Software Effort Estimation Part 1

Lecture 5 by Professor Vladimir Geroimenko

Module “Software Project Management“

23 October 2016 - Teaching Week 5

Textbook reference: Chapter 5

Lecture Outline

- Introduction to Software Effort Estimation
- Estimation Approaches
  - Top-down estimation
  - Bottom-up estimation
- Estimation Technique
  - Expert judgment
  - Estimation by analogy
  - Algorithmic models
    - Albrecht function points
    - Mark II function points
    - **COCOMO 81 / COCOMO II – Lecture Part 2 next week**

What makes a successful project?

- Delivered on time
- At the agreed cost
- With agreed functionality
- with the required quality

To achieve this and to meet all the deadlines and targets,

realistic estimates are crucial

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

What to estimate in a software project?

Software size

- *SLOC* (Source Lines of Code)
- *KLOC* (Thousands of Lines of Code)

Effort

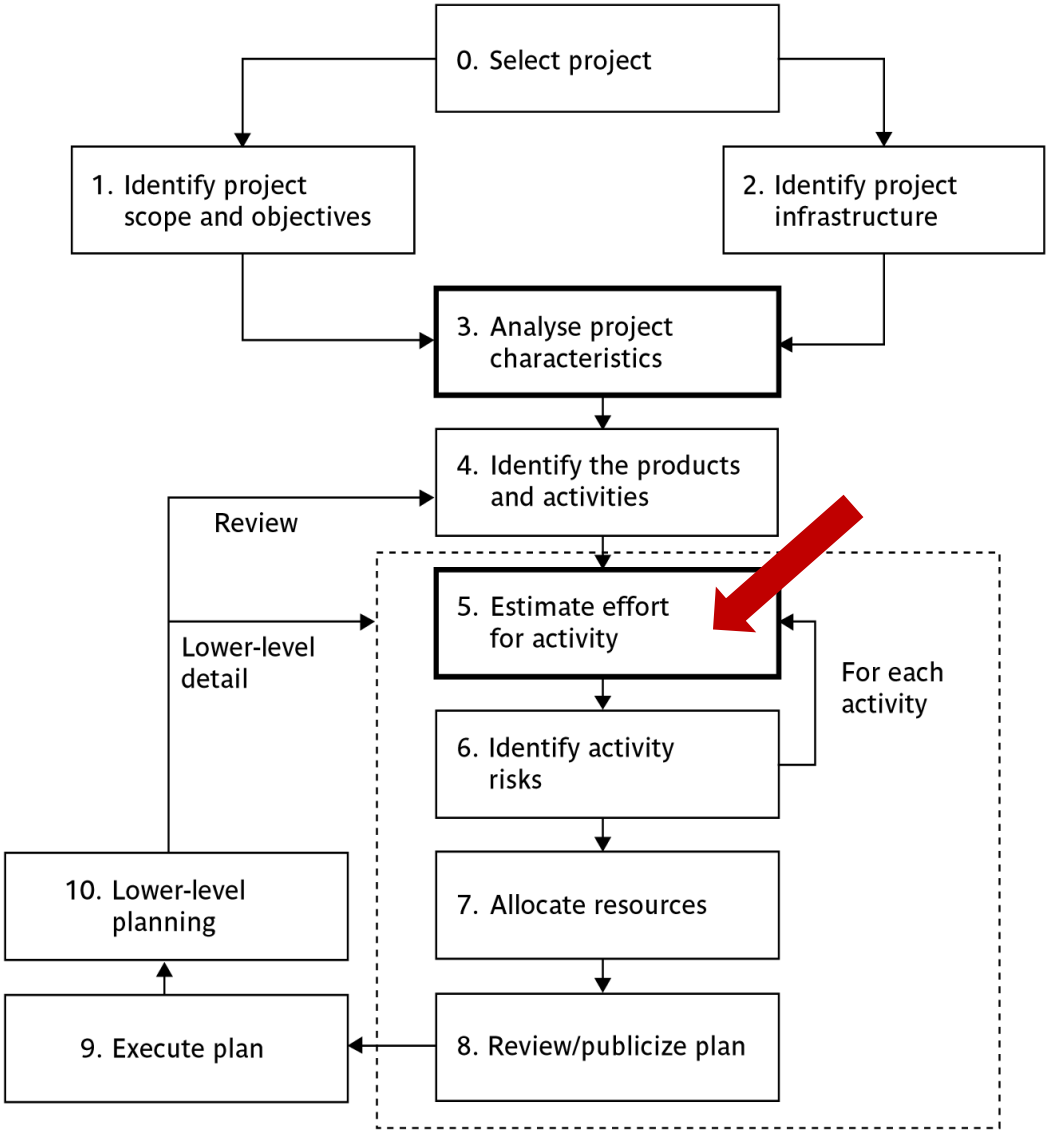
- Person-**hours**
- Please note: Number of hours in person-month can be different in different countries

