



# Software Cost Prediction

Andrea Polini

Software Project Management  
MSc in Computer Science  
University of Camerino

# Objective

Before starting a new software project we would like to have a **rough estimation** of the required effort

- costs
- time
- required people
- ...

Cost estimation and project scheduling are generally **carried out together**

# Why?

A company has to assess the possible future cost of a development activity for different reasons:

- to provide a reference price to a **customer commissioning** a software
- to understand if entering a market is fruitful or not
- to understand if it is more convenient to buy or to build in house a software or a component

# Costs in well established engineering domains

## House

Consider the building of a new house:

- terrain area
- $m^2$
- used material
- number of flors
- ...

In general you make a rather simple calculation and you get a good estimation of the final price

- most of the time you can get less the 25% difference

# Impeding Factors

Many different factors make the estimation activity particularly difficult:

- requirements are not clear from the beginning
- costs are dominated by human labour
- many sources of uncertainty (technology, people, market)
- outsourcing and reuse
- subjective nature – generally simple project suffer from underestimation and big projects of overestimation
- lack of homogeneity of project experience
- ...

You can know that the estimation was correct, only when this information is not anymore useful to the project

# Impeding Factors

Many different factors make the estimation activity particularly difficult:

- requirements are not clear from the beginning
- costs are dominated by human labour
- many sources of uncertainty (technology, people, market)
- outsourcing and reuse
- subjective nature – generally simple project suffer from underestimation and big projects of overestimation
- lack of homogeneity of project experience
- ...

You can know that the estimation was correct, only when this information is not anymore useful to the project

# Where?

## Where are estimates done?

Estimates are carried out at various stages for a variety of reasons:

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

As a project proceeds the accuracy of estimates should improve as knowledge about the project increases

## Brooks' Law

Putting more people on a late job makes it later

# Where?

## Where are estimates done?

Estimates are carried out at various stages for a variety of reasons:

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

As a project proceeds the accuracy of estimates should improve as knowledge about the project increases

## Brooks' Law

Putting more people on a late job makes it later



# What can be done?

Different non alternative approaches can be taken to face the estimation problem:

- **Delay estimation** until late in the project
- Use relatively **simple decomposition techniques** to generate project cost and effort estimates (top-down or bottom-up)
- Use one or more **empirical models** for software cost and effort estimation
- Base estimates on **similar projects** that have already been completed
- **expert judgement**
- **Parkinson law**
- **Price to win**

Software cost estimation can be considered **more an art than a science**. The suggestion is to **apply different strategies by different people and check if they are too much far from each other**

# What can be done?

Different non alternative approaches can be taken to face the estimation problem:

- **Delay estimation** until late in the project
- Use relatively **simple decomposition techniques** to generate project cost and effort estimates (top-down or bottom-up)
- Use one or more **empirical models** for software cost and effort estimation
- Base estimates on **similar projects** that have already been completed
- **expert judgement**
- **Parkinson law**
- **Price to win**

Software cost estimation can be considered **more an art than a science**. The suggestion is to **apply different strategies by different people and check if they are too much far** from each other

# What can be done?

## Software Productivity

### Productivity

The very basic idea is to start from the formula:

$$\textit{effort} = (\textit{system size}) \times (\textit{productivity rate})$$

Productivity is the ratio between the number of units produced in a certain period of time split by the number of person hours needed to produce them.

- But is this approach really meaningful when software is considered?
  - Clearly it is an oversimplification. Additional factors have to be considered

Measure attribute of software and divide by the effort required. Two different approaches:

- Size related metrics
- Function related metrics

# What can be done?

## Software Productivity

### Productivity

The very basic idea is to start from the formula:

$$\textit{effort} = (\textit{system size}) \times (\textit{productivity rate})$$

Productivity is the ratio between the number of units produced in a certain period of time split by the number of person hours needed to produce them.

- But is this approach really meaningful when software is considered?
  - Clearly it is an oversimplification. Additional factors have to be considered

Measure attribute of software and divide by the effort required. Two different approaches:

- Size related metrics
- Function related metrics

# What can be done?

## Software Productivity

### Productivity

The very basic idea is to start from the formula:

$$\textit{effort} = (\textit{system size}) \times (\textit{productivity rate})$$

Productivity is the ratio between the number of units produced in a certain period of time split by the number of person hours needed to produce them.

