Politecnico
di Milano

Project Management Criticals: Co$t E$timation!

Damian Andrew Tamburri

Credits to:
Elisabetta Di Nitto,
Barry Bohem,
Ian Sommerville,
Luigi Lavazza

Roadmap

- Necessary Recalls
- Software Cost
- Software Cost, the Estimation Problem
- Algorithmic Techniques: Function-Points (FP)
- Algorithmic Techniques: COnstructive COst MOdel (COCOMO)

Roadmap

- **Necessary Recalls**
- Software Cost
- Software Cost, the Estimation Problem
- Algorithmic Techniques: Function-Points (FP)
- Algorithmic Techniques: COnstructive COst MOdel (COCOMO)

Necessary Recalls

- Software engineering concerns VERY BIG systems! ☺
- For such systems, Brooks' Law stands ☺
  - "Adding manpower to a late project will only make it later" [Cit. F.P. Brooks, *The mythical man-month*]
- The software myth! ☺
  - Productivity is constant and can be determined by management
  - Product is directly proportional to effort

Roadmap

- Necessary Recalls
- **Software Cost**
- Software Cost, the Estimation Problem
- Algorithmic Techniques: Function-Points (FP)
- Algorithmic Techniques: COnstructive COst MOdel (COCOMO)

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.).

Software Cost: Costing and pricing

- Estimates are made to discover the cost, to the developer, of producing a software system.
- There is a non-trivial relationship between the development cost and the price charged to the customer.
- Broader **organisational, social, economic, political and business** considerations influence the price charged.

Software Cost: Known Factors

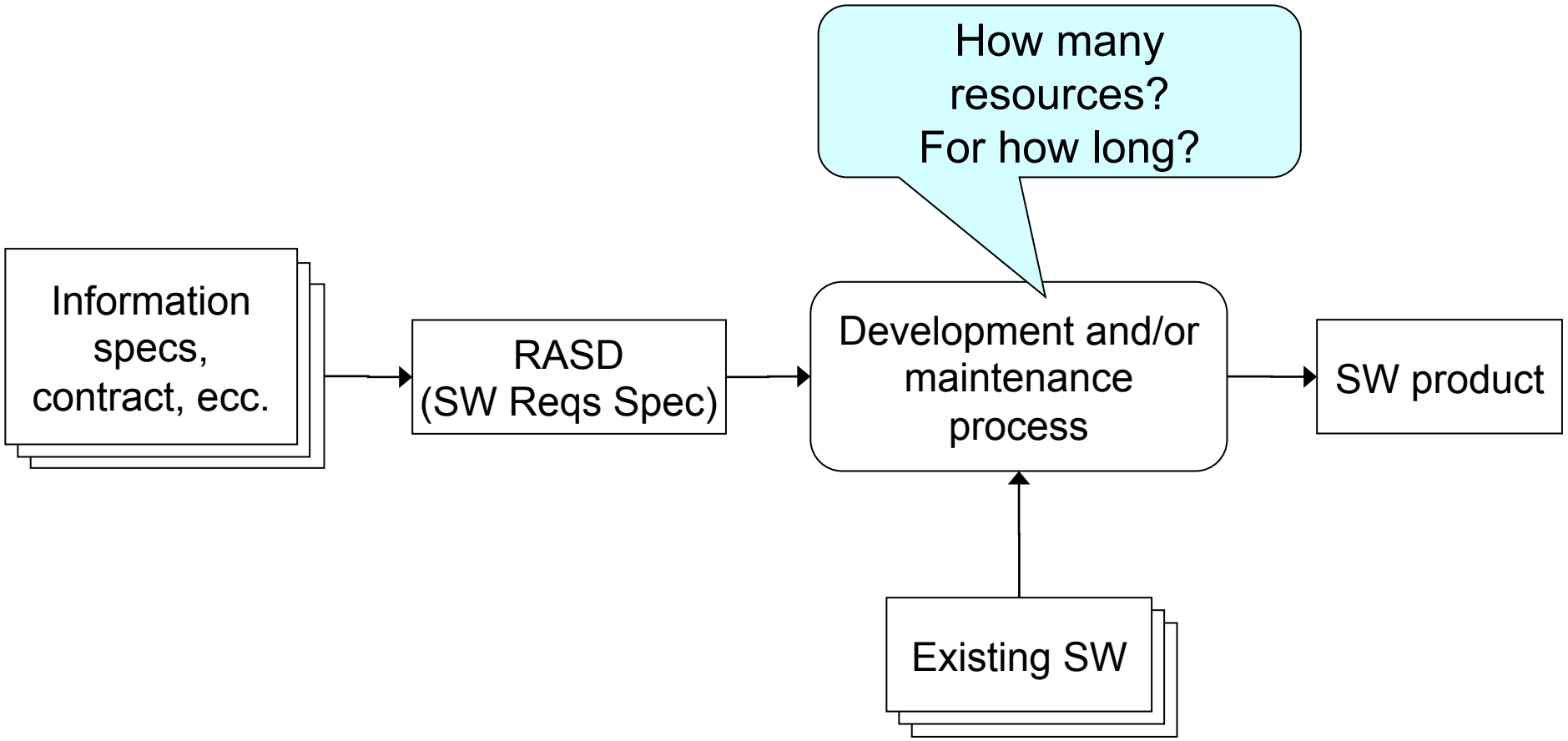
Market opportunity	A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed.	*
Cost estimate uncertainty	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 can 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 smaller than normal profit or break even than to go out of business.	

*Courtesy of I. Sommerville, "Software Engineering, 7th Ed." Ch. 26

Roadmap

- Necessary Recalls
- Software Cost
- **Software Cost, the Estimation Problem**
- Algorithmic Techniques: Function-Points (FP)
- Algorithmic Techniques: COnstructive COst MOdel (COCOMO)

Software Cost, the estimation problem



Estimation techniques

- Organizations need software effort and cost estimates. Two widely known approaches:
  - *Experience-based techniques*
    - estimate based on the manager's experience of past projects and application domain.
    - manager makes an informed judgment of what the effort requirements are likely to be.
  - *Algorithmic cost modeling*
    - formulated approach is used to compute the project effort based on estimates of product attributes(e.g., size, process characteristics, staff experience, etc.)

