Chapter 23

Estimation

Software Engineering: A Practitioner's Approach
6th Edition
Roger S. Pressman

Software Project Estimation

Software estimation is the process of predicting the effort, time, and resources required for the development of a software project. It involves making educated guesses or calculations to determine the size, scope, and complexity of the project, as well as the associated costs and schedule.

Estimation is a critical activity in project management and is performed at various stages of the software development life cycle to guide decision-making, resource allocation, and planning. There are different types of software estimates, including cost estimation, effort estimation, risk estimation, size estimation.

Software Project Estimation (1)

- S/W is the most expensive element of virtually all computer based systems
- S/W cost and effort estimation will never be an exact science
  - Too many variables
    - Human
    - Technical
    - Environmental
    - Political

Decomposition Techniques

In software estimation, the decomposition technique involves breaking down a project into smaller, more manageable components or tasks to facilitate a more accurate estimation of effort, time, and resources. This technique is particularly useful for complex projects where estimating the entire scope at once may be challenging. Decomposition helps in creating a hierarchical structure of the project, allowing for a more detailed and focused estimation at different levels.

Decomposition Techniques

- Two different points of view for the decomposition approach
  - Decomposition of the problem
  - Decomposition of the process
- But first, the project planner must
  - Understand the scope of the s/w to be built
  - Generate an estimate of its “size”

Software Sizing (1)

- The accuracy of a s/w project estimate is predicated on a number of things:
  – The degree to which the planner has properly estimated the size of the product to be built
  – The ability to translate the size estimate into human effort, calendar time, and dollars (required availability of past records)
  – The degree to which the project plan reflects the abilities of the s/w team
  – The stability of product requirements and the environment that supports the s/w engineering effort

Software Sizing (2)

- Sizing represents the project planner's first major challenge
- *Size* refers to a quantifiable outcome of the s/w project (e.g. LOC and/or FP)
- Four different approaches to the sizing problem [PUT92]
  - *"Fuzzy Logic" sizing*
  - *Function point sizing*
  - *Standard component sizing*
  - *Change sizing*

What is LOC?

Source lines of code (SLOC), also known as lines of code (LOC), is a software metric used to measure the size of a computer program by counting the number of lines in the text of the program's source code.

What is Function Point(FP)?

Function points measure the size of an application system based on the functional view of the system. The size is determined by counting the number of inputs, outputs, queries, internal files and external files in the system and adjusting that total for the functional complexity of the system.

Problem-Based Estimation (1)

- Example of baseline productivity metrics are LOC/pm or FP/pm
- Making the use of single baseline productivity metric is discouraged
- In general, LOC/pm or FP/pm averages should be computed by project domain
- Local domain averages should be used

Problem-Based Estimation (2)

- Statistical approach – *three-point* or *expected-value* estimate
- $S = (s_{opt} + 4s_m + s_{pess})/6$
  - S = expected-value for the estimation variable (size)
  - s_{opt} = optimistic value
  - s_m = most likely value
  - s_{pess} = pessimistic value

An Example of LOC-Based Estimation (1)

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

An Example of LOC-Based Estimation (2)

- Estimated lines of code = W = 33,200
- Let,
  - Average productivity = 620 LOC/pm = X
  - Labor rate = $8,000 per month = Y
- So,
  - Cost per line of code = Z = Y/X = $13 (approx.)
  - Total estimated project cost = W*Z = $431,000 (approx.)
  - Estimated effort = W/X = 54 person-months (approx)

An Example of FP-Based Estimation (1)

Information Domain Value	Count		Weighting factor				
			Simple	Average	Complex		
External Inputs (EIS)	3	X	**3**	4	6	=	9
External Outputs (EOs)	2	X	**4**	5	7	=	8
External Inquiries (EQs)	2	X	**3**	4	6	=	6
Internal Logical Files (ILFs)	1	X	**7**	10	15	=	7
External Interface Files (EIFs)	4	X	**5**	7	10	=	20
Count Total							50

Figure 15.4: Computing function points

An Example of FP-Based Estimation (2)

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
Count total						*320*

An Example of FP-Based Estimation (3)

Factor	Value
1. Backup and recovery	4
2. Data communications	2
3. Distributed processing	0
4. Performance critical	4
5. Existing operating environment	3
6. On-line data entry	4
7. Input transaction over multiple screens	5
8. ILFs updated online	3
9. Information domain values complex	5
10. Internal processing complex	5
11. Code designed for reuse	4
12. Conversion/installation in design	3
13. Multiple installations	5
14. Application designed for change	5