Schedule

- Calendar-**days, weeks or months**

Where we are now in SPMM?

Stepwise Project Management Method (Described in detail in Lecture 2)



Source Lines Of Code - Calculations

- Line = statement
- Line = data definition
- Line ≠ comments
- Line ≠ debug code
- Line ≠ test cases
- Line ≠ automatically generated code
- *Please note: Prerequisite for SLOC calculations: WBS*

- Advantages:
  - Simple measure
- Disadvantages:
  - Different languages - Use tables: C = 2.5, Fortran = 3, Java = 6, …
  - How can we do all of this prior to coding?
  - Bad design may lead to excessive lines of code
  - No account for functionality or complexity

Where are estimations done?

Estimations are carried out during different stages of software project:

- Strategic planning
- Feasibility study
- System specification
- Evaluation of suppliers' proposals
- Project planning

Please note: The accuracy of the estimates improves as the project proceeds.

Over- and under-estimating

Over-estimating

Generous estimates tend to lead to reductions in productivity.

- Parkinson's Law: '*Work expands to fill the time available*'
- Brook's Law: '*Putting more people on a late job, makes it later*'

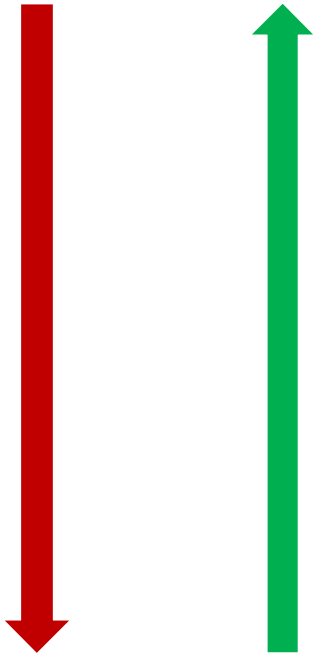
Under-estimating

Aggressive targets in order to increase productivity could lead to poorer product quality.

- Weinberg's Zeroth Law: '*If the system doesn't have to work, we can meet any other constraints*'

Estimation Approaches

- Top-down estimation
- Bottom-up estimation



Top-down Estimation

Based on either WBS or PBS

- Start at the highest level
- Produce estimate for the overall project
- Allocate proportions of an effort estimate to different activities/products of the project