Experience-based approaches

- Experience-based techniques rely on judgments
- Typically, identify:
  - deliverables to be produced
  - different software components or systems to be developed.
- Document the above in a spreadsheet
- estimate the above individually
- compute the total effort required

- HELP: get a group of people involved in the effort estimation and to ask each member of the group to explain their estimate. → *community-based estimation*

Experience-based approaches: Parkinson's Law

- “The project costs whatever resources are available” [Cit.]
- Advantages: No overspend
- Disadvantages: System is usually unfinished

Experience-based approaches: Parkinson's Law

- “The project costs whatever resources are available” [Cit.]
- Advantages: No overspend
- Disadvantages: System is handed-in (usually) unfinished

Curiously reminiscent of the disease ☺

Algorithmic cost modelling

- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers:

  $$\textit{Effort} = A \times \textit{Size}^{B} \times M, \textit{ where:}$$

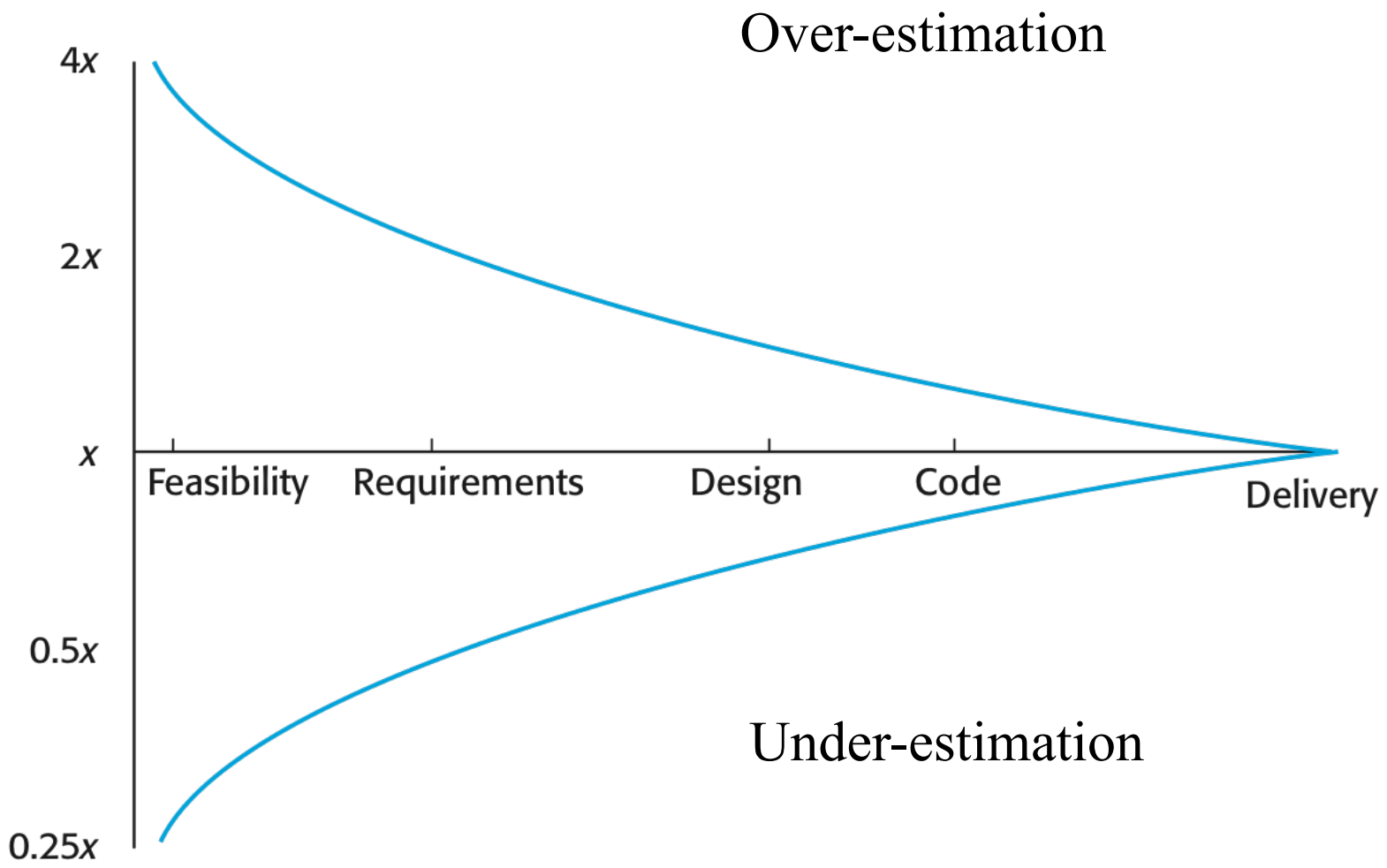
  - ***A*** is an organisation-dependent constant,
  - ***B*** reflects the disproportionate effort for large projects
  - ***M*** is a multiplier reflecting product, process and people.

- The most commonly used product attribute for cost estimation is projected code-size.
- Most algorithmic models are similar but they use different values for A, B and M.

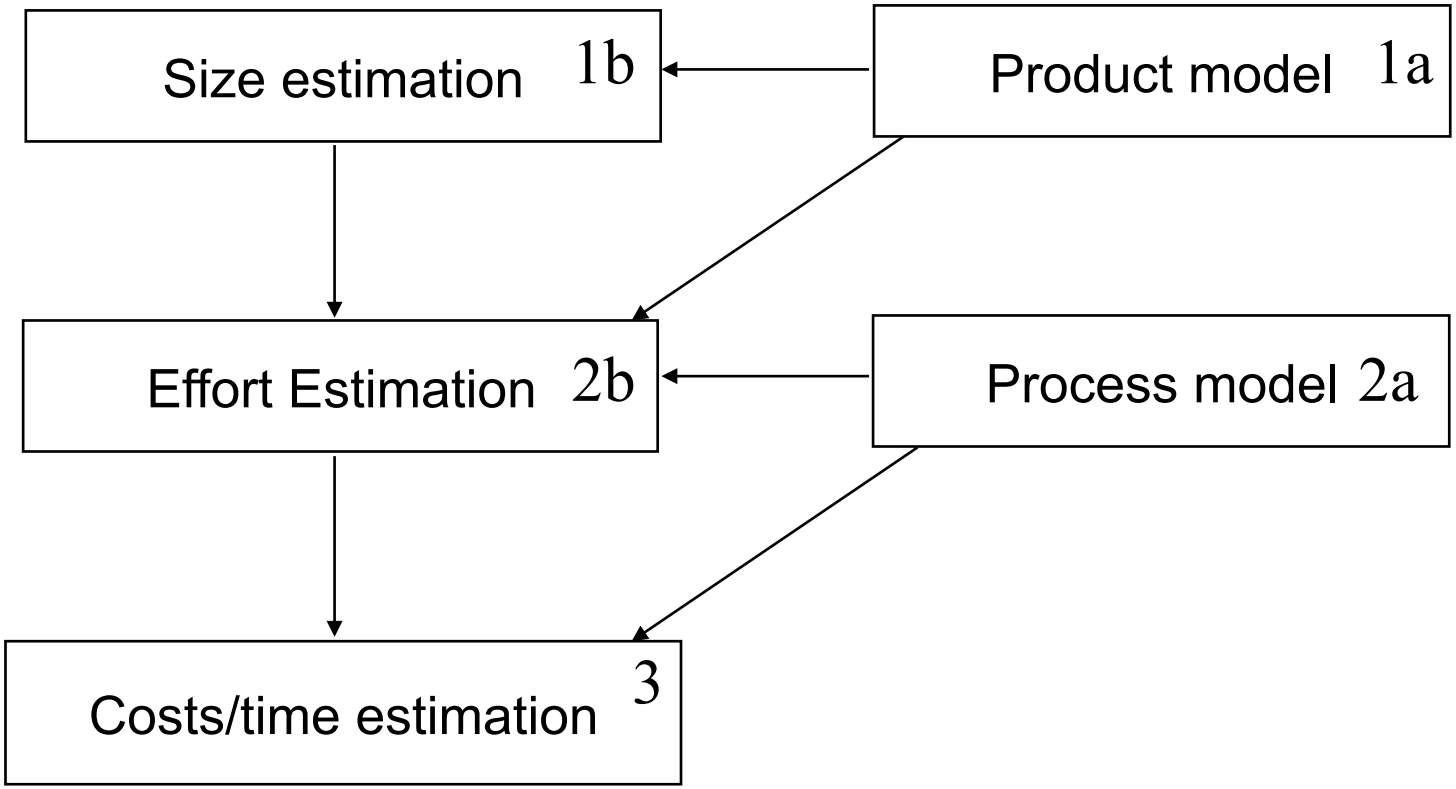
Algorithmic cost modelling: Estimation accuracy

