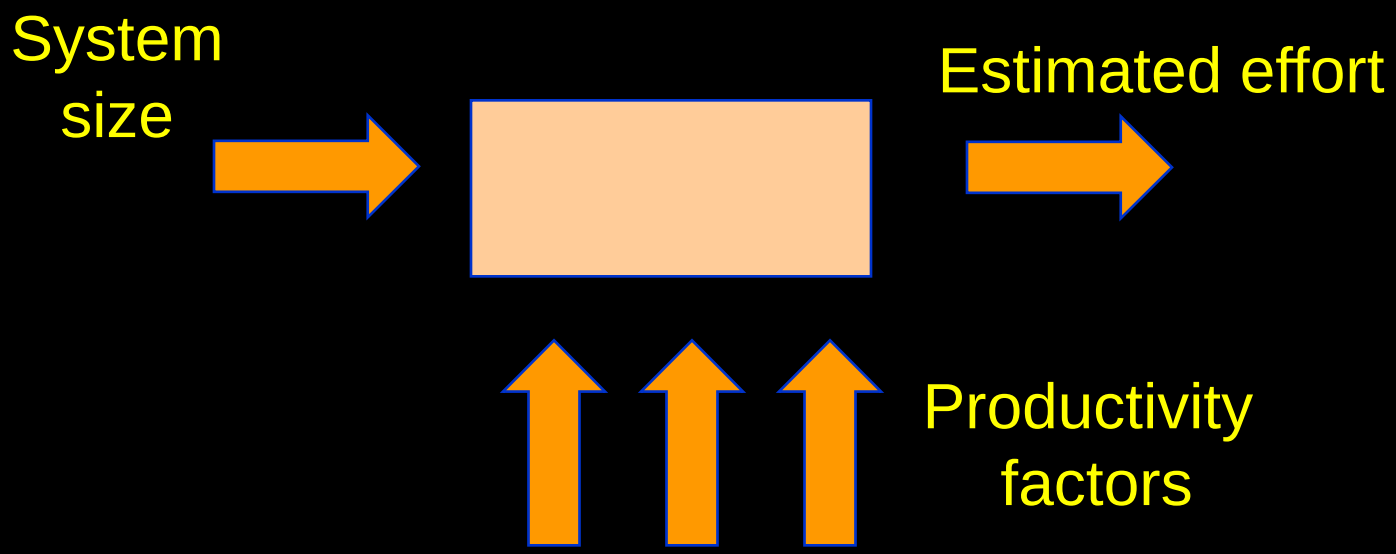
Parametric models

- Some models focus on task or system size e.g. Function Points
- FPs originally used to estimate Lines of Code, rather than effort



