DA-IICT

IT 314: Software Engineering

Software Effort/Cost Estimation
Function Point – Object Point

Saurabh Tiwari

1

Bus Facility to and fro between Campus and Railway station, some city points

Motivation

- How much effort is required to complete the project?
- How much calendar time is needed to complete the project?
- What is the total cost?
- Project estimation and scheduling (project -> activities -> tasks -> roles)

Resources (employees, stakeholders (users, developers testers...)
Time
Effort
Cost

Software Cost Components

- Effort costs (dominant factor in most projects)
  - salaries
  - Social and insurance & benefits
- Tools costs: Hardware and software for development
  - Depreciation on relatively small # of years
- Travel and Training costs (for particular client)
- Overheads(OH): Costs must take overheads into account
  - costs of building, air-conditioning, heating, lighting
  - costs of networking and communications (tel, fax,)
  - costs of shared facilities (e.g. library, staff restaurant, etc.)
  - depreciation costs of assets
  - Others...

Issues with Effort/Cost Estimation

- Ambiguous area (Impossible to estimate the unknown)
- Perfectionist & Confident Estimation (usually optimistic)
- A lot of time is spent on things that are not estimated
- Estimates do not consider productivity variations between programmers
- Changes in requirements do not reflect on estimations
- Wrong people do the estimates
- Others...

Software Estimation

Predicting the resources required for a software development process

Objectives:

•To introduce the fundamentals of software costing and pricing

•To describe

- **LOC model**
- **COCOMO 'Constructive Cost Model'**
  - Object-point model
  - Function points model

Programmer productivity

- A measure of the rate at which individual engineers involved in software development produce software and associated documentation
- Not quality-oriented although quality assurance is a factor in productivity assessment
- Essentially, we want to measure useful functionality produced per time unit

Productivity measures

- Size related measures based on some output from the software process. This may be lines of delivered source code, object code instructions, etc.
- Function-related measures based on an estimate of the functionality of the delivered software. Function-points are the best known of this type of measure

Lines Of Code (LOC)

What's a line of code?

- The measure was first proposed when programs were typed on cards with one line per card
- How does this correspond to statements as in Java which can span several lines or where there can be several statements on one line

What programs should be counted as part of the system?

Assumes linear relationship between system size and volume of documentation

Productivity comparisons

- The lower level the language, the more productive the programmer
  - The same functionality takes more code to implement in a lower-level language than in a high-level language
- The more verbose the programmer, the higher the productivity
  - Measures of productivity based on lines of code suggest that programmers who write verbose code are more productive than programmers who write compact code

System Development Times

	Analysis	Design	Coding	Testing	Documentation
Assembly code	3 weeks	5 weeks	8 weeks	10 weeks	2 weeks
High-level language	3 weeks	5 weeks	8 weeks	6 weeks	2 weeks
	Size	**Effort**	**Productivity**		
Assembly code	5000 lines	28 weeks	714 lines/month		
High-level language	1500 lines	20 weeks	300 lines/month		

The COCOMO Cost model (Constructive Cost Model)

- COCOMO II is a 3-level model that allows increasingly detailed estimates to be prepared as development progresses
- Early prototyping level
  - Estimates based on object-points and a simple formula is used for effort estimation
- Early design level
  - Estimates based on function-points that are then translated to LOC
  - Includes 7 cost drivers
- Post-architecture level
  - Estimates based on lines of source code or function point
  - Includes 17 cost drivers

Object-Points (for 4GLs)

- Object-points are an alternative function-related measure to function points when 4Gls or similar languages are used for development
- Object-points are NOT the same as object classes
- The number of object-points in a program is considered as a weighted estimate of 3 elements:
  - The number of separate screens that are displayed
  - The number of reports that are produced by the system
  - The number of 3GL modules that must be developed to supplement the 4GL code

Object-Points (for 4GLs)

- Object-points are an alternative function-related measure to function points when 4Gls or similar languages are used for development
- Object-points are NOT the same as object classes
- The number of object-points in a program is considered as a weighted estimate of 3 elements:
  - The number of separate screens that are displayed
  - The number of reports that are produced by the system
  - The number of 3GL modules that must be developed to supplement the 4GL code

Object-Points Weighting

Object Type	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
Each 3GL Module	10	10	10

Object-Points Complexity Levels

Object point complexity levels for screens

	Number and sources of data tables		
Number of Views Contained	Total < 4	Total < 8	Total 8+
<3	simple	simple	medium
3-7	simple	medium	difficult
8+	medium	difficult	difficult