- 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 progresses size estimate is more accurate.
- Estimates for factors acting on B and M are subjective and vary according to the judgment of the estimator.

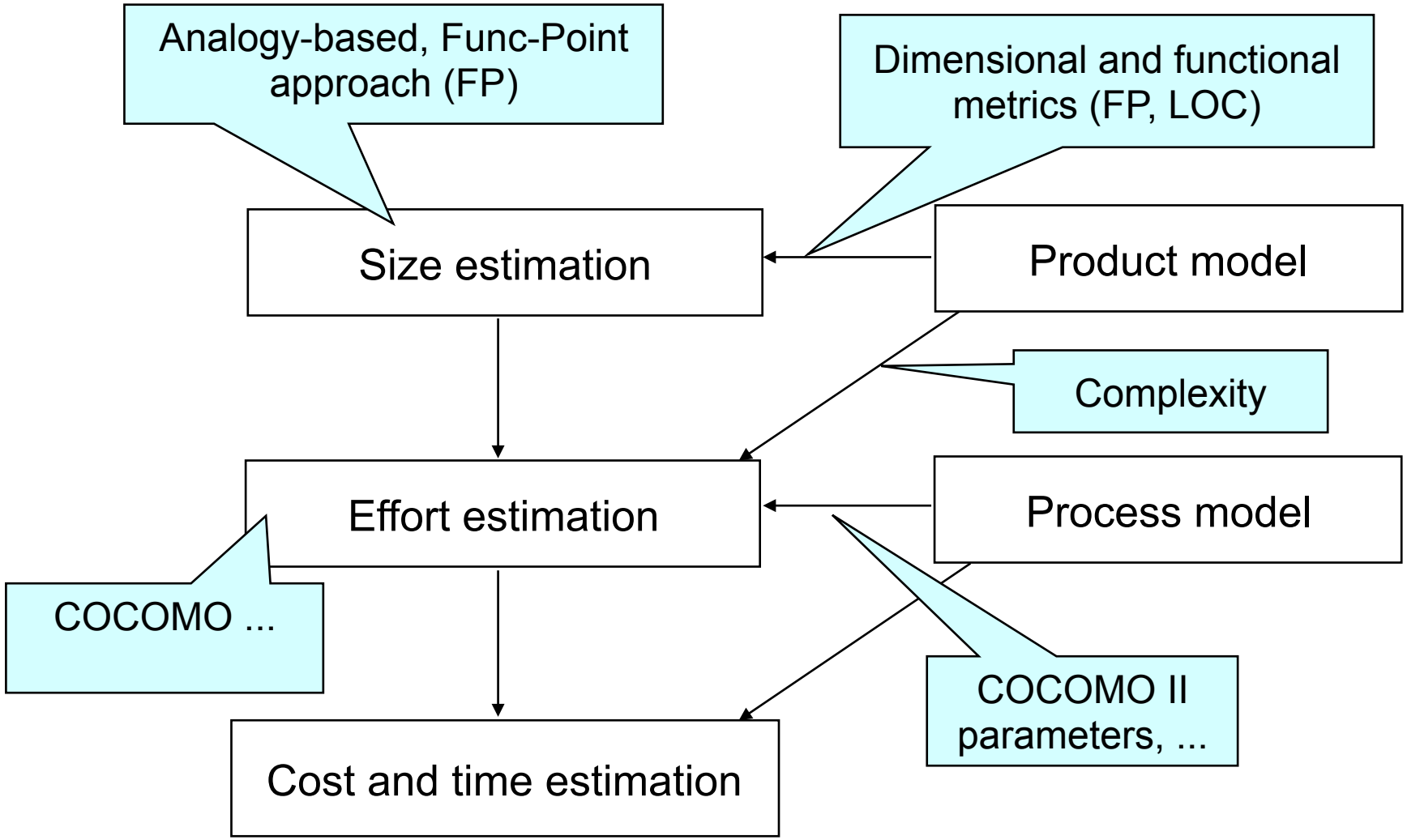
Estimate uncertainty



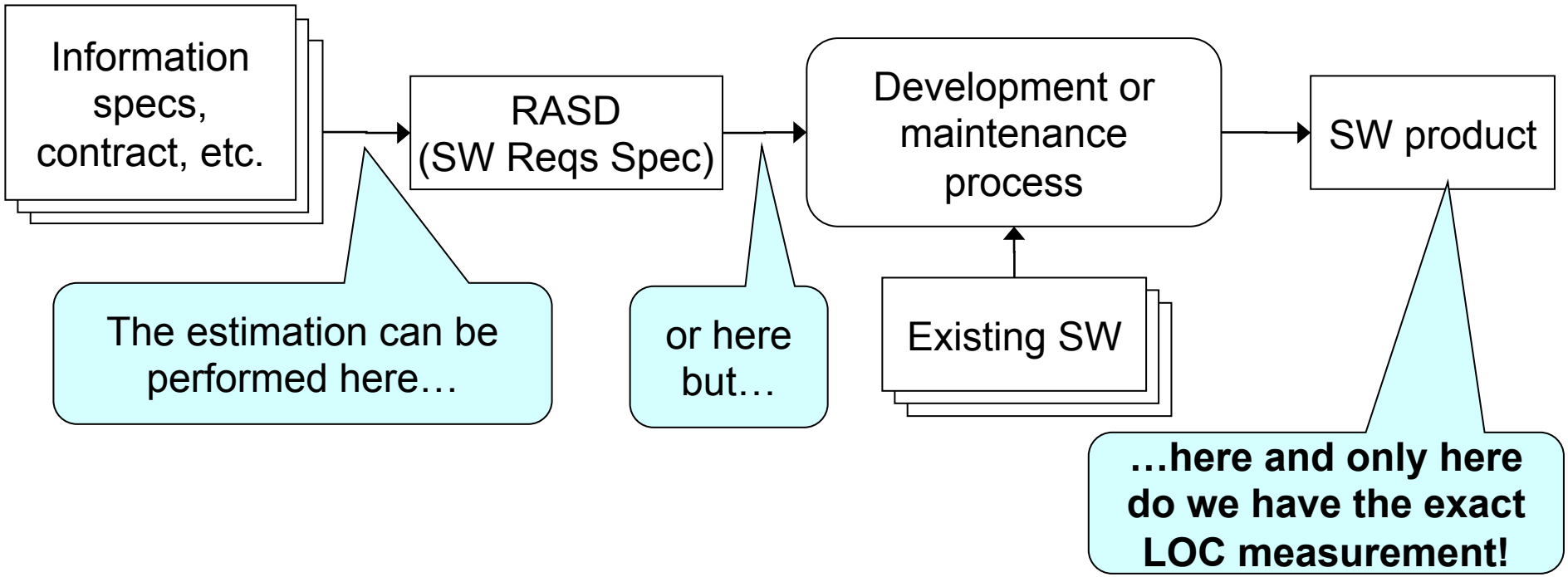
A possible estimation procedure



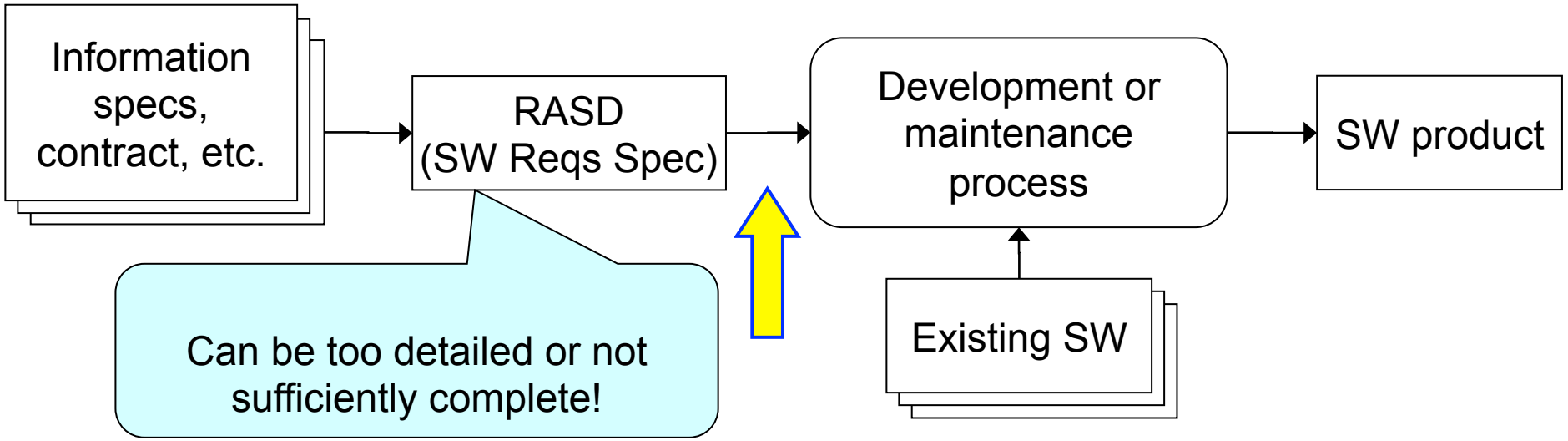
Techniques to support the estimation procedure



Size Estimation



RASD-based size estimation



- Options:
  - Calculate the number of Function Points
    - This can be used directly as a measure of the software dimension
  - Estimate the LOC
    - By converting the FP into LOCs
    - Directly
  - Any estimation has to be complemented with a complexity analysis

Roadmap

- Necessary Recalls
- Software Cost
- Software Cost, the Estimation Problem
- **Algorithmic Techniques: Function-Points (FP)**
- Algorithmic Techniques: COnstructive COst MOdel (COCOMO)

Function-points

- The FP approach was defined in 1975 by Allan Albrecht (IBM)

- Fundamental Assumption:

  “the dimension of software can be characterized by abstraction → functionalities vs. number of lines”

The original definition

*"a technique to assess the effort needed to design and develop custom software applications" **

* A. Albrecht IBM Technical Journal

What are 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 FP counts; the total is computed by multiplying each “raw” count by the weight and summing all partial values:

$$UFC = \sum(\text{number of elements of given type}) \times (\text{weight})$$

FP properties (1)

- The effort to develop a software product depends on its functionalities
  - The “functionality offered to the user” is in relationship with the system inputs and outputs
  - It has to be connected to the requirements expressed by the user
  - It HAS to be calculated in the first phases of the development process
  - It has to be independent of the specific development technology

FP Properties (2)

- The function point count is 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.