Parametric models

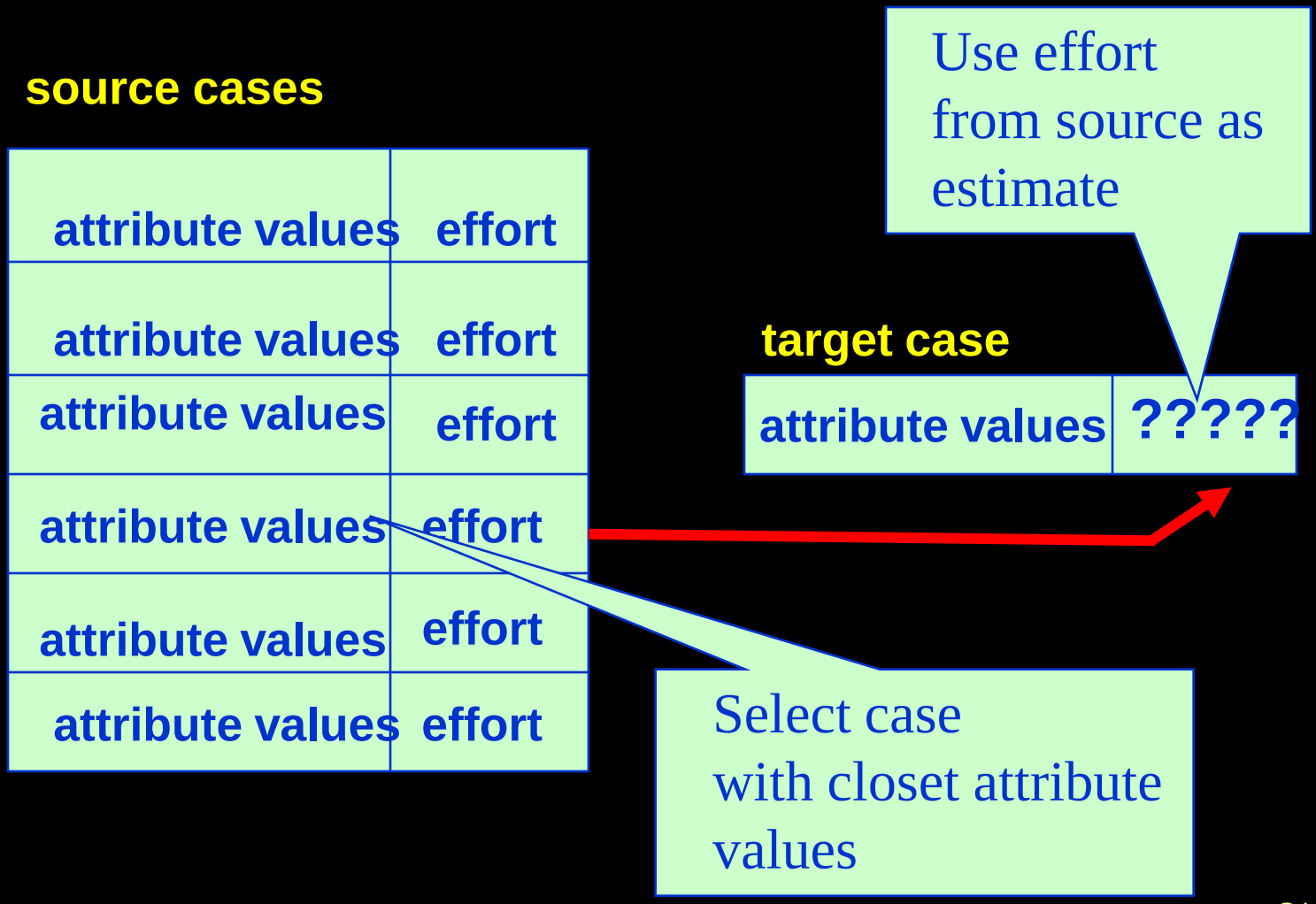
- Other models focus on productivity: e.g. COCOMO
- Lines of code (or FPs etc) an input



Expert judgement

- Asking someone who is familiar with and knowledgeable about the application area and the technologies to provide an estimate
- Particularly appropriate where existing code is to be modified
- Research shows that experts judgement in practice tends to be based on analogy