Object point complexity levels for reports

	Number and source of data tables		
Number of Sections Contained	Total < 4	Total < 8	Total 8+
0-1	simple	simple	medium
2-3	simple	medium	difficult
4+	medium	difficult	difficult

Source: http://yunus.hacettepe.edu.tr/~sencer/objectp.html

Object-Points Estimation

Object-points are easy to estimate

- simply concerned with screens, reports and 3GL modules

At an early point in the development process:

- Object-points can be early estimated
- It is very difficult to estimate the number of lines of code in a system

Productivity Estimates

LOC productivity
Real-time embedded systems: 40-160 LOC/P-month
Systems programs: 150-400 LOC/P-month
Commercial applications: 200-800 LOC/P-month

Object-points productivity: PROD
measured 4 - 50 object points/person-month
depends on tool support and developer capability

Developer's experience and Capability / ICASE maturity and capability	Very low	Low	Nominal	High	Very high
PROD: Productivity Object-point per person-month	4	7	13	25	50

Object Point Effort Estimation

Effort in p-m = NOP / PROD
NOP = number of OP of the system

Example:
An application contains 840 Object-points (NOP=840) & Productivity is very high (= 50 object points/person-month)

Then, Effort = 840/50 = (16.8) = 17 p-m

Adjustment for % of Reuse

% reuse: the % of screens, reports, & 3GL modules reused from previous applications, pro-rated by degree of reuse

•Adjusted NOP = NOP * (1 - % reuse / 100)
•Adjusted NOP: New NOP

•Example: An application contains 840 OP, of which 20% can be supplied by existing components.

Adjusted NOP = 840 * (1 – 20/100) = 672 OP "New OP"

Adjusted effort = 672/50 = (13.4) = 14 p-m

OP Estimation Procedure

1. Assess **object-counts** in the system: number of screens, reports, & 3GL.
2. Assess complexity level for each object (use table): simple, medium and difficult.
3. Calculate **"NOP" the object-point count** of the system: add all weights for all object instances
4. Estimate the % of reuse and compute the adjusted NOP "New Object Points" to be developed
5. Determine the productivity rate PROD (use metrics table)
6. Compute the adjusted effort PM = adjusted NOP / PROD

OP Estimation Example

- Assessment of a software system shows that:
- The system includes
  - 6 screens: 2 simple + 3 medium + 1 difficult
  - 3 reports: 2 medium + 1 difficult
  - 2 3GL components
- 30 % of the objects could be supplied from previously developed components
- Productivity is high
- Compute the estimated effort PM 'Person-months' needed to develop the system

OP Estimation Example: Solution

Object counts:

2 simple screens	x 1 =	2
3 medium screens	x 2 =	6
1 difficult screen	x 3 =	3
2 medium reports	x 5 =	10
1 difficult report	x 8 =	8
2 3GL components	x 10 =	20
NOP		**49**

OP Estimation Example: Solution

Adjusted NOP 'New NOP' = NOP * (1 - % reuse / 100)
= 49 * (1- (30/100))
= (34.3)
= 35

For high productivity (metric table): PROD = 25 OP/P-M

Estimated effort Person-Month = Adjusted NOP / PROD
= 35/ 25
= 1.4 P-M

Function Points

Function Points

A **Function Point** (FP) is a unit of measurement to express the amount of business functionality, an information system (as a product) provides to a user.

FPs measure software size.

FPs are a standard unit of measure that represent the functional size of a software application

They are widely accepted as an industry standard for functional sizing.

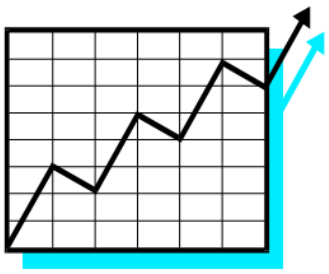
Function Point Analysis

Most practitioners of Function Point Analysis (FPA) will probably agree that there are three main objectives within the process of FPA:

- Measure software by quantifying the functionality requested by and provided to the customer.
- Measure software development and maintenance independently of technology used for implementation.
- Measure software development and maintenance consistently across all projects and organizations.

Why use Function Points?