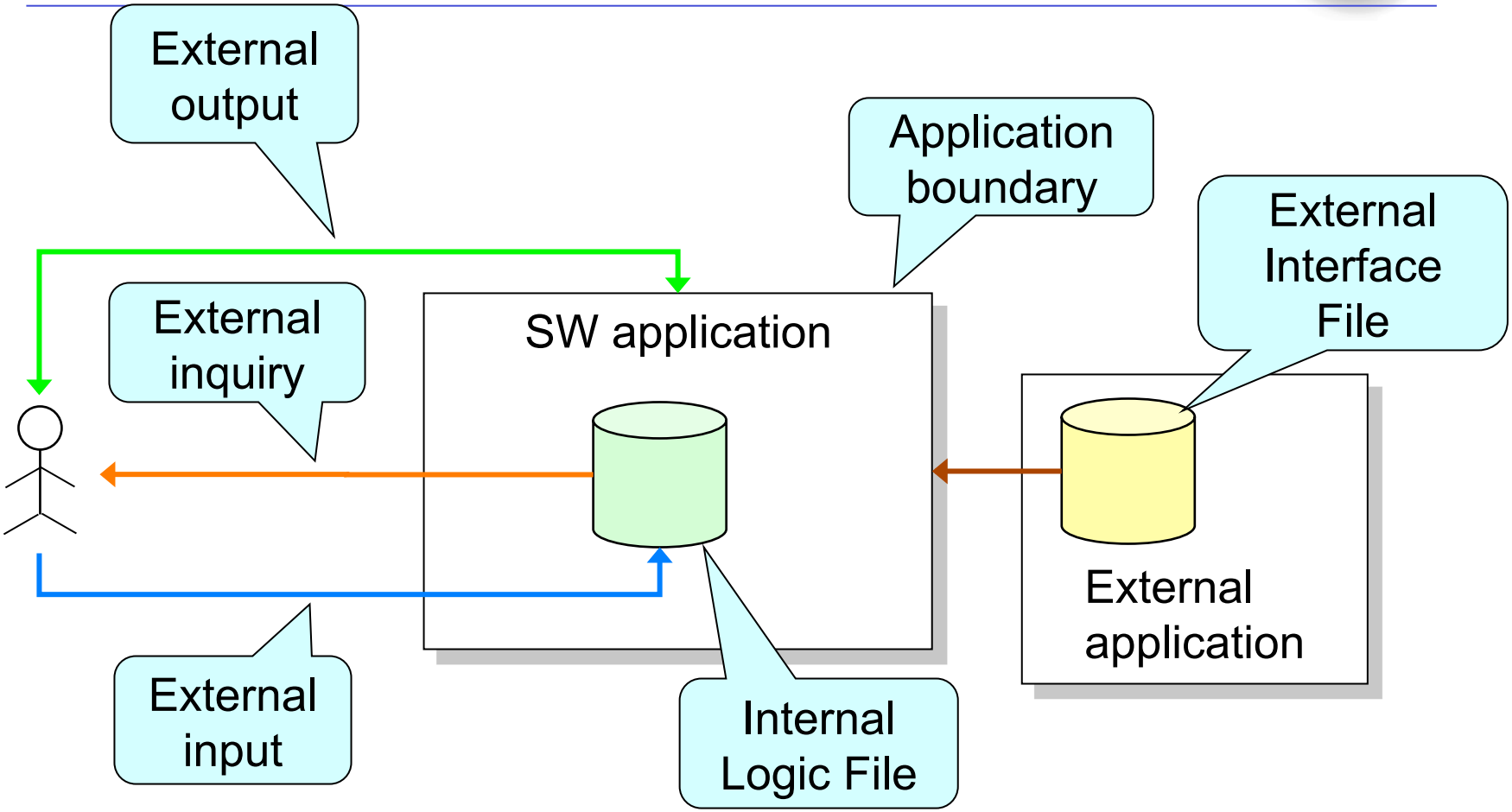
FP Types (1)

- The Albrecht's method identifies and count the number of function types
- They represent the "external representation" of an application, that is, its functionality
- Function types
  - Internal Logical File (ILF): homogeneous set of data used and managed by the application
  - External Interface File (EIF): homogeneous set of data used by the application but generated and maintained by other applications

FP Types (2)

- External Input
  - Elementary operation to elaborate data coming form the external environment
- External Output
  - Elementary operation that generates data for the external environment
    - It usually includes the elaboration of data from logic files
- External Inquiry
  - Elementary operation that involves input and output
    - Without significant elaboration of data from logic files

FP Types (3)


External output
Application boundary
External Interface File
External inquiry
SW application
External application
External input
Internal Logic File

How was the approach defined?

- Albrecht has considered 24 applications
  - 18 COBOL, 4 PL/1, 2 DMS
  - …ranging from 3000 to 318000 LOC,
  - …from 500 to 105200 person hours

- Based on these he has defined the number of FP as the sum of Function Types weighted in order to correlate them to the development effort

Weighting function points

Complexity is evaluated based on the characteristics of the application

Function Types	Weight		
	Simple	Medium	Complex
N. Inputs	3	4	6
N. Outputs	4	5	7
N. Inquiry	3	4	6
N. ILF	7	10	15
N. EIF	5	7	10

- We obtain the Unadjusted Function Points (UFP)

Example

- Calculate FPs for an application managing a mobile telephony service. The application:
  - Manages data about both users and the different types of rates. The system will store the user data (phone number, personal data, the type of rate, any applying promotion). Each rate type is characterized by: call cost, cost of text messages, cost of surfing the Internet. Promotions change some of the cost items only for a well-defined period of time.
  - Allows users to access their information, change rates, and activate promotions.
  - Manages user billing based on information retrieved from the network management system. This last system sends to the service management system data concerning the voice calls (call duration, and the user location in case of roaming), SMSs (the information that messages have been sent, and the corresponding user location in case of roaming), and Internet surfing (connection duration, amount of downloaded data, user location) of each user.

What do we do?

1. Evaluate ILFs (Internal Logic Files)
2. Evaluate EIFs (External Interface Files)
3. Evaluate EIs (External Inputs)
4. Evaluate EIQs (External Inquiries)
5. Evaluate EOs (External Outputs)
6. Compute UFPs (Un-adjusted Function-Points)
7. “Adjust” UFPs

ILFs

- The application stores the information about
  - *users*,
  - *rates*, and
  - *promotions*.
  - As the application also manages billing information, it will have to produce *invoices*.

- Each of these entities has a simple structure as it is composed of a small number of fields. Thus, we can decide to adopt for all the four the simple weight.

- Thus, 4 x 7 = 28 FPs concerning ILFs.

ELFs

- The application manages the interaction with the network management system for acquiring information about
  - *calls*,
  - *SMSs*, and
  - *Internet usage*.
- Three entities with a simple structure. Thus, we decide to adopt a simple weight for the three of them.
- We get 3 x 5 = 15 FPs.

External inputs (1)

- The application interacts with the customer as follows:
  - Login/logout: these are simple operations, so we can adopt the simple weight for them. 2 x 3 = 6 FPs
  - Select a rate: this operation involves two entities, the rate and the user. It can still be considered simple, so, again we adopt the simple weight. 1 x 3 = 3 FPs
  - Select/eliminate a promotion: These two operations involve three entities, the user, the rate and the promotion. As such, we can considered them of medium complexity. 2 x 4 = 8 FPs

External inputs (2)

- The application also interacts with the company operators to allow them to:
  - Insert/delete information about a new rate or promotion.
  - Insert/delete information about a new user.