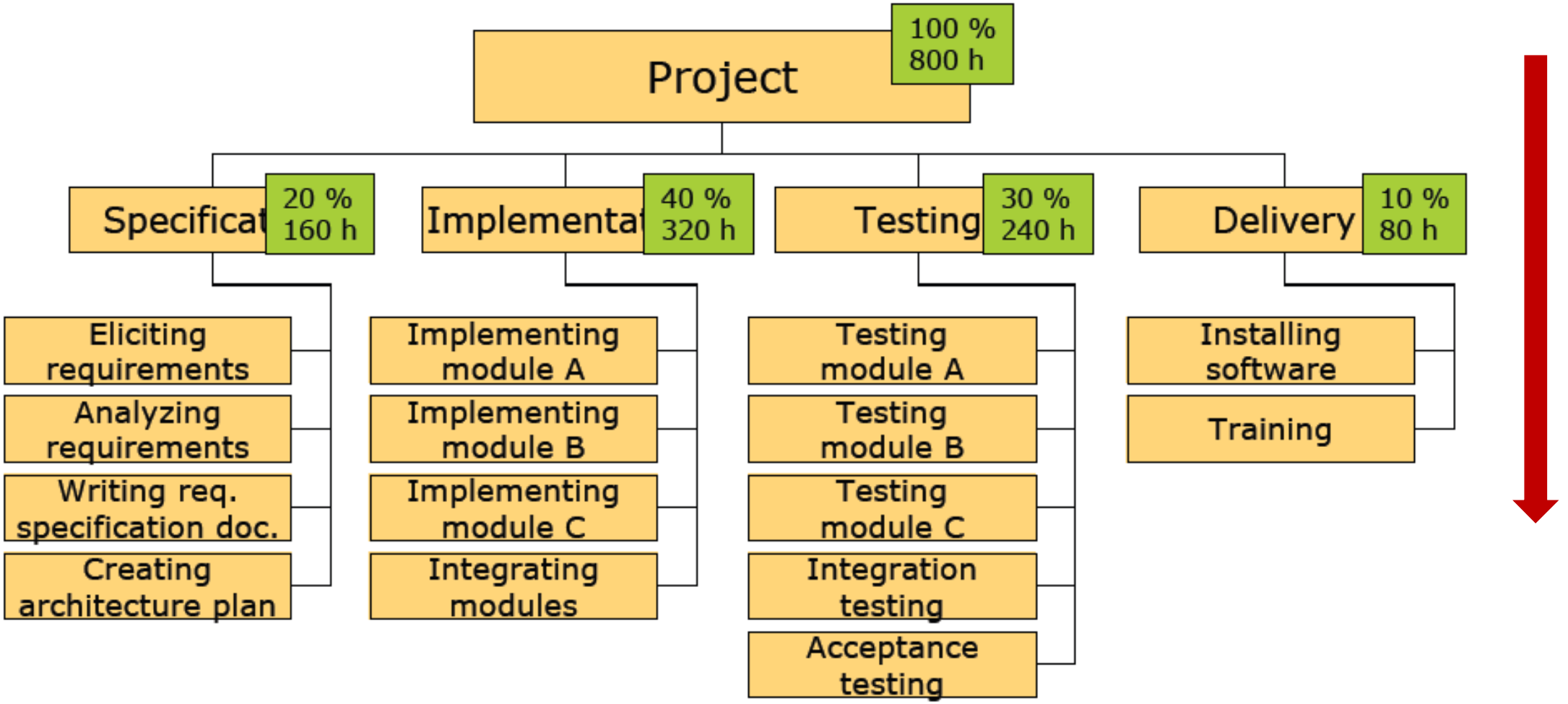
Advantages

- Takes into account integration, configuration management and documentation costs

Disadvantages

- Underestimating the difficult low-level technical problems

Top-down Estimation: Example



Bottom-up Estimation

Based on either WBS or PBS

- Start at lowest level tasks
- Estimate costs for the lowest level activities
- At each higher level aggregate estimate by adding estimates for lower levels