- But is this approach really meaningful when software is considered?
  - Clearly it is an oversimplification. Additional factors have to be considered

Measure attribute of software and divide by the effort required. Two different approaches:

- Size related metrics
- Function related metrics

# What can be done?

## size related metrics

### Lines of Code – LOC

Estimation are generally based on the Lines of Code that are written by an average developer

The montly cost of an employee is at the base of many approaches. Consider that the cost of the labour is **not just the salary** but includes many related costs:

- lighting, heating, cleaning, ... of offices
- ICT resources such as networking, hardware
- administrators, accountants, ...

### IMPORTANT

- When you run a project keep track of “performances” for many different aspects. This data will be the main sources for future estimation activities.
- in small teams individual factors strongly influence the assessment

# What can be done?

## size related metrics

### Lines of Code – LOC

Estimation are generally based on the Lines of Code that are written by an average developer

The montly cost of an employee is at the base of many approaches. Consider that the cost of the labour is **not just the salary** but includes many related costs:

- lighting, heating, cleaning, . . . of offices
- ICT resources such as networking, hardware
- administrators, accountants, . . .

### IMPORTANT

- When you run a project keep track of “performances” for many different aspects. This data will be the main sources for future estimation activities.
- in small teams individual factors strongly influence the assessment

# What can be done?

## Function related metrics - Function Points

### Albrecht function point analysis

A Function Point (FP) is a **metric used to measure the functionality** provided by a software system. FP have been introduced to solve some **issues with LOCs** and in particular their dependency from the used programming language.

- Nonetheless it is possible to transform FPs in LOC taking into account many different dimension to derive a multiplying factor.



# Function Points (How-to 1/4)

A five steps strategy to identify FP:

- 1 Identify the functions (e.g. “retrieve”, “display” . . . ) that the application must have. The International Function Point Users Group has published criteria as to what constitutes a “function”
- 2 For each function compute its function point contribution considering the following parameter (**external user type**):
  - **#External inputs – EI** : each individual input that affect the function/system in a different way from each other
  - **#External outputs – EO**: only outputs that account for true separate algorithms or non-trivial functionalities are counted
  - **#External inquiry – EQ**: each independent inquiry, that is information provided to the user that do not require to modify internal data
  - **#Internal Logical Files – ILF**: each unique logical group of user data created by or maintained by the application
  - **#External Logical Files – ELF**: each unique logical group of data on system external to the application

**Note:** UML diagrams or even a DFD can include enough information to derive the values for each parameter

## Function Points (How-to 2/4)

- 3 For each parameter historical data should be used to define a **value expressing the level o complexity** of an FP and derive the *UFP* – Unadjusted FP – value

FP	Low	Average	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
ELF	5	7	10

- 4 Use adjustment for general characteristics of the project according to 14 identified categories. The value goes from 0 to 5:



# Function Points (How-to 3/4)

## General Characteristics – GCs

- 1 Does the system require reliable backup and recovery?
- 2 Are specialized data communications required to transfer information to or from the application?
- 3 Are there distributed processing functions?
- 4 Is performance critical?
- 5 Will the system run in an existing, heavily utilized operational environment?
- 6 Does the system require online data entry?
- 7 Does the online data entry require the input transaction to be built over multiple screens or operations?
- 8 Are the ILFs updated online?
- 9 Are the inputs, outputs, files, or inquiries complex?
- 10 Is the internal processing complex?
- 11 Is the code designed to be reusable?
- 12 Are conversion and installation included in the design?
- 13 Is the system designed for multiple installations in different organizations?
- 14 Is the application designed to facilitate change and ease of use by the user?

## Function Points (How-to 4/4)

5 Compute the adjusted FP using the following formula:

$$FP = UFP \times (0.65 + 0.01 \times \sum(GC_i))$$

Where:

*UFP* – Unadjusted FP

*GC<sub>i</sub>* – value for the i-th general characteristic

# What can be done?

## Function related metrics - Objects points

Object points constitute a further strategy to define the cost of a system. They intend to solve some issues related to FP:

- FP are considered to be too much dependent on the judgement of the estimator (e.g. complexity)
- biased towards data-processing systems
- different to make estimation for event-driven systems

OP can be computed in the following way:

- Number of separated screens displayed (1 Simple, 2 Complex, 3 Very Complex)
- Number of reports that are produced (2S,5C,8V)
- Number of modules in imperative programming languages to be developed to supplement database programming code (10 for each)

# What can be done?

## Function related metrics - Objects points

Object points constitute a further strategy to define the cost of a system. They intend to solve some issues related to FP:

- FP are considered to be too much dependent on the judgement of the estimator (e.g. complexity)
- biased towards data-processing systems
- different to make estimation for event-driven systems

OP can be computed in the following way:

- Number of separated sceens diplayed (1 Simple, 2 Complex, 3 Very Complex)
- Number of reports that are produced (2S,5C,8V)
- Number of modules in imperative programming languages to be developed to supplement database programming code (10 for each)

# Use Case based estimation

Use cases provide information on the scope and requirements of the system, so they seem suitable to make predictions. However different UCs do not require the **same amount of time to be satisfied**

$$SLOC = N \times LOC_{avg} + ((S_a/S_h - 1) + (P_a/P_h - 1)) \times LOC_{adg}$$

Where:

$N$  – number of UCs

$LOC_{avg}$  – historical average of LOC for this kind of subsystems

$LOC_{adg}$  – adjustment to measure the difference between the project and the average one

$S_a$  – actual scenarios for the UC

$S_h$  – average scenarios for UC for this kind of subsystem

$P_a$  – actual pages per UC

$P_h$  – average pages for UC of this type of subsystems

# Use cases – issues

UC can be useful but it is important to consider that:

- **specification styles** are strongly depending from the analyst
- different levels of **abstractions**
- do not generally include **information related to complexity**
- do not include information concerning the **interactions of system components**

This aspects make UC based estimation not completely reliable



# Estimation based on similarities

In such a case the estimation is based on the cost recorded in carrying on a **project that appear to be rather similar**

Strategies are then based on “**Ask the expert**” approach:

## Analogy

- 1 select several experts
- 2 ask them to independently provide an estimation for the software
  - **optimistic, pessimistic and most likely**
- 3 
$$S = \frac{s_{opt} + 4s_{ml} + s_{pes}}{6}$$

## Delphi technique

- 1 select several experts
- 2 ask them to independently provide an estimation for the software
- 3 organize a meeting in which they can present their results
- 4 ask them if they want to revise, if yes go to 2
- 5 compute the estimation  $S$  as the average

# Decomposition

To reach an estimation of the project costs the problem is decomposed according to:

- Product
- Process

and then costs are associated to “components” while the estimation is a recomposition of the factors

## Product

To proceed with a product based decomposition a rough architecture and main components are identified (bottom-up)

# Decomposition

To reach an estimation of the project costs the problem is decomposed according to:

- Product
- Process

and then costs are associated to “components” while the estimation is a recomposition of the factors

## Product

To proceed with a product based decomposition a rough architecture and main components are identified (bottom-up)

# Estimation based on similarities

Wolverton suggest to codify previous experiences in a matrix where rows are **type of components** while columns are related to **difficulty** related to two different factors:

- **Novelty** (with two levels: O,N)
- **Complexity** (with three levels: E,M,H)

Cells contains the historical cost for the development of one **unit of development** for such kind of components

<i>Type of Software</i>	<i>Difficulty</i>					
	OE	OM	OH	NE	NM	NH
Control	21	27	30	33	40	49
Input/Output	17	24	27	28	35	43
Pre/post processor	16	23	26	28	34	42
Algorithm	15	20	22	25	30	35
Data management	24	31	35	37	46	57
Time-critical	30	35	37	45	55	65

# Process based estimations

Simple approach that foresees the decomposition of the project in **smaller activities** and the provisioning of estimation for each activity. Data are organized in a matrix in which **rows include the various components** and the **columns the engineering activities**. Each cell will include the Person Months (PM).

This strategy relates to the decomposition in activities performed to derive the duration of the project and then to the corresponding allocation of resources.

# Algorithmic cost modelling

This approach **uses mathematical formula to predict costs** based on estimates of the project size. They are generally derived using **data regression approaches**.

The general structure is:

$$E = (a+bS^c)m(\mathbf{X})$$

Where:

$E$  – represents the effort in PM

$a, b, c$  – are constants

$S$  – represents the size of the product (e.g. FP, LOC ...)

$m$  – adjustment multiplier based on the factors in  $\mathbf{X}$

$\mathbf{X}$  – is a vector of cost factors combining process, product, and development attributes (staff experience, problem complexity ...)

Some more words about  $c$  ...

# Algorithmic cost modelling

This approach **uses mathematical formula to predict costs** based on estimates of the project size. They are generally derived using **data regression approaches**.

The general structure is:

$$E = (a+bS^c)m(\mathbf{X})$$

Where:

$E$  – represents the effort in PM

$a, b, c$  – are constants

$S$  – represents the size of the product (e.g. FP, LOC ...)