- Both the above operations can be consider of average complexity. 4 x 4 = 16 FPs

- In summary, we have FPs = 6+3+8+16 = 33

External inquiries

- The application allows customers to request information about
  - their profiles
  - the list of rates, promotions, and invoices that have been defined for them.
- The application also allows the operator to visualize the information of all customers.
- In summary, we have 5 different external inquiries that we can consider of medium complexity. Thus, 5 x 4 = 20 FPs.

External outputs

- The application allows the creation of invoices.
- This is a quite complex operation as it needs to collect information from IFLs and EFLs.
- Thus, we can apply the weight for the complex cases: FP = 1 x 7.

Total number of FPs

- ILFs: 28
- ELFs: 15
- External Inputs: 33
- External Inquiries: 20
- External Outputs: 7

- Total Estimate: 103

Adjusted Function Points

- A final correction [+/-35%] accounts for 14 params F_i
- The final result (UFP) can be used as an independent variable to obtain the effort estimate (FP)

$$FP = UFP \times \left(0.65 + 0.01 \times \sum_{i=1}^{14} F_i \right)$$

- Examples of F_i
  - "Does the system require reliable recovery and backup procedures?"
  - "Is [remote] data transmission required?"

UFP vs FP

- Correcting UFP is not necessarily a good idea
- It is an attempt to adapt a metric that is focusing on functionality to the prediction of development costs
  - “It is like correcting the tallness of a person to relate it to a measure of his/her intelligence” [Cit. Fenton]
  - Correction does not improve precision of the estimation

UFP vs FP

- Correcting UFP is not necessarily a good idea
- It is an attempt to adapt a metric that is focusing on functionality to the prediction of development costs
  - "It is like correcting the tallness of a person to relate it to a measure of his/her intelligence" [Cit. Fenton]
  - Correction does not improve precision of the estimation
- **MUCH BETTER:** *We can use UFP in combination with other approaches, e.g., COCOMO, instead of FP!* ☺

The International Function Point Users Group

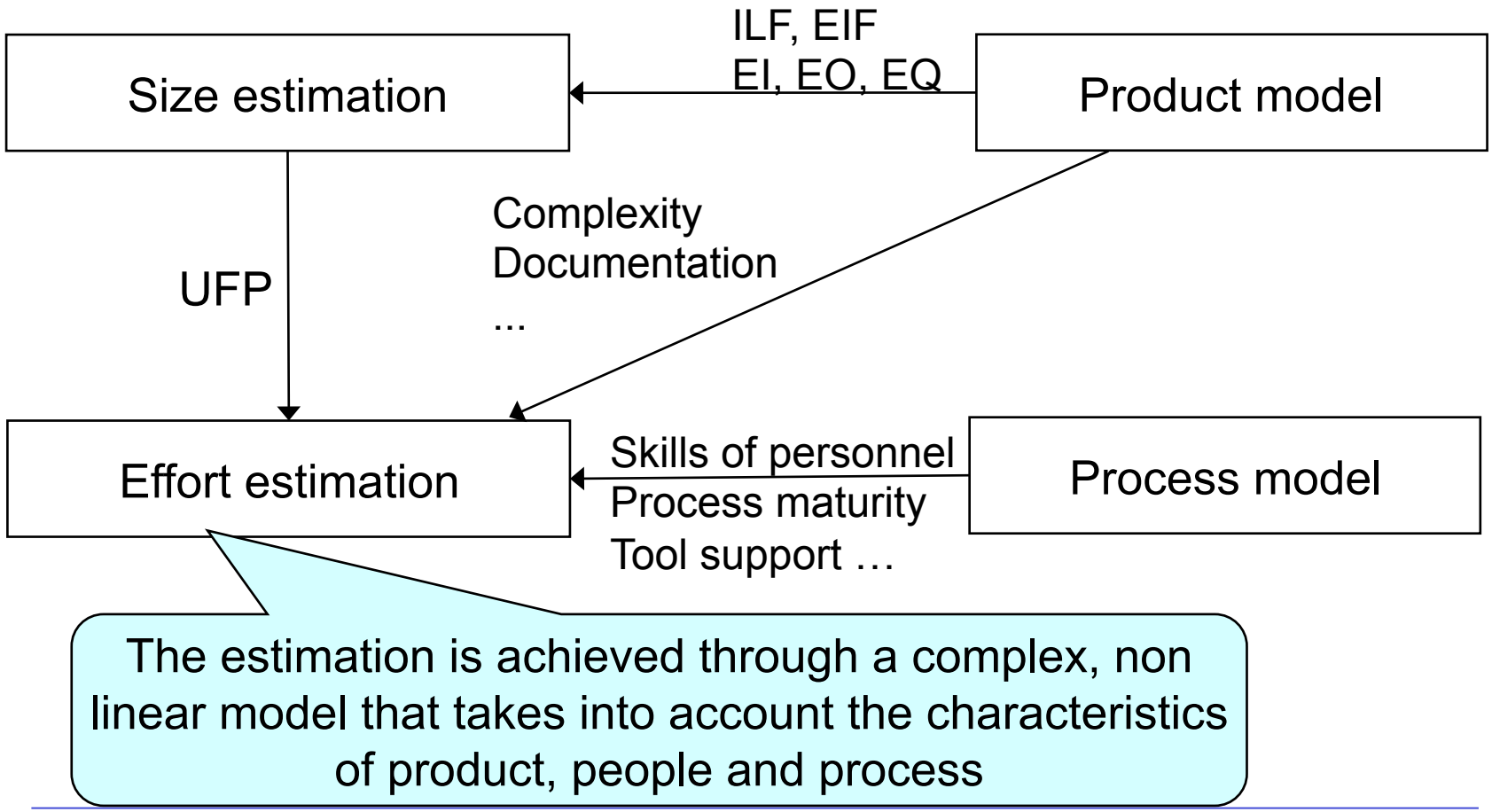
- Mission: takes care of the evolution of FPs
- Produces a manual for supporting the adoption of the FP approach
- Defines various versions of FP to apply them outside the initial context
- Offers examples useful as a reference (http://www.ifpug.org/)

Any Questions?