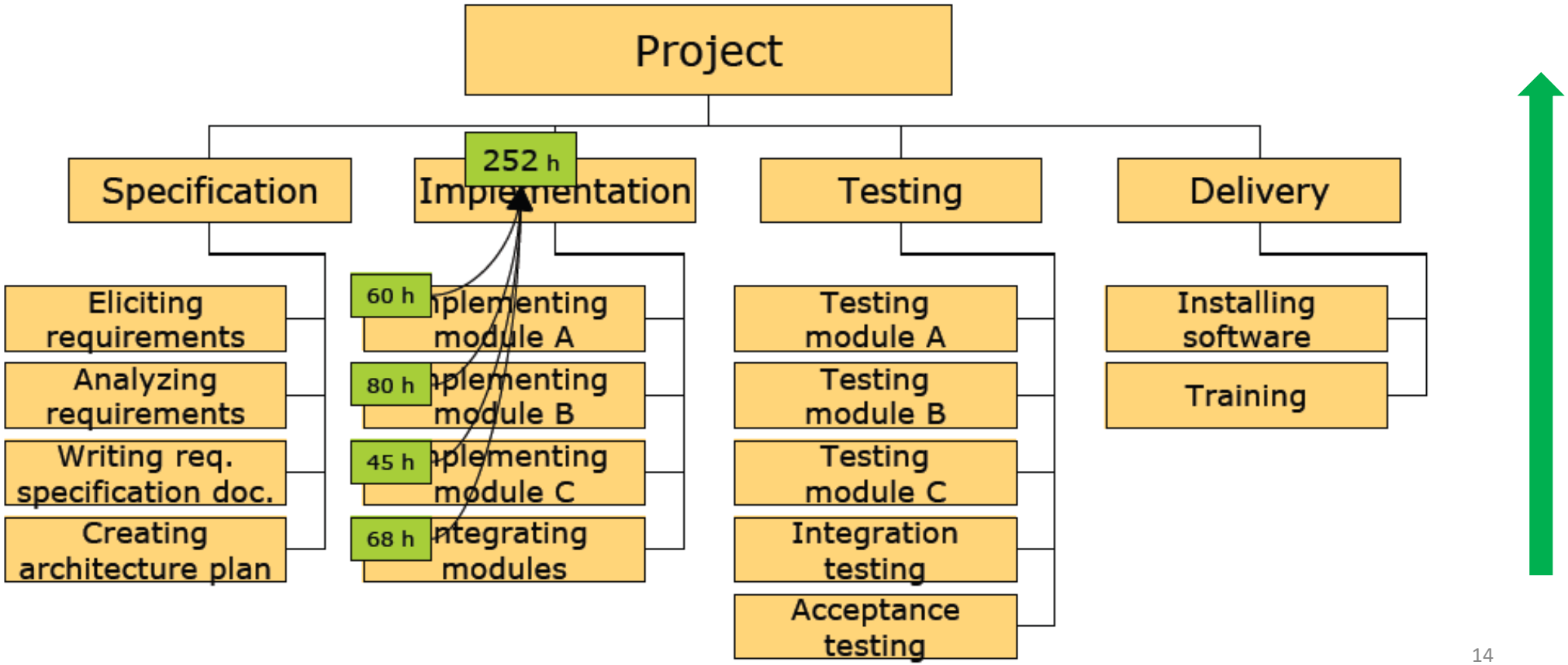
Advantages

- If the task division is accurate, the model may provide the most accurate estimates

Disadvantages

- Underestimates costs of system level (e.g. integration and documentation)
- Needs detailed breakdown of project tasks

Bottom-up Estimation: Example



Bottom-up versus Top-down

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

- 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

Estimation Techniques

- Expert judgment
- Estimation by analogy
- Algorithmic models
  - Albrecht function points
  - Mark II function points
  - COCOMO 81 / COCOMO II (Next week's lecture)

Expert judgment

Asking for required estimate from persons that have previous experience on some parts of the problem

Advantages

- Cheap, fast, and easy to adapt to changing situations
- More accurate than algorithmic methods

Disadvantages

- The evaluation criteria cannot be easily made visible
- How to distinguish good estimates from bad ones?
- Underestimation of small tasks and overestimating large
- Can be strongly biased by information that is irrelevant for the estimates

Estimation by Analogy

Finding similar cases to the target project

Example: calculate Euclidean distance

Example software: ANGEL tool

Advantages

- If done systematically, the estimate can be justified to project stakeholders
- If a good analogy is found, the estimation can be fairly accurate

Disadvantages

- By definition, no project is exactly equivalent to any other
- There may not be an analogous project in the project data set
- Choosing the right features for finding analogous project