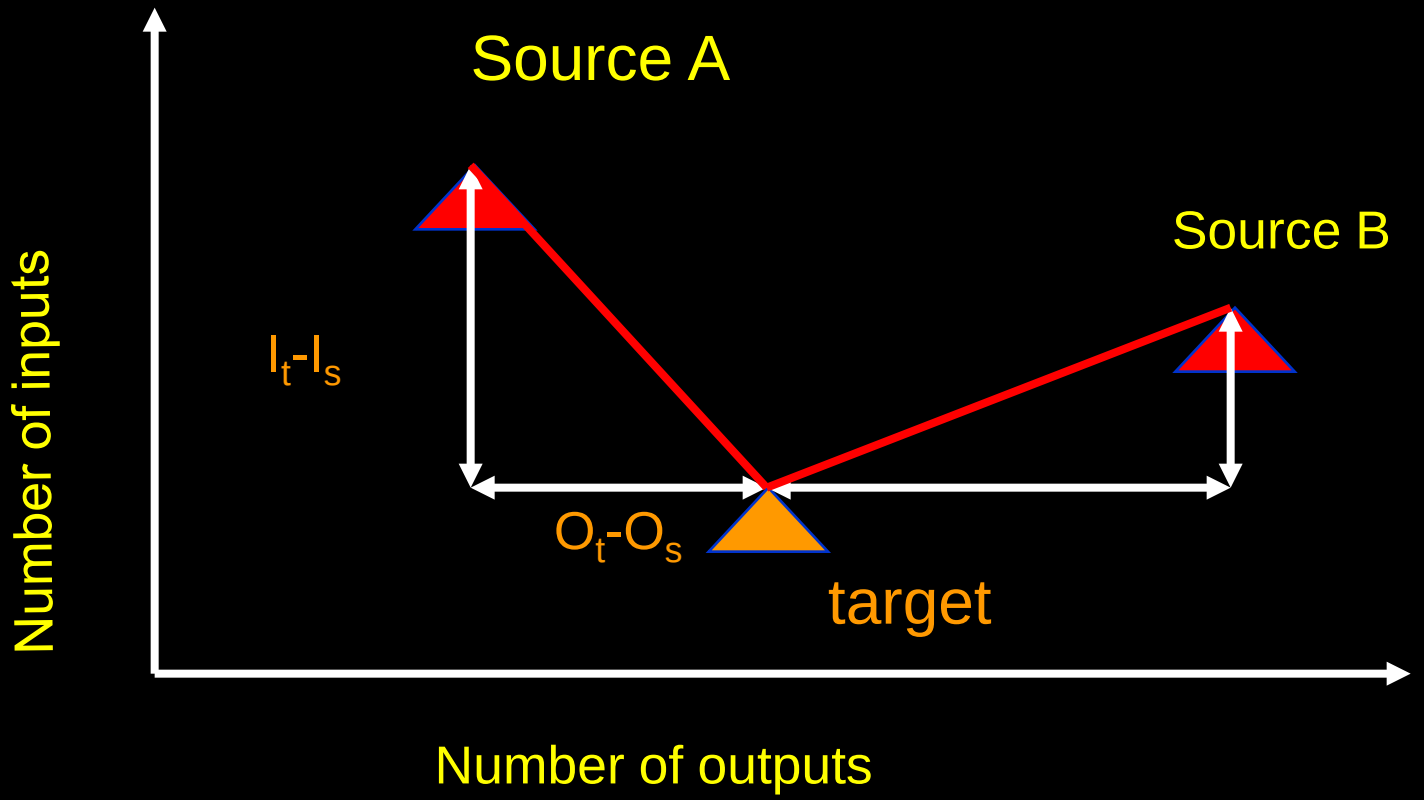
Estimating by analogy



Stages: identify

- Significant features of the current project
- previous project(s) with similar features
- differences between the current and previous projects
- possible reasons for error (risk)
- measures to reduce uncertainty

Machine assistance for source selection (ANGEL)



Euclidean distance = sq root $((I_t - I_s)^2 + (O_t - O_s)^2)$

Parametric models

We are now looking more closely at four parametric models:

- Albrecht/IFPUG function points
- Symons/Mark II function points
- COSMIC function points
- COCOMO81 and COCOMO II

Albrecht/IFPUG function points

- Albrecht worked at IBM and needed a way of measuring the relative productivity of different programming languages.
- Needed some way of measuring the size of an application without counting lines of code.
- Identified five types of component or functionality in an information system
- Counted occurrences of each type of functionality in order to get an indication of the size of an information system

Albrecht/IFPUG function points - continued

Five function types

1. Logical interface file (LIF) types – equates roughly to a data store in systems analysis terms. Created and accessed by the target system
2. External interface file types (EIF) – where data is retrieved from a data store which is actually maintained by a different application.

Albrecht/IFPUG function points - continued

3. External input (EI) types – input transactions which update internal computer files
4. External output (EO) types – transactions which extract and display data from internal computer files. Generally involves creating reports.
5. External inquiry (EQ) types – user initiated transactions which provide information but do not update computer files. Normally the user inputs some data that guides the system to the information the user needs.

Albrecht complexity multipliers

External user types	Low complexity	Medium complexity	High complexity
EI	3	4	6
EO	4	5	7
EQ	3	4	6
LIF	7	10	15
EIF	5	7	10

Examples

Payroll application has:

1. **Transaction to input, amend and delete employee details – an EI that is rated of medium complexity**
2. **A transaction that calculates pay details from timesheet data that is input – an EI of high complexity**
3. **A transaction of medium complexity that prints out pay-to-date details for each employee – EO**
4. **A file of payroll details for each employee – assessed as of medium complexity LIF**
5. **A personnel file maintained by another system is accessed for name and address details – a simple EIF**

What would be the FP counts for these?