Value Adjustment Factors

An Example of FP-Based Estimation (4)

- Now,
  - $FP_{estimated}$ = count-total $\times$ [0.65 + 0.01 $\times \Sigma$ (F_i)]
    - F_i (i = 1 to 14 are value adjustment factors)
- So,
  - $FP_{estimated}$ = W = 320 $\times$ [0.65 + 0.01 $\times$ 52] = 375 (approx.)
- Let,
  - Average Productivity = X = 6.5 FP/pm
  - Labor rate = Y = $8,000 per month
- So,
  - Cost per FP = Z = Y/X = $1,230 (approx.)
  - Total estimated project cost = W*Z = $461,000 (approx.)
  - Estimated effort = W/X = 58 person-months (approx)

COCOMO II, which stands for Constructive Cost Model II, is a software cost estimation model. It is an extension and improvement of the original COCOMO (COnstructive COst MOdel) developed by Barry Boehm in the 1980s. COCOMO II was introduced to address the evolving complexities of software development projects and to provide more accurate and flexible estimates.

Here are the key aspects of the COCOMO II model:

Three Sub-Models:

Application Composition Model (COCOMO II-ACM): This sub-model is designed for estimating the effort and cost of developing software using commercial off-the-shelf (COTS) components or existing software assets.

Early Design Model (COCOMO II-EDM): Used for quick and rough estimates during the early stages of a project when only limited information is available.

Post-Architecture Model (COCOMO II-Post-Architecture): Applied when detailed information about the system architecture is available, enabling more accurate estimation.

Estimation Parameters:

COCOMO II considers a range of parameters to estimate software development effort, including lines of code, personnel capability, development flexibility, and risk resolution.

Effort Estimation:

COCOMO II estimates effort in person-months, where effort is a measure of the amount of human labor required to complete a project.

Cost Estimation:

The model provides estimates of project cost based on effort, taking into account labor rates, overhead, and other cost factors.

Schedule Estimation:

COCOMO II helps in estimating the project schedule by considering the planned development time and the distribution of effort across different development phases.

Risk Management:

COCOMO II includes features for managing and incorporating project risks into the estimation process, allowing for a more realistic and informed assessment of potential challenges.

Flexibility:

The model is designed to be adaptable to various types of projects, including those involving different development methodologies, technologies, and team sizes.

COCOMO II provides a more sophisticated and customizable approach compared to the original COCOMO model. It is widely used in the software industry for project planning, budgeting, and risk management, helping stakeholders make informed decisions about resource allocation and project feasibility.

The COCOMO II Model (1)

- ***CO**nstructive **CO**st **Mo**del*
- A hierarchy of estimation models
- Addresses the following areas
  - Application composition model
  - Early design stage model
  - Post-architecture stage model
- Three different sizing options are available
  - Object points
  - Function points
  - Lines of source code

The COCOMO II Model (2)

- The COCOMO II application composition model uses object points
- Object point is computed using counts of the number of
  - Screens (at the user interface)
  - Reports
  - Components likely to be required to build the application

The COCOMO II Model (3)

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

Figure 23.6: Complexity weighting for object types

The COCOMO II Model (4)

Developer's experience/capability	Very low	Low	Nominal	High	Very High
Environment maturity/capability	Very low	Low	Nominal	High	Very High
PROD	4	7	13	25	50

Figure 23.7: Productivity rate for object points

The COCOMO II Model (5)

- NOP = (object points) × [(100-%reuse)/100]
  - NOP = New Object Points
  - Object Points = Weighted Total
  - %reuse = Percent of reuse
- Estimated effort = NOP/PROD
  - PROD = Productivity Rate
  - PROD = NOP/person-month

Functional Point - Estimation

1. What is the **Functional Point?**

To measure the standard worth of the software, as a unit of software worth, Function Point was developed.

Allan Albrecht of IBM in 1977.

FP measures functionality from the User's point of view like what the user receives from the software and what the user requests from the software. It focuses on what functionality is being delivered.