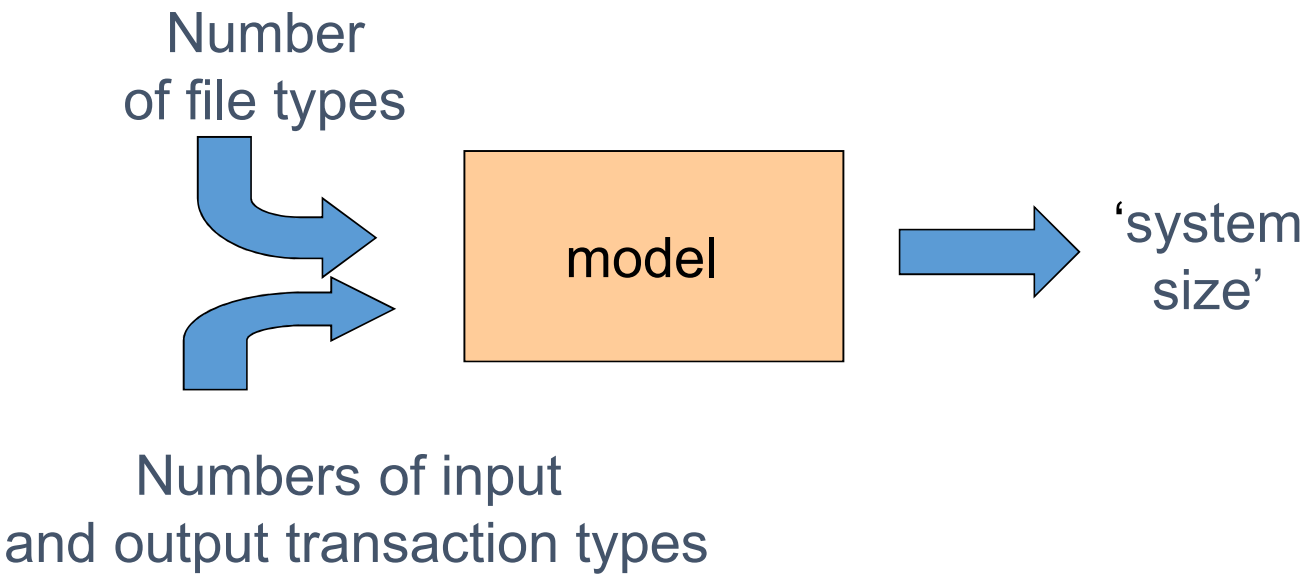
Algorithmic Models

Based on an algorithm (procedure of calculation)

- **Function Point** Methods are used to estimate software ***SIZE***
- **Constructive Cost Models (COCOMO)** is used to estimate ***EFFORT***

Function Point Model

Function Points were originally used to estimate Lines of Code (= System Size), rather than effort.



Function Point Analysis

- Uses the functionality of the software as a measure of its complexity.
- Analogy: For a given house, we can say how many square meters it has or we can say how many bedrooms and bathrooms it has.
- The idea was first put forward by Allan Albrecht of IBM in 1979.
- Works best for traditional applications; not very suitable for algorithmically intensive apps, e.g. real-time and embedded applications

Albrecht 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**

Function Point Analysis (Albrecht, 1993)

For quantifying the functional size of programs independent of its coding language, he

Identifies five main components:

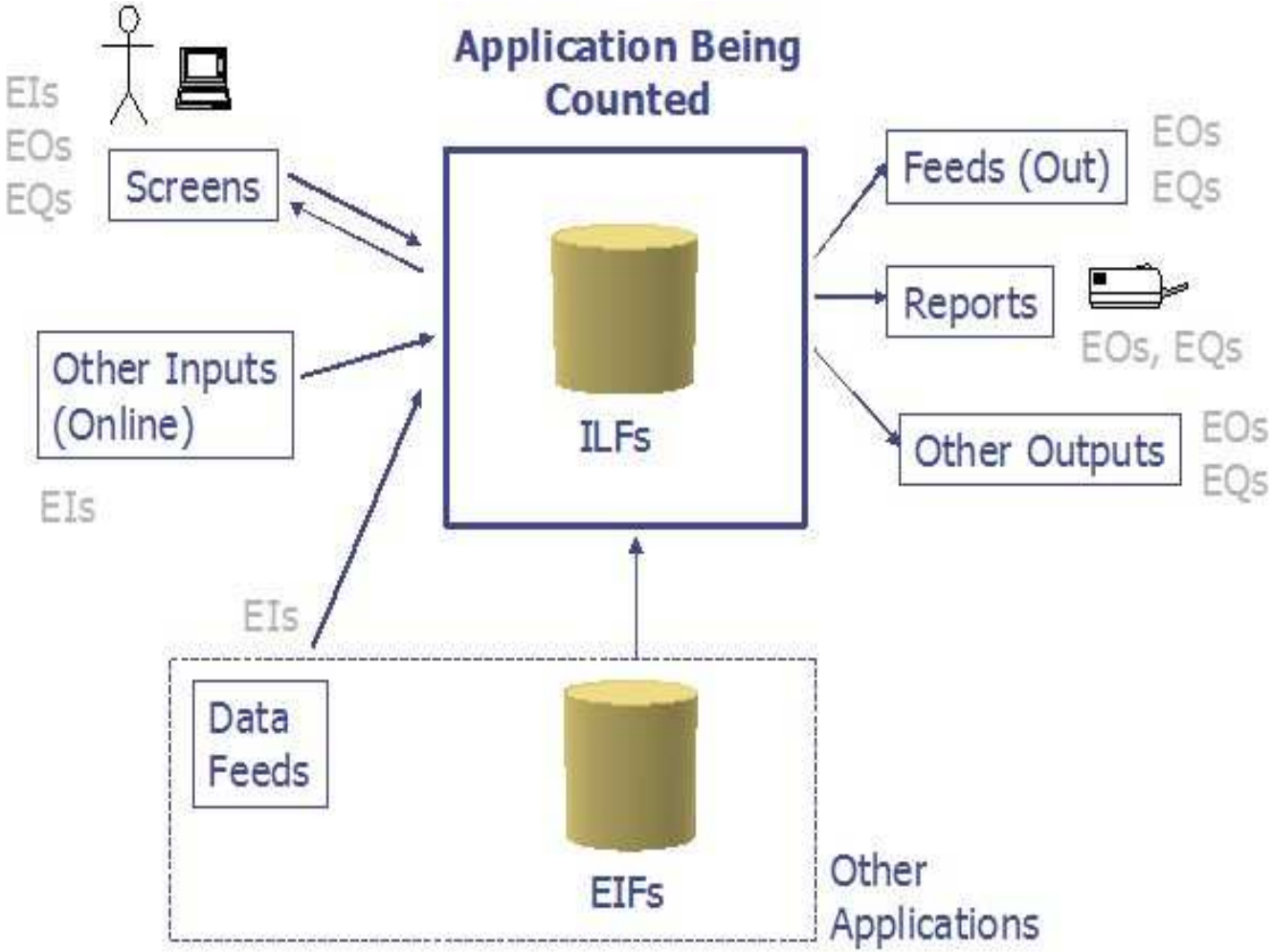
1. External input type: input transactions from user that update internal files (Input Forma)
2. External output type: transactions that output data to user (Reports)
3. Logical internal file type: data storage used by the system (DB Files)
4. External interface file type: input/output between different computer systems (files shared with other software systems)
5. External inquiry type: transactions initiated by users that do not update the internal files (DB Query)

Function Points

- External Input Type = Input
  Keyboard data, touch screen, … (update internal files)
- External Output Type = Output
  Analysis reports, error message, screen output, data report, …
- Logical Internal File = Files
  Data store, relational tables, entity types
- External Interface File = Interfaces
  Data shared with other software systems
- External Enquiry Type = Inquiry (Input / Output)
  User enquiry followed by data output that does not affect internal files

Function Points

A. Data Functions:
   1. Internal logical files
   2. External interface files
B. Transactional Functions:
   3. External Inputs
   4. External Outputs
   5. External Inquiries



Function Point Analysis – Main Steps

1. Count # of components per each category
   - Categories: outputs, inputs, db inquiries, files or data structures, and interfaces
2. Classify components by their Complexity Level
   - Low, Average, or High
3. Find the complexity multiplier for each category

	Multiplier		
External user type	**Low**	**Average**	**High**
External Input	3	4	6
External Output	4	5	7
Logical Internal File	7	10	15
External Interface File	5	7	10
External Inquiry	3	4	6

Albrecht complexity multipliers

Function Point Analysis – Main Steps

4. Multiply number of components for each category by their complexity multiplier.
5. Sum up to obtain the overall FP that indicates the size of the system.

But… how can we identify the complexity level of each component?

- Original FP by Albrecht: Intuitive
- International FP User Guide (IFPUG) formulated rules on how to assess the complexity for each component type

Function Point Analysis - IFPUG

The IFPUG's tables are used to judge the complexity level of an external user type.