$m$  – adjustment multiplier based on the factors in  $\mathbf{X}$

$\mathbf{X}$  – is a vector of cost factors combining process, product, and development attributes (staff experience, problem complexity ...)

**Some more words about  $c$  ...**

## Different proposal

From the formula defined above researchers have made many proposals in relation to **different application domains and companies**:

$$E = 5.2 \times (KLOC)^{0.91}$$

Walston - Felix

$$E = 5.5 + 0.73 \times (KLOC)^{1.16}$$

Bailey - Basili

$$E = 3.2 \times (KLOC)^{1.05}$$

simple of Boehm

$$E = 5.288 \times (KLOC)^{1.047}$$

Doty (for KLOC > 9)

Similar proposal can be found using FP instead of LOC

- **often difficult to make estimation of size**
- **c and X** are based on subjective analysis



# COCOMO

COCOMO is an **empirical model** derived by collecting data from a large number of software projects. Three levels: basic, intermediate, detailed

Assumptions:

- Waterfall process
- imperative programming language

For the basic level the following data are suggested:

Simple	$PM = 2.4(KDSI)^{1.05} \times M$	Well understood, small team
Moderate	$PM = 3.0(KDSI)^{1.12} \times M$	complex and teams with limited experience
Embedded	$PM = 3.6(KDSI)^{1.20} \times M$	complex project, coupling Hw/Sw, regulations, procedures, ...

# COCOMO

COCOMO is an **empirical model** derived by collecting data from a large number of software projects. Three levels: basic, intermediate, detailed

Assumptions:

- Waterfall process
- imperative programming language

For the basic level the following data are suggested:

Simple	$PM = 2.4(KDSI)^{1.05} \times M$	Well understood, small team
Moderate	$PM = 3.0(KDSI)^{1.12} \times M$	complex and teams with limited experience
Embedded	$PM = 3.6(KDSI)^{1.20} \times M$	complex project, coupling Hw/Sw, regulations, procedures, ...

# COCOMOII

Introduction of novel technologies and methodologies made COCOMO outdated:

- Distributed objects and web services
- CoTS
- Powerful CASE tools

COCOMO II (2000) is a revision of the COCOMO model defined in 1981 to update the model to recent development strategies, like **reuse of code, component composition, database programming**.

Sub models part of COCOMO II are:

- **Application composition model**
- **Early design model**
- **Reuse Model**
- **Post-architecture model**

# Application composition model

## When

System characteristics:

- reusable components, scripting or database programming
- development of a prototype

The estimation is based on object points:

$$PM = (NAP \times \%reuse/100)/PROD$$

*NAP*      Number of Object Points - here called Action Points

*PROD*    Object point productivity of programmers

# Application composition model

## When

System characteristics:

- reusable components, scripting or database programming
- development of a prototype

The estimation is based on object points:

$$PM = (NAP \times \%reuse/100)/PROD$$

*NAP*      Number of Object Points - here called Action Points

*PROD*     Object point productivity of programmers

# Early design model

## When

Early stages of system design when **requirements have been established**

The estimation is based on classical formula:

$$PM = A \times Size^B \times M$$

Where

- A should be 2,94 and size is expressed in function points and reconducted to KLOC using tables that take into accounts the used language
- $1.1 < B < 1.24$  depending on the novelty of the project, development flexibility, risk resolution processes used, cohesion of the team, process maturity level.
- M is the product of seven parameters related to project and process characteristics that influence the estimate.

# Early design model

## When

Early stages of system design when **requirements have been established**

The estimation is based on classical formula:

$$PM = A \times Size^B \times M$$

Where

- A should be 2,94 and size is expressed in function points and reconducted to KLOC using tables that take into accounts the used language
- $1.1 < B < 1.24$  depending on the novelty of the project, development flexibility, risk resolution processes used, cohesion of the team, process maturity level.
- M is the product of seven parameters related to project and process characteristics that influence the estimate.