- Size of Requirements
- Changes to Requirements
- Estimation Based on Requirements
- Measuring and Improving Productivity and Quality



Estimating Examples

Function Point Size	Project Variables	Project Estimate Based on Historical Data and/or Vendor Tool
Project A – 100 FPs	• On-line/database • New development • C++ • Highly experienced development staff	Effort = 5 months Schedule = 3 months Cost (@ $5K) = $25,000 KLOC = 6 Delivered Defects = 25 Productivity Rate = 20 FP/Month.
Project B – 100 FPs	• Batch • Enhancement • Cobol • Average experienced development staff	Effort = 20 months Schedule = 6 months Cost (@ $5K) = $100,000 KLOC = 10 Delivered Defects = 100 Productivity Rate = 5 FP/Month

Function Points vs. LOC

- Technology and platform independence
- Available from early requirements phase
- Consistent and objective unit of measure throughout the life cycle
- Objectively defines software application from the customer perspective
- Objectively defines a series of software applications from the customer's, not the technician's perspective
- Is expressed in terms that users can readily understand about their software

What is Wrong with LOC?

- There is no standard for a line of code
- Lines of Code measure components, not completed products
  - Don't measure the panels produced; measure the number of cars assembled
- Measuring lines of code
  - Rewards profligate design
  - Penalizes tight design
- Positively misleading?

Classic Productivity Paradox

Lines of Code	10,000	3,000
Function Points	25	25
Total Months effort	25	15
Total Costs	$125,000	$75,000
Cost per Source Line	$12.50	$25.00
Lines per Person month	400	200
FPs per Person month	1.2	2
Cost per FP	$5,000	$3,000

Function Points (5 Characteristics)

Based on a combination of program 5 characteristics

- The number of :
  - External (user) inputs: input transactions that update internal files
  - External (user) outputs: reports, error messages
  - User interactions: inquiries
  - Logical internal files used by the system: Example a purchase order logical file composed of 2 physical files/tables Purchase_Order and Purchase_Order_Item
  - External interfaces: files shared with other systems

FPs Standard

For sizing software based on FP, several recognized standards and/or public specifications have come into existence. As of 2013, these are -

ISO Standards

COSMIC - ISO/IEC 19761:2011 Software engineering. A functional size measurement method.

FiSMA - ISO/IEC 29881:2008 Information technology - Software and systems engineering - FiSMA 1.1 functional size measurement method.

IFPUG - ISO/IEC 20926:2009 Software and systems engineering - Software measurement - IFPUG functional size measurement method.

Mark-II - ISO/IEC 20968:2002 Software engineering - Ml II Function Point Analysis - Counting Practices Manual.

NESMA - ISO/IEC 24570:2005 Software engineering - NESMA function size measurement method version 2.1 - Definitions and counting guidelines for the application of Function Point Analysis.

Another View of FP

FP is a standard method for quantifying the software deliverable based upon the user view, where:

- User is any person or thing that communicates or interacts with the software at any time
- User View is the Functional User Requirements as seen by the user
- Functional user requirements describe what the software shall do, in terms of tasks and services.
- It is also important to keep in mind that function points look at the logical view, not the physical. So things like coding algorithms, database structure, screenshots of transactions are not counted.

FPs - Artifacts

Things that are counted within the Function Points methodology:

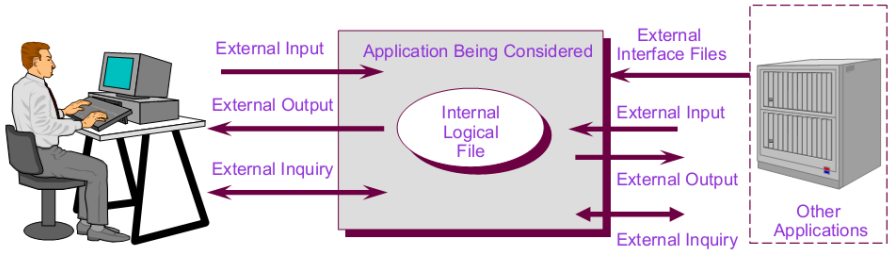
- Input files and input transactions (batch interfaces)
- Screens (adds, changes, deletes)
- Control Information
- Internal Logical Files (tables, files with data, control files)
- External tables and referenced files from other applications
- Output files and transactions (batch interfaces)
- Other outputs - reports, files, dvd's, views, notices, warnings

FPs

A weight is associated with each of the above 5 characteristics

- Weight range:
  - from 3 for simple feature
  - to 15 for complex feature
- The function point count is computed by multiplying each raw count by the weight and summing all values.