Roadmap

- Necessary Recalls
- Software Cost
- Software Cost, the Estimation Problem
- Algorithmic Techniques: Function-Points (FP)
- **Algorithmic Techniques: COnstructive COst MOdel (COCOMO)**

COCOMO: positioning

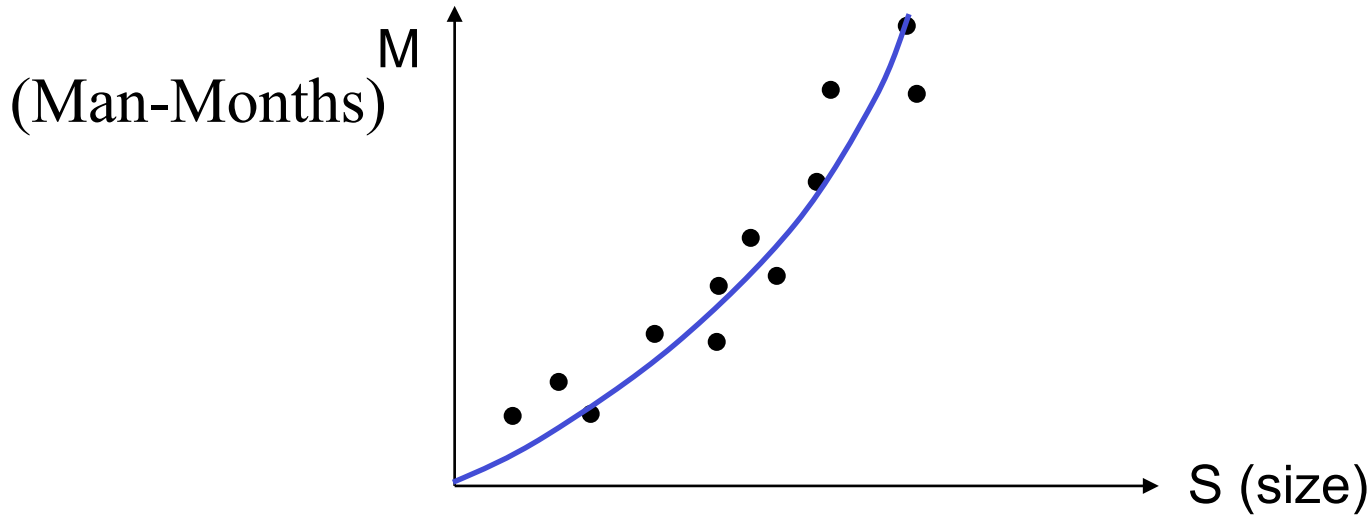


COCOMO: History and structure

- Developed around the ’80
- Based on the statistical analysis of 63 real projects from various domains;
- The original database is now periodically updated and enriched with data of new projects
- The model is updated also through the definition of new cost coefficients
- From 1997 onwards, we have COCOMO II

How was the approach defined?

- Barry Boehm has adopted a statistical approach to calculate its formula
- Given the size and effort of a number of homogeneous projects (same complexity) Boehm has been looking for the best correlation... What he found is that:
  - “a linear regression model applied to the logarithms of variables involved has offered the best results”*



* “ CoCoMo-based cost estimations ” B. Bohem

Basic hypotheses of COCOMO 81 (1)

- The development process is assumed to follow the traditional waterfall scheme
- Requirements are considered mainly stable for all the project duration
- The architectural design is accomplished by a small high quality team deeply knowledgeable about the application problem

Basic hypotheses of COCOMO 81 (2)

- Detailed design, development and unit test are performed in parallel on different sub-systems by different teams of programmers
- Documentation is written incrementally during the project lifetime
- Modern project management techniques are used throughout the project.

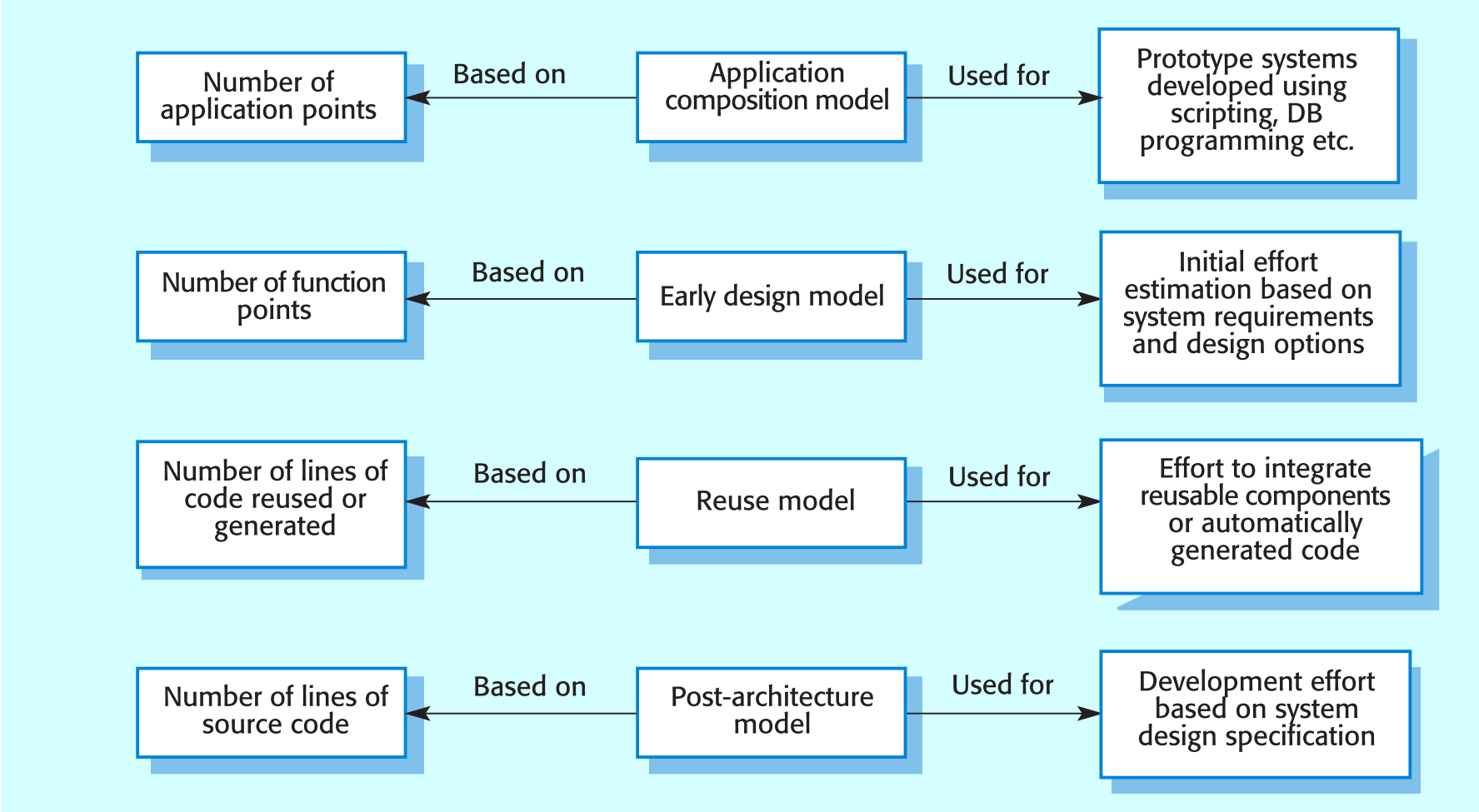
From COCOMO 81 to COCOMO II

- Eventually researchers found out that the hypotheses behind COCOMO 81 were overly optimistic and not reflecting the state of the art
  - COCOMO 81 was refined and integrated in COCOMO II

- most fundamental calculation in COCOMO II is the use of the Effort Equation to estimate the number of Staff-Months required to develop a project.