# Early design model

## Factor M

Based on the multiplication of **seven factors related to project and process** that are evaluated on a 7 value scale (Extra Low – Extra High)

- **RCPX**: reliability and complexity
- **RUSE**: reuse required
- **PDIF**: platform difficulty
- **PERS**: personnel capability
- **PREX**: personnel experience
- **SCED**: schedule
- **FCIL**: support facilities

M can assume values in the range 0.078 – 60.679



# Reuse model

## When

Software built over reusable components or automatically generated code

## Code automatically generated

The estimation is based on object points:

$$PM_{AUTO} = (ASLOC \times AT / 100) / ATPROD$$

- AT is the percentage of adapted code that is automatically generated
- ATPROD is the productivity of engineers in integrating such code

## White box reuse

$$ESLOC = ASLOC \times (1 - AT / 100) \times AAM$$

- AAM adaptation adjustment multiplier (AAF+SU+AA)

# Reuse model

## When

Software built over reusable components or automatically generated code

## Code automatically generated

The estimation is based on object points:

$$PM_{AUTO} = (ASLOC \times AT/100) / ATPROD$$

- AT is the percentage of adapted code that is automatically generated
- ATPROD is the productivity of engineers in integrating such code

## White box reuse

$$ESLOC = ASLOC \times (1 - AT/100) \times AAM$$

- AAM adaptation adjustment multiplier (AAF+SU+AA)

# Reuse model

## When

Software built over reusable components or automatically generated code

## Code automatically generated

The estimation is based on object points:

$$PM_{AUTO} = (ASLOC \times AT/100) / ATPROD$$

- AT is the percentage of adapted code that is automatically generated
- ATPROD is the productivity of engineers in integrating such code

## White box reuse

$$ESLOC = ASLOC \times (1 - AT/100) \times AAM$$

- AAM adaptation adjustment multiplier (AAF+SU+AA)

# Post-architecture model

## When

The system architecture is available. Therefore constitutes the **most detailed model**

The estimation is based on classical formula:

$$PM = A \times Size^B \times M$$

But now factors to be considered are 17

# Post-architecture model

## When

The system architecture is available. Therefore constitutes the **most detailed model**

The estimation is based on classical formula:

$$PM = A \times Size^B \times M$$

But now factors to be considered are 17

# Post-architecture model

## What about B

B becomes a variable that can assume values in a **continuous set**. Five scale factors (0-5) – **To compute B Make the sum divide by 100 and sum to 1.01** :

- **Precedentedness**: previous experience of the organization
- **Development flexibility**: flexibility in the process
- **Architecture/risk resolution**: risk analysis carried out
- **Team Cohesion**: how well team members know each other and work well together
- **Process maturity**: process maturity of the organization

Driver	Very Low	Low	Nominal	High	Very High	Extra High
PREC	6.20	4.96	3.72	2.48	1.24	0
FLEX	5.07	4.05	3.04	2.03	1.01	0
RESL	7.07	5.65	4.24	2.83	1.41	0
TEAM	5.48	4.38	3.29	2.19	1.10	0
PMAT	7.80	6.24	4.68	3.12	1.56	0

B can assume values in the range 1.073 – 1.326

# Post-architecture model

## What about M

The 17 factors are organized according to 4 classes:

- Product
- Hardware
- Personnel
- Project

Each parameter can be given a rating of *very low*, *low*, *nominal*, *high*, *very high*

# Project Cost Drivers

- Required system reliability – Product
- Complexity of system modules – Product
- Extent of documentation required – Product
- Size of database used – Product
- Required percentage of reusable components – Product
- Execution time constraints – Hardware
- Volatility of development platforms – Hardware
- Memory constraints – Hardware
- Capability of project analysts – Personnel
- Personnel continuity – Personnel
- Programmer capability – Personnel
- Programmer experience in project domain – Personnel
- Analyst experience in project domain – Personnel
- Language and tool experience – Personnel
- Use of software tools – Project
- Development schedule compression – Project
- Extent of multi-site working and quality or inter-site communications – Project



## Other alternatives

### Parkinson's Law

Cost estimated **by available resources** rather than by assessment. 5 people available and software to be delivered in 6 month ...

### Pricing to win

Whatever the customer is willing to spend on the software.

# Conclusion

## Rule of thumb


Make estimations involving many experts, and using many different approaches. If results are not coherent investigate on possible causes

# Resources


Study material can be found here (*also previous editions of the following books*):

 **Bob Hughes and Mike Cotterell**  
*Software Project Management, 5th Ed.*  
McGraw-Hill, 2009

- Chapter 5 - Software Effort Estimation

 **Roger Pressman and Bruce Maxim**  
*Software Engineering a Practioner's Approach 8<sup>th</sup> Ed.*  
McGraw-Hill 2015.

- Chapter 33 - Estimation of Software Projects

 **Ian Sommerville**  
*Software Engineering 10<sup>th</sup> Ed.*  
Addison Wesley 2016.

- Chapter 23 - Project Planning