"A Functional Point" is a unit of measurement to express the amount of business functionality an information system provides to a user. - Wiki

Using historical data, the FP metric can then be used to

1. Estimate the cost or effort required to design, code, and test the software.

2. Predict the number of errors that will be encountered during testing

3. Forecast the number of components and/or the number of projected source lines in the implemented system.

Functional Point - Estimation

The five functional units to calculate the FP are.

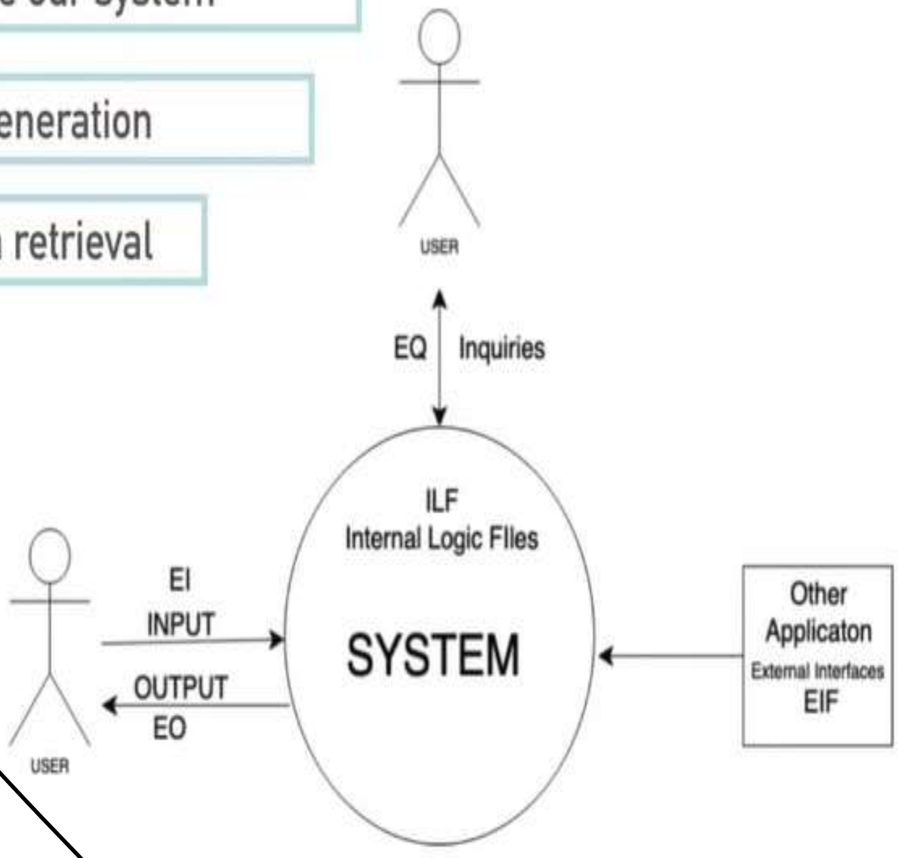
1. Internal Logic Files (ILF) – The control info or logically related data that is present within the system.

2. External Interface Files (EIF) – The control data referenced by the system but present in another system.

3. External Inputs (EI) – Data / control info that comes from outside our system

4. External Outputs (EO) – data that goes out of the system after generation

5. External Enquired (EQ) – Combination of i/o – o/p resulting data retrieval



Functional Point - Estimation

To compute FP, the following relationship is used

$$F.P = UFP \ X \ CAF$$

Where,

UFP = *Unadjusted Functional Point*

CAF = *Complexity Adjustment Factor*

Functional Point - Estimation

Software Engineering

Step 1: Calculating UFP

Functional Unit	Wighting Factors		
	Low	Average	High
External Inputs (EI)	3	4	6
External Outputs (EO)	4	5	7
External Enquired (EQ)	3	4	6
Internal Logic Files (ILF)	7	10	15
External Interface Files (EIF)	5	7	10

$$F.P = UFP \ X \ CAF$$

Step 1: Calculating UFP - Unadjusted Function Point

$$F.P = UFP \ X \ CAF$$

Measurement Parameter	Counts		Weighting Factor Low	Average	High		
External Inputs (EI)		X	3	4	6	=	+
External Outputs (EO)		X	4	5	7	=	+
External Enquired (EQ)		X	3	4	6	=	+
Internal Logic Files (ILF)		X	7	10	15	=	+
External Interface Files (EIF)		X	5	7	10	=	