FP counts

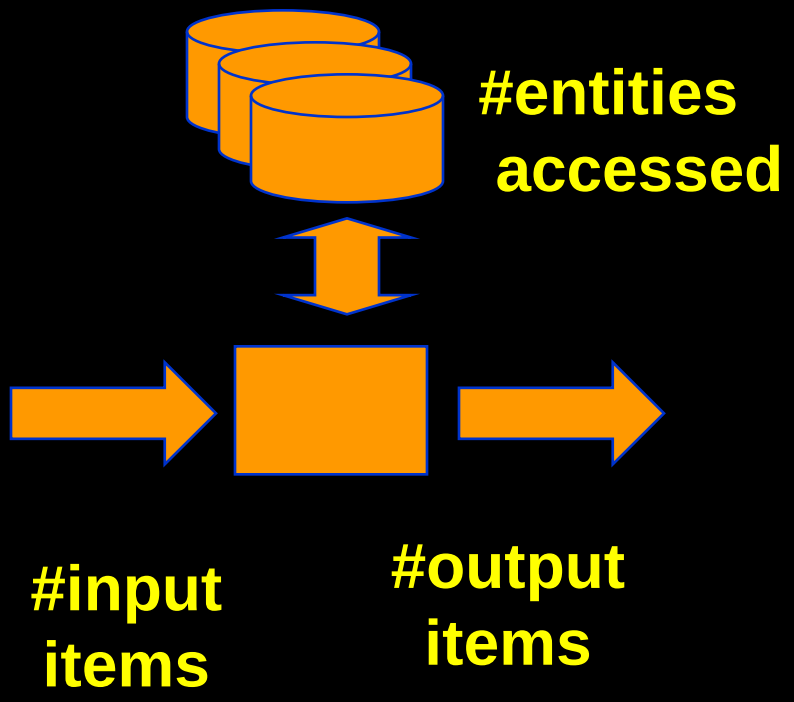
Medium EI	4 FPs
High complexity EI	6 FPs
Medium complexity EO	5 FPs
Medium complexity LIF	10 FPs
Simple EIF	5 FPs
Total	**30 FPs**

If previous projects delivered 5 FPs a day, implementing the above should take 30/5 = 6 days

Function points Mark II

- Developed by Charles R. Symons
- ‘Software sizing and estimating - Mk II FPA’, Wiley & Sons, 1991.
- Builds on work by Albrecht
- Work originally for CCTA:
  - should be compatible with SSADM; mainly used in UK
- has developed in parallel to IFPUG FPs
- A simpler method