Table 1. IFPUG Internal File / Interface complexity

	Number of data types		
Number of record types	**1-19**	**20-50**	**51-**
1	Low	Low	Average
2-5	Low	Average	High
6-	Average	High	High

Table 2. IFPUG External input complexity

	Number of data types		
Number of file types	**1-4**	**5-15**	**16-**
0-1	Low	Low	Average
2	Low	Average	High
3-	Average	High	High

Table 3. IFPUG External Output complexity

	Number of data types		
Number of file types	**1-5**	**6-19**	**20-**
0-1	Low	Low	Average
2-3	Low	Average	High
3-	Average	High	High

Example: Albrecht Function Points

- External input types
  - 2 with High complexity
  - 3 with Average complexity
- External output types
  - 1 with Low complexity
- Logical internal files
  - 4 with High complexity
- External interface types
  - 2 with Average complexity
- External inquiry types
  - None

Example: Albrecht Function points

Tables have been calculated to convert the FPs to Lines of Code for various languages:

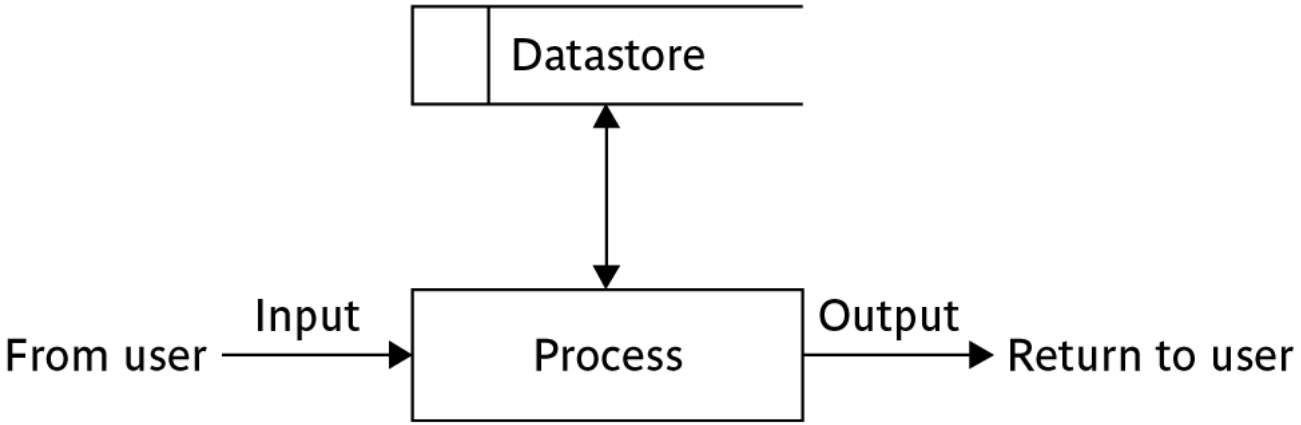
- 60 lines of Java code are needed on average to implement one FP
- For C++ the figure is 34

Type of Component	Complexity of Components			
	Low	Average	High	Total
External input types	---- x 3 =	**3** x 4 = 12	**2** x 6 = 12	**24**
External output types	**1** x 4 = **4**	---- x 5 =	---- x 7 =	**4**
Logical internal files	---- x 7 =	---- x 10 =	**4** x 15 = **60**	**60**
External interface types	**2** x 5 = **10**	---- x 7 =	---- x 10 =	**10**
External inquiry types	---- x 3 =	---- x 4 =	---- x 6 =	**zero**
		Total Number of Function Points		**98**

Function Points - Mark II

- Developed by Charles R. Symons
  - 'Software sizing and estimating - Mk II FPA', Wiley & Sons, 1991.
- Improved version of Albrecht FP
- Mainly used in UK
- Was developed in parallel to IFPUG FPs

Function Points Mk II



- For each transaction, the UFPs are calculated:

  W_i X (Number of input data element types, N_i) +

  W_e X (Number of entity data element types, N_e) +

  W_0 X (Number of output data element types, N_o)

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

Example: Mark II Function Points

- A screen that lists category products based on the category name entered by the user. This screen lists the product, its name, price, short description, thumbnail image, variations (size, colors and materials) and their stock availability.
- **Step 1:** Identify input types, entity types and output types
  - Input types: categoryName
  - Entity types: Category, Product
  - Output types: productName, price, shortDescription, smallImage, size, color, material, availability
- **Step 2:** Calculate function point amount using weights

0.58*1 + 1.66*2 + 0.26*8 = 5.98 function points.