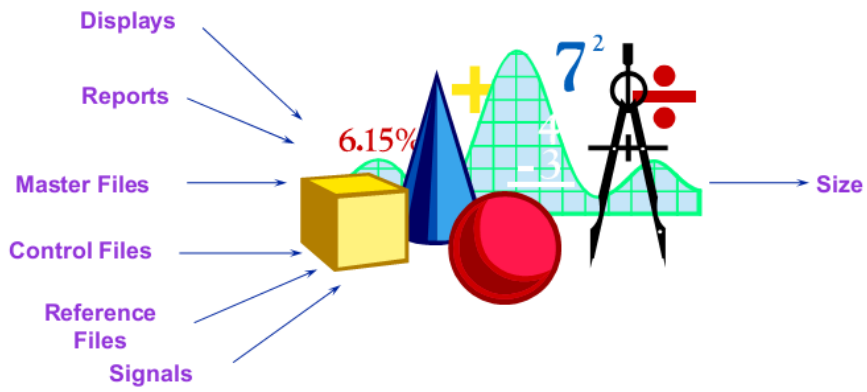
FPs are a Unit of Measure



- Functionality as viewed from the user's perspective

How to Count Function Points???

How to Count Function Points?



How to Count Function Points?

Internal Logical File (ILF)
Logical group of data maintained by the application (e.g., Employee file)

External Interface File (EIF)
Logical group of data referenced but not maintained (e.g., Global state table)

External Input (EI)
Maintains ILF or passes control data into the application

External Output (EO)
Formatted data sent out of application with added value (e.g., calculated totals)

External Query (EQ)
Formatted data sent out of application without added value

How to Count Function Points?

Function Type	Low	Average	High
EI	x 3	x 4	x 6
EO	x 4	x 5	x 7
EQ	x 3	x 4	x 6
ILF	x 7	x 10	x 15
EIF	x 5	x 7	x 10

Function Points - Calculation

measurement parameter	count		weighting factor simple	avg.	complex		
number of user inputs		X	3	4	6	=	
number of user outputs		X	4	5	7	=	
number of user inquiries		X	3	4	6	=	
number of files		X	7	10	15	=	
number of ext.interfaces		X	5	7	10	=	
count-total →							
complexity multiplier							
function points →							

Adjusted FPs Count Complexity:14 Factors Fi (cont.)

14 factors: Each factor is rated on a scale of:

Zero: not important or not applicable

Five: absolutely essential

1. Backup and recovery
2. Data communication
3. Distributed processing functions
4. Is performance critical?
5. Existing operating environment
6. On-line data entry
7. Input transaction built over multiple screens

Adjusted FPs Count Complexity:14 Factors Fi (cont.)

8. Master files updated on-line
9. Complexity of inputs, outputs, files, inquiries
10. Complexity of processing
11. Code design for reuse
12. Are conversion/installation included in design?
13. Multiple installations
14. Application designed to facilitate change by the user

Adjusted FPs Count Complexity:14 Factors Fi (cont.)

$$AFPC = UFPC * \left[0.65 + 0.01 * \sum_{i=1}^{i=14} F_i \right]$$

AFPC: Adjusted function point count
UFPC: Unadjusted function point count
$0 <= F_i <= 5$

Estimated LOC Approach

Assuming
- Estimated project LOC = 33200
- Organisational productivity (similar project type) = 620 LOC/p-m
- Burdened labour rate = 8000 $/p-m

Then
- Effort = 33200/620 = (53.6) = 54 p-m
- Cost per LOC = 8000/620 = (12.9) = 13 $/LOC
- Project total Cost = 8000 * 54 = 432000 $

Estimated FPs Approach

	A	B	C	D	E	F	G
1	Info Domain	Optimistic	Likely	Pessim.	Est Count	Weight	FP count
2	# of inputs	22	26	30	26	4	104
3	# of outputs	16	18	20	18	5	90
4	# of inquiries	16	21	26	21	4	84
5	# of files	4	5	6	5	10	50
6	# of external interf	1	2	3	2	7	14
7	UFC: Unadjusted Function Count						342
8			Complexity adjustment factor				1.17
9						FP	400