Function points Mk II continued



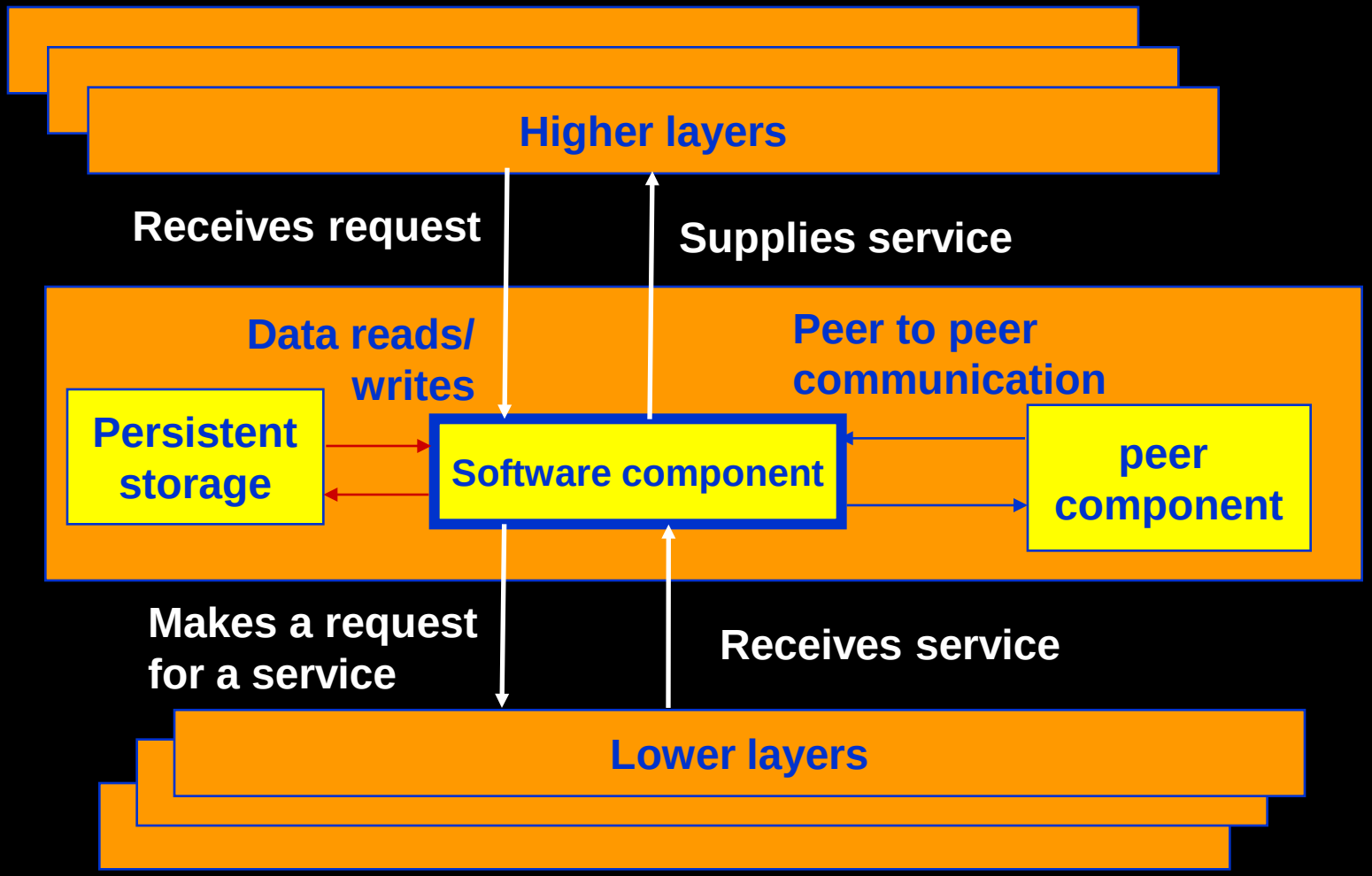
- For each transaction, count
  - data items input (N_i)
  - data items output (N_o)
  - entity types accessed (N_e)

FP count = N_i * 0.58 + N_e * 1.66 + N_o * 0.26

Function points for embedded systems

- Mark II function points, IFPUG function points were designed for information systems environments
- COSMIC FPs attempt to extend concept to embedded systems
- Embedded software seen as being in a particular ‘layer’ in the system
- Communicates with other layers and also other components at same level

Layered software



COSMIC FPs

The following are counted:

- Entries: movement of data into software component from a higher layer or a peer component
- Exits: movements of data out
- Reads: data movement from persistent storage
- Writes: data movement to persistent storage

Each counts as 1 ‘COSMIC functional size unit’ (Cfsu)

COCOMO81

- Based on industry productivity standards - database is constantly updated
- Allows an organization to benchmark its software development productivity
- **Basic model**

  effort = c x size^{k}

- C and k depend on the type of system: organic, semi-detached, embedded
- Size is measured in ‘kloc’ ie. Thousands of lines of code

The COCOMO constants

System type	c	k
Organic (broadly, information systems)	2.4	1.05
Semi-detached	3.0	1.12
Embedded (broadly, real-time)	3.6	1.20

k exponentiation – ‘to the power of…’
adds disproportionately more effort to the larger projects
takes account of bigger management overheads

Development effort multipliers (dem)

According to COCOMO, the major productivity drivers include:

Product attributes: required reliability, database size, product complexity

Computer attributes: execution time constraints, storage constraints, virtual machine (VM) volatility

Personnel attributes: analyst capability, application experience, VM experience, programming language experience

Project attributes: modern programming practices, software tools, schedule constraints

Using COCOMO development effort multipliers (dem)

An example: for analyst capability:

- Assess capability as very low, low, nominal, *high* or very high
- Extract multiplier:

very low	1.46
low	1.19
nominal	1.00
high	*0.80*
very high	0.71

- Adjust nominal estimate e.g. 32.6 x 0.80 = 26.8 staff months

As Time Passed… COCOMO 81 Showed Limitations…

- COCOMO 81 was developed with the assumption:
  - 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 practices:
  - Made it difficult to use COCOMO meaningfully.

Major Changes in Program Development Practices

- Software reuse
- Application generation of programs
- Object oriented approaches
- Need for rapid development
- Agile models

COCOMO II

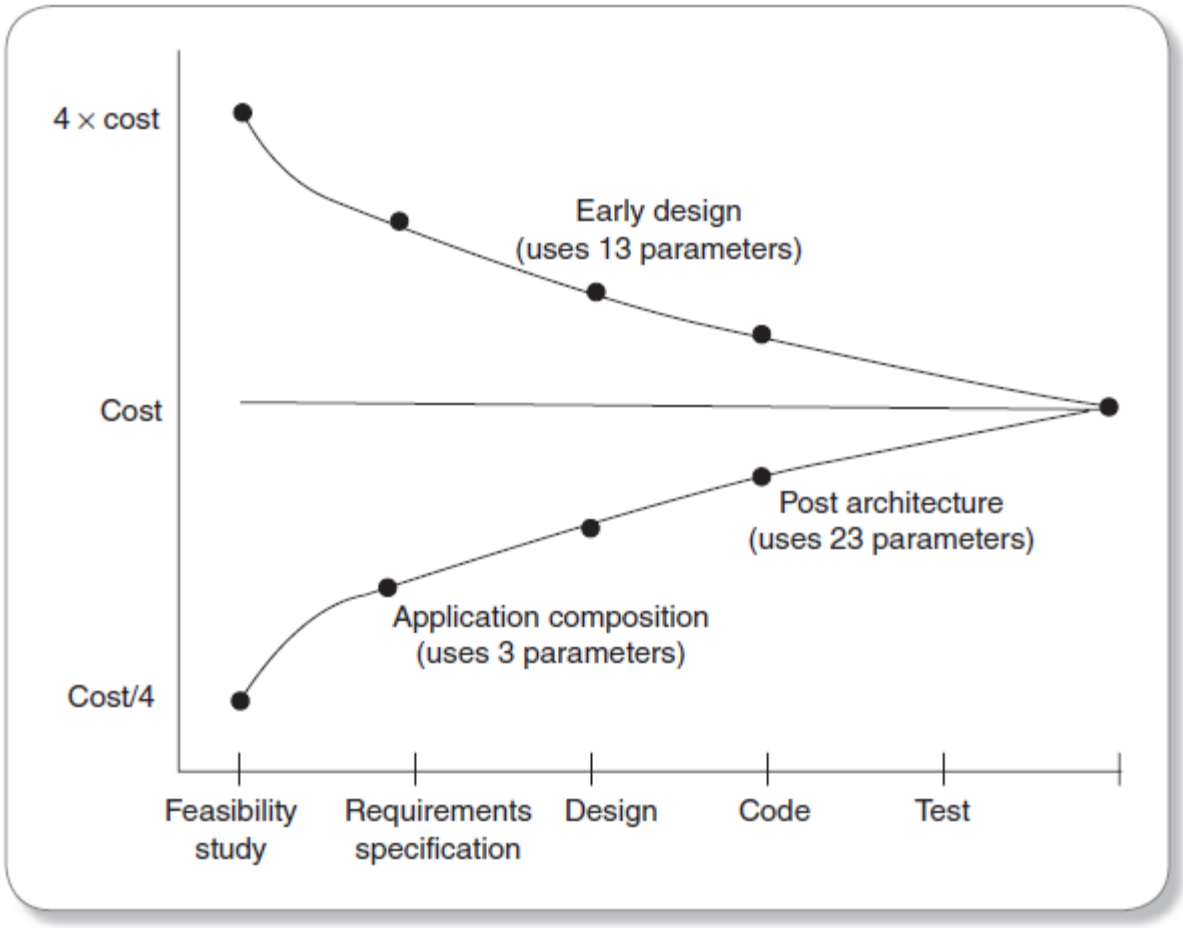


FIGURE 5.4 Accuracy of different COCOMO II estimations

COCOMO II Models

- COCOMO 2 incorporates a range of sub-models:
- Produces increasingly accurate estimates.
- The 4 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.

COCOMO II

An updated version of COCOMO:

- There are different COCOMO II models for estimating at the 'early design' stage and the 'post architecture' stage when the final system is implemented. We'll look specifically at the first.
- The core model is:

  $pm = A(size)^{(sf)} \times(em_1) \times(em_2) \times(em_3)\ldots.$

  where **pm** = person months, **A** is 2.94, **size** is number of thousands of lines of code, **sf** is the scale factor, and $\mathbf{em_i}$ is an effort multiplier

COCOMO II Scale factor

Based on five factors which appear to be particularly sensitive to system size

- Precedentedness (PREC). Degree to which there are past examples that can be consulted
- Development flexibility (FLEX). Degree of flexibility that exists when implementing the project
- Architecture/risk resolution (RESL). Degree of uncertainty about requirements