Unadjusted Function Points (UFP) = Total Count =

UFP = Sum of all the Complexities of all the EI's, EO's, EQ's, ILF's, and EIF's

Step 2: Calculating CAF - Complexity Adjustment Factor

1. Data Communication
2. Distributed Data Processing
3. Performance
4. Heavily Used Configuration
5. Transaction Role
6. Online Data Entry
7. End-User Efficiency
8. Online Update
9. Complex Processing
10. Reusability
11. Installation Ease
12. Operational Ease
13. Multiple Sites
14. Facilitate Change

Complexity Adjustment Factor is calculated using 14 aspects of processing complexity and these 14 questions answered on a scale of 0 – 5

0 - No Influences or No Important

1 - Incidental

2 - Moderate

3 - Average

4 - Significant

5 - Essential

Functional Point - Estimation

Software Engineering

Step 1: Calculating UFP

$$F.P = UFP \ X \ CAF$$

Measurement Parameter	Counts		Weighting Factor Low	Average	High		
External Inputs (EI)	3	X	3			=	9
							+
External Outputs (EO)	2	X	4			=	8
							+
External Enquired (EQ)	2	X	3			=	6
							+
Internal Logic Files (ILF)	1	X	7			=	7
							+
External Interface Files (EIF)	4	X	5			=	20

Unadjusted Function Points (UFP) = Total Count = 50

UFP = Sum of all the Complexities of all the EI's, EO's, EQ's, ILF's, and EIF'

Step 2: Calculating CAF - Complexity Adjustment Factor

$$F.P = UFP \times CAF$$

$$CAF = 0.65 + (0.01 \times \sum Fi)$$

Where,

F_i = *Value adjustment factors based on responses to the following 14 questions*

$$F.P = UFP \times CAF$$

$$CAF = 0.65 + (0.01 \times \sum Fi)$$

Moderately complex product

Then,

$$\sum Fi = 14 \times 2 = 28$$

2 - Moderate

$$CAF = 0.65 + (0.01 \times 28)$$

$$CAF = 0.93$$

$$F.P = UFP \times CAF$$

$$F.P = 50 \times 0.93 = 46.5$$

COCOMO-II Model

- COnstructive COst Model - II by Barry Boehm

- <u>Application composition model</u> - Used during the early stages of software engineering

- Early design stage model - used once requirements have been stabilized and basic software architecture has been established.

- Post-architecture-stage model - Used during the construction of the software

- The sizing information followed by this model is the indirect software measure *object points.*

Object Points

- Object Point is computed using counts of the number of
  - Screens (at the user interface)
  - Reports
  - Components likely to be required to build the application.
- Each object instance (e.g., a screen or report) is classified into one of three complexity levels (i.e., simple, medium, or difficult).
- In essence, complexity is a function of
  - number and source of the client and server data tables that are required to generate the screen or report and
  - number of views or sections presented as part of the screen or report.

Object Points

•Once complexity is determined, the number of screens, reports, components are weighted according to following table

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

•The object point count is then determined by multiplying the original number of object instances by the weighting factor in the figure and summing to obtain a total object point count.

•When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted:

$$NOP = (\text{object points}) * [(100 - \%\text{reuse})/100]$$

where NOP is defined as new object points.

Effort Estimation

- Now the estimate of project effort is computed as follows.

$$\text{Estimated effort} = \frac{NOP}{PROD}$$

- Productivity rate can be derived from following table based on developer experience and organization maturity.

Developer's experience/capability	Very low	Low	Nominal	High	Very high
Environment maturity/capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

COCOMO – II Example

- Use the COCOMO II model to estimate the effort required to build software for a simple ATM that produces 12 screens, 10 reports, and will require approximately 80% as new software components. Assume average complexity and average developer/environment maturity. Use the application composition model with object points.

- Given

Object	Count	Complexity	Weight Factor	Total Objects
Screen	12	Simple	1	12
Report	10	Simple	2	20
3GL Components	0	NA	NA	0
			Total Objects Points :	32

COCOMO – II Example

- It is given that 80% of components have to be newly developed. So remaining 20% can be reused

- Now compute new object points as

  NOP = (object points) * [(100 - %reuse)/100]

  NOP = 32 * (100-20)/100 = 32*80 / 100

  NOP = 25. 6 object points

- Since productivity is given average, we can assume PROD = 13

- Hence , effort = NOP/PROD

  effort = 25.6/13