Estimated FPs Approach: Complexity Factor

Complexity factor: Fi	value=0	value=1	value=2	value=3	value=4	value=5	Fi
Backup and recovery	0	0	0	0	1	0	4
Data communication	0	0	1	0	0	0	2
Distributed processing functions	0	0	0	0	0	0	0
Is performance critical?	0	0	0	0	1	0	4
Existing operating environment	0	0	0	1	0	0	3
On-line data entry	0	0	0	0	1	0	4
Input transaction built over multiple screens	0	0	0	0	0	1	5
Master files updated on-line	0	0	0	1	0	0	3
Complexity of inputs, outputs, files, inquiries	0	0	0	0	0	1	5
Complexity of processing	0	0	0	0	0	1	5
Code design for re-use	0	0	0	0	1	0	4
Are conversion/installation included in design?	0	0	0	1	0	0	3
Multiple installations	0	0	0	0	0	1	5
Application designed to facilitate change by the user	0	0	0	0	0	1	5
						Sigma (F)	52
Complexity adjustment factor	0.65 + 0.01 * Sigma (F) =			1.17			

Estimated FPs Approach

Assuming $\sum_i F_i = 52$

$$FP = UFC * [\ 0.65 + 0.01 * \sum_i F_i\ ]$$

$$FP = 342 * 1.17 = 400$$

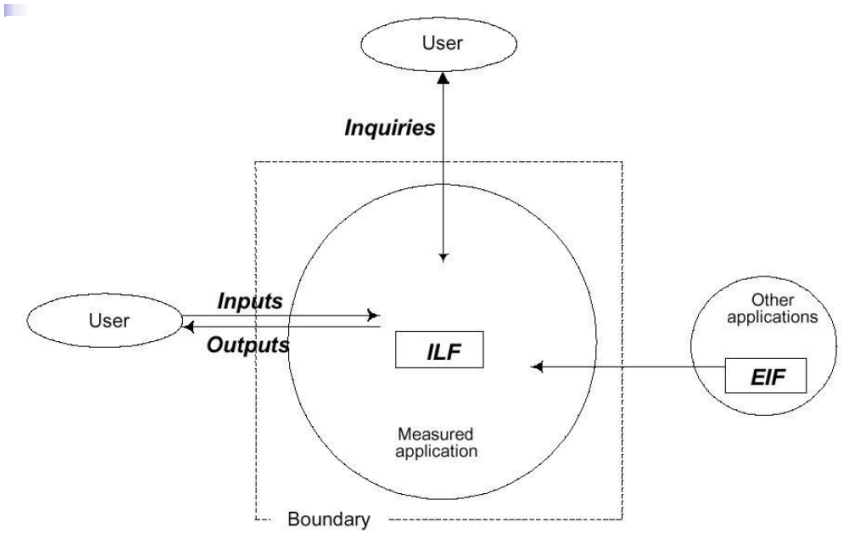
Complexity adjustment factor = 1.17

FPs Approach

Assuming
- Estimated FP = 400
- Organisation average productivity (similar project type) = 6.5 FP/p-m (person-month)
- Burdened labour rate = 8000 $/p-m

Then
- Estimated effort = 400/6.5 = (61.53) = 62 p-m
- Cost per FP = 8000/6.5 = 1231 $/FP
- Project cost = 8000 * 62 = 496000 $

FP Components



FP Example

STOCK CONTROL SYSTEM - estimating the time needed to develop application

Let's imagine a company which sells goods on the phone - if agents call the customers, customers call the agents, and so on - business operates successfully, but there comes a time for putting the whole in order. There occurs a need for developing a system able to control the whole stock, from orders to payments. Our thing is to estimate how complex such system can be and - after that - try to predict how long it would take to develop it.

FP Example

- **External Inputs -** customer, order, stock, and payment details. There are four things we need to consider.
- **External Outputs -** customer, order, and stock details, and credit rating. Once again, there are four things to consider.
- **External Inquiries -** the system is requested for three things, which are customer, order, and stock details.
- **External Interface Files -** there's no EIFs to consider.
- **Internal Logical Files -** finally, the four elements belong to the last group. Customer, and good files, and customer, and good transaction files.

FP Example

Category	Multiplier	Weight
EI	4	3
EO	4	4
EQ	3	3
ILF	3	7

*4*3+4*4+3*3+3*7=58 [Function Points]*

Omit additional technical complexity factors, so the only thing left to do is to check how long it takes to produce 58 function points. Some sources prove that one function point is an equivalent of eight hours of work in C++ language.

*58*8=464 [hours]*
The estimate for developing the application would take about 464 hours of work.