- Team cohesion (TEAM).
- Process maturity (PMAT) could be assessed by CMMI – see Section 13.10

COCOMO II Scale factor values

Driver	Very low	Low	Nom-inal	High	Very high	Extra high
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Example of scale factor

- A software development team is developing an application which is very similar to previous ones it has developed.
- A very precise software engineering document lays down very strict requirements. PREC is very high (score 1.24).
- FLEX is very low (score 5.07).
- The good news is that these tight requirements are unlikely to change (RESL is high with a score 2.83).
- The team is tightly knit (TEAM has high score of 2.19), but processes are informal (so PMAT is low and scores 6.24)

Scale factor calculation

The formula for sf is

$$sf = B + 0.01 \times \Sigma \text{ scale factor values}$$

i.e. $sf = 0.91 + 0.01$

$\times (1.24 + 5.07 + 2.83 + 2.19 + 6.24)$

$= 1.0857$

If system contained 10 kloc then estimate would be $2.94 \times 10^{1.0857} = 35.8$ person months

Using exponentiation ('to the power of') adds disproportionately more to the estimates for larger applications

Effort multipliers

As well as the scale factor effort multipliers are also assessed:

RCPX	Product reliability and complexity
RUSE	Reuse required
PDIF	Platform difficulty
PERS	Personnel capability
FCIL	Facilities available
SCED	Schedule pressure

Effort multipliers

	Extra low	Very low	Low	Nom-inal	High	Very high	Extra high
RCPX	0.49	0.60	0.83	1.00	1.33	1.91	2.72
RUSE			0.95	1.00	1.07	1.15	1.24
PDIF			0.87	1.00	1.29	1.81	2.61
PERS	2.12	1.62	1.26	1.00	0.83	0.63	0.50
PREX	1.59	1.33	1.12	1.00	0.87	0.74	0.62
FCIL	1.43	1.30	1.10	1.00	0.87	0.73	0.62
SCED		1.43	1.14	1.00	1.00	1.00	

Example

- Say that a new project is similar in most characteristics to those that an organization has been dealing for some time
- **except**
  - the software to be produced is exceptionally complex and will be used in a safety critical system.
  - The software will interface with a new operating system that is currently in beta status.
  - To deal with this the team allocated to the job are regarded as exceptionally good, but do not have a lot of experience on this type of software.