- Other COCOMO II results (e.g., estimates for Requirements and Maintenance) are derived from this quantity.

COCOMO II models



How Does COCOMO II work?

Elements of the COCOMO II model:

- Source Lines of Code (SLOC)
- Scale Drivers
- Cost Drivers
- The Effort Equation
- The Effort Adjustment Factor
- The Schedule Equation
- The SCED (Schedule Constraints) Cost Driver

How Does COCOMO II work?

Elements of the COCOMO II model:

- **Source Lines of Code (SLOC)**
- **Scale Drivers**
- **Cost Drivers**
- **The Effort Equation**
- **The Effort Adjustment Factor**
- **The Schedule Equation**
- The SCED (Schedule Constraints) Cost Driver

How Does COCOMO II work? SLOCs

The COCOMO II calculations are based on estimates of a project's size in Source Lines of Code (SLOC). SLOC is defined such that:

- Only Source lines that are DELIVERED as part of the product are included – e.g., test-drivers and other support software is excluded
- SOURCE lines are created by the project staff -- code created by applications generators is excluded
- One SLOC is one logical line of code
- Declarations are counted as SLOC
- Comments are not counted as SLOC

How Does COCOMO II work? Scale Drivers

In COCOMO II, some of the most important factors contributing to a project's duration and cost are the Scale Drivers (SD).

SDs determine the *exponent* used in the Effort Equation.

5 Scale Drivers are defined:

- Precedentedness
- Development Flexibility
- Architecture / Risk Resolution
- Team Cohesion
- Process Maturity

How Does COCOMO II work? Scale Drivers (2)

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.

How Does COCOMO II work? Cost Drivers

- COCOMO II has 17 cost drivers, i.e., multiplicative factors that determine the effort required to complete the software project.

- COCOMO II defines each of the cost drivers, and the Effort Multiplier associated with each rating.

*For example, if your project will develop **software for airplanes**, you would likely use the **Required Software Reliability (RELY)** cost driver to Very High. This means an effort multiplier of 1.26, i.e., your project will require 26% more effort than a typical project.*

How Does COCOMO II work? Cost Drivers (2)

Cost Driver Examples:

ACAP Analyst Capability
APEX } Applications Experience
AEXP }
PCAP Programmer Capability
LEXP Programming Language Experience
VEXP Virtual Machine Experience
PERS Personnel Capability
. . .

How Does COCOMO II work?
The effort-equation

COCOMO II model makes its estimates of required effort (measured in Person-Months – PM) based primarily on ***your*** estimate of the software project's size (as measured in thousands of SLOC - KSLOC) :

$$€ffort = 2.94 * EAF * (KSLOC)^{E}$$

Where:

EAF → Effort Adjustment Factor derived from Cost Drivers (product of the effort multipliers corresponding to each of the cost drivers for your project)

E → Exponent derived from Scale Drivers

How Does COCOMO II work?
The effort-equation

For example:

Consider a project with all “Nominal” (i.e., normal) Cost Drivers and Scale Drivers would have an EAF of 1.00 and exponent, E, of 1.0997.

Assuming that the project is expected to consist of 8,000 source lines of code, COCOMO II estimates that 28.9 Person-Months of effort is required to complete it:

€effort = 2.94 * (1.0) * (8)1.0997 = 28.9 Person-Months

How Does COCOMO II work? The effort-equation

For example:

Consider a project with all "Nominal" (i.e., normal) Cost Drivers and Scale Drivers would have an EAF of 1.00 and exponent, E, of 1.0997.

Assuming that the project is expected to consist of 8,000 source lines of code, COCOMO II estimates that 28.9 Person-Months of effort is required to complete it:

€effort = 2.94 * (1.0) * (8)1.0997 = 28.9 Person-Months

IMHO: I would say it's more likely to be 36!

How Does COCOMO II work?
The schedule-equation

COCOMO II schedule equation predicts the number of months required to complete a software project. The duration of a project is based on the effort predicted by the effort equation:

$$\text{Duration} = 3.67 * (€\text{ffort})^{E}$$

Where

€ffort → Is the effort from the COCOMO II effort equation

SE → Is the schedule equation exponent derived from the five Scale Drivers

Finally, dimensioning the community

- Given the values of effort “€” and duration “D” for the project based on COCOMO II, the number of required people “N” is:
  - N_{people} = € / D;

Finally, dimensioning the community... An example

Continuing the previous example:

let's substitute the exponent with 0.3179 (calculated from new scale drivers for schedule), COCOMO II yields an estimate of just over a year duration, and an average staffing of between 3 and 4 people:

Duration = 3.67 * (42.3)0.3179 = 12.1 months

Team/community size = (42.3 Person-Months) / (12.1 Months) = 3.5 people ➔ **4 people**

COCOMO precision

- By comparing COCOMO estimations with the actual execution of some projects the following statistics have been generated:
  - In 25% of cases, Basic COCOMO offers estimations with variations of 20% or less.
  - In 68% of cases, Intermediate COCOMO offers estimations with variations of 20% or less.
  - In 70% of cases, Advanced COCOMO offers estimations with variations of 20% or less.

COCOMO: tricks of the trade

- Never forget the statistical nature of COCOMO!
- When we apply it and get an estimation of effort, the resulting number is based on the knowledge of the current practice encapsulated in the model
- Never Forget that “waterfall” and a number of nasty pitfalls were (and to some degree still are) fundamental assumptions behind COCOMO! This biases COCOMO tables and coefficients
- If the project is similar to the ones that have been considered for the definition of the model, then the computed effort is likely to be correct
  - Calibrate the model!

Effort Estimation: tricks of the trade

- There is **no simple relation between development and cost prices.**
- Factors affecting productivity include individual aptitude, domain experience, the development project, the project size, tool support and the working environment.
- Software IS priced to gain a contract → **functionality is adjusted to price.**
- Different estimation techniques should be used together, **for reasoning.**
- The COCOMO is a rather complete model (takes project, product, personnel and hardware attributes into account when predicting effort).
- Algorithmic cost models support **quantitative option analysis**, i.e., compute and evaluate different options!
- **Time-to-completion is not proportional to # people!** **[Cit. Moore's Law]**

Any Questions?