Function Point Analysis – Summary

Measured from the user's perspective

- The size measured is based on the user's view of the system. The way the user interacts with the system, including the screens that the user uses to enter input, and the reports the users receive as output.

Technology-independent

- "Just show me your screens and your data tables and I will derive *number of function points* from there."

Low cost

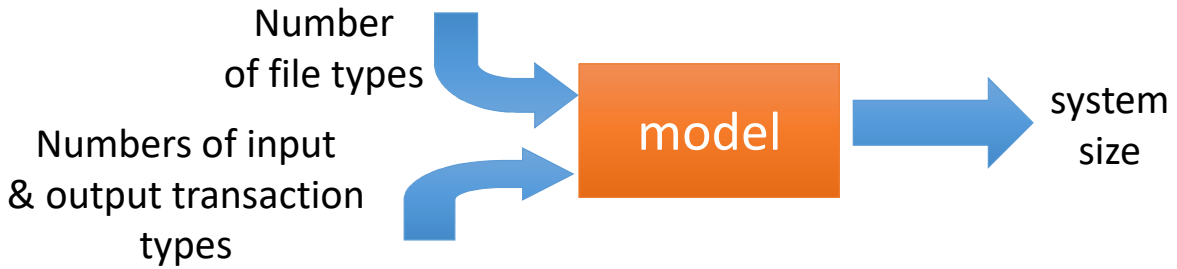
- Adding FPA to development process results in a cost increase of only 1%.

Repeatable

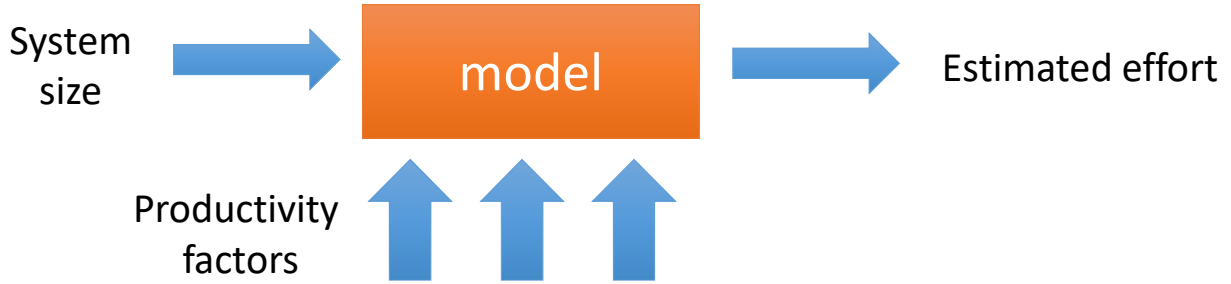
- Studies have shown that multiple function point counters can independently count the same application to within 10% accuracy of each other.

Algorithmic Models

- Based on an algorithm (procedure of calculation)
  - Function Point Methods are used to estimate software ***SIZE***



  - Constructive Cost Models (COCOMO) is used to estimate ***EFFORT***



Empirical Model (COCOMO)

COnstructive COst MOdel, by Barry W. Boehm,1970

Provide computational means for deriving software cost estimates as functions of variables (major cost drivers)

Functions used contain constants derived from statistical analysis of data from past projects:

- can only be used if data from past projects is available
- must be **calibrated** to reflect local environment
- relies on initial size and cost factor estimates which themselves are questionable

Next week …

Effort estimation best practices - 1

- Allow time for the estimate, and plan it
- Use documented data from previous projects
- Estimate by walk-through the WBS
- Estimate at a low level of detail
- Don’t omit common tasks
- Use software estimation tools
- Use several different estimation techniques, and compare the results
- Change estimation practices as the project progresses

Effort estimation best practices - 2

- Reduce situational and human bias
- Find estimation experts with highly relevant domain background and good estimation records
- Assess the uncertainty of the estimate
- Quantify risks – explicate the reasons of uncertainty
- Provide feedback on estimation accuracy and development task relations
- Provide estimation training opportunities

Thank you for your attention

Any questions, please?