Example -continued

RCPX	very high	1.91
PDIF	very high	1.81
PERS	extra high	0.50
PREX	nominal	1.00

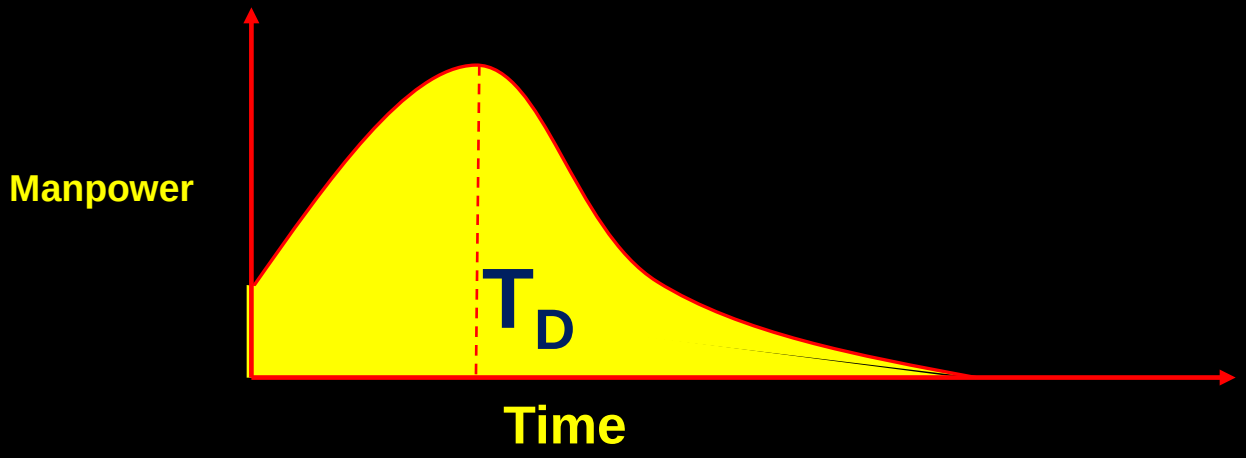
All other factors are nominal

Say estimate is 35.8 person months

With effort multipliers this becomes 35.8 x 1.91 x 1.81 x 0.5 = 61.9 person months

Staffing

- Norden was one of the first to investigate staffing pattern:
  - Considered general research and development (R&D) type of projects.
- Norden concluded:
  - Staffing pattern for any R&D project can be approximated by the Rayleigh distribution curve



Putnam's Work

- Putnam adapted the Rayleigh-Norden curve:
  - Related the number of delivered lines of code to the effort and the time required to develop the product.
  - Studied the effect of schedule compression:

Example

- If the estimated development time using COCOMO formulas is 1 year:
  - Then to develop the product in 6 months, the total effort required (and hence the project cost) increases 16 times.

Boehm's Result

- There is a limit beyond which a software project cannot reduce its schedule by buying any more personnel or equipment.
  - This limit occurs roughly at 75% of the nominal time estimate for small and medium sized projects

Capers Jones' Estimating Rules of Thumb

- Empirical rules:
  - Formulated based on observations
  - No scientific basis
- Because of their simplicity,:
  - These rules are handy to use for making off-hand estimates.
  - Give an insight into many aspects of a project for which no formal methodologies exist yet.

Capers Jones' Rules

- *Rule 1: SLOC-function point equivalence:*
  - One function point = 125 SLOC for C programs.
- Rule 2: Project duration estimation:
  - Function points raised to the power 0.4 predicts the approximate development time in calendar months.
- Rule 3: Rate of requirements creep:
  - User requirements creep in at an average rate of 2% per month from the design through coding phases.

Capers Jones' Rules

- Rule 4: Defect removal efficiency:
  - Each software review, inspection, or test step will find and remove 30% of the bugs that are present.
- Rule 5: Project manpower estimation:
  - The size of the software (in function points) divided by 150 predicts the approximate number of personnel required for developing the application.

Capers' Jones Rules

- Rule 6: Number of personnel for maintenance
  - *Function points divided by 500 predicts the approximate number of personnel required for regular maintenance activities.*
- Rule 7: Software development effort estimation:
  - The approximate number of staff months of effort required to develop a software is given by the software development time multiplied with the number of personnel required.

Some conclusions: how to review estimates

Ask the following questions about an estimate

- What are the task size drivers?
- What productivity rates have been used?
- Is there an example of a previous project of about the same size?
- Are there examples of where the productivity rates used have actually been found?