605.401
Foundations of Software Engineering

Function Point & COCOMO II Estimating Examples

Introduction

This document provides a brief tutorial on the Function Point and COCOMO II estimating techniques. It provides some basic background information on each technique as well as an example of each technique. All of the variations of these two techniques are not included here, as we just want to convey the basics and provide concrete cases so that readers will have a general idea of how the techniques are applied in practice. The references may be consulted for more detailed information.

Function Point Estimating Model

Estimating using function points involves using the requirements for a software product as a key input and then determining a function point count for the software product to be built. The function point count is an abstract measure of product size. It is generally preferred over lines of code as a size measure, because it has been shown to be a more meaningful and reliable measure of the work required to build a software product. Once a function point count is arrived at, it is then correlated with either industry statistics or organizational statistics to estimate effort, cost, or schedule.

There are five basic steps in arriving at a function point count:

1. Determine the type of function point count
2. Determine the application boundary
3. Calculate the unadjusted function point count
4. Determine the Value Adjustment Factor (VAF)
5. Calculate the adjusted function point count

In practice, an organization may just complete the first three steps and use the unadjusted function point count for estimating purposes.

The first step determines the type of function count that will be calculated. There are three types of function point counts: a count for new product development, a count for enhancements to an existing product, and a count for an existing production application. This document will illustrate calculating a function point count for a new product.

The second step determines the application boundary. It's necessary to know what functionality will be internal to the product being counted and also how other external products may interact with the product being counted, since external interactions will impact the function point count.

The third step calculates the unadjusted function point count for the product, and that's where all the work takes place.

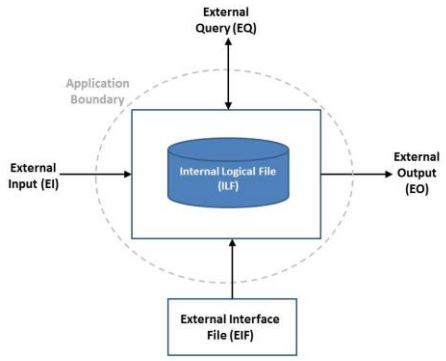
605.401
Foundations of Software Engineering

Function Point & COCOMO II Estimating Examples

The procedure requires making counts of five functional components: the number of external inputs (EI), the number of external outputs (EO), the number of external queries (EQ), the number of internal logical files (ILF), and the number of external interface files (EIF). The definitions of these components are as follows:

- **External Input (EI)** – An elementary process that has the purpose of processing data that enters the application boundary and either maintains the data contained in an ILF or controls the behavior of the application.
- **External Output (EO)** – an elementary process of the application that presents data to a user. At least one of the following logical steps must be part of the process: calculation, data derivation, update of at least one ILF, control behavior of the application.
- **External Query (EQ)** – an elementary process that retrieves data from at least one ILF or EIF for presentation to a user. Cannot include calculations, derived data, ILF file updating, or control application behavior.
- **Internal Logical File (ILF)** – a related set of logical data maintained within the application by an elementary process of the application.
- **External Interface File (EIF)** – a related set of logical data that is maintained within another application and is referenced by this application.

The schematic below illustrates the concept of application boundary and the five function point components:



Each of the counts is then rated as being simple, average, or complex, and a function point value is then calculated, as indicated in the table below. The point values for the five components are then summed to get the unadjusted function point count.

Function Point Values

Component	Simple	Average	Complex	Points (P_i)
External Inputs (EI)	___ x 3	___ x 4	___ x 6	
External Outputs (EO)	___ x 4	___ x 5	___ x 7	
External Queries (EQ)	___ x 3	___ x 4	___ x 6	
Internal Logical Files (ILF)	___ x 7	___ x 10	___ x 15	

605.401
Foundations of Software Engineering

Function Point & COCOMO II Estimating Examples

External Interface Files (EIF)	___ x 5	___ x 7	___ x 10	
			Sum	

There are guidelines for how to determine whether a component is simple, average, or complex. They are rather involved and are based upon the number of files and data elements associated with the components, and we won't go into that for purposes of this presentation.

Let's take an example to see how this works. Here's a set of functional requirements for a new software product…an employee information system. The system shall perform the following functions:

- Store and maintain the following employee information – name, employee number, job code, street address, city, state, zip code, date of birth, phone number, department code, and date of last update.
- Provide the ability to add new employees, update employee information, and delete employees.
- Provide a scheduled weekly report for all employees whose information has been changed in the last seven days. The report shall contain heading information, the reporting period, employee number, and employee name.
- Provide a mechanism for a user to view an employee's data
- Provide a mechanism to require a user id and password to access the application. This is done via an interface to the existing security application.

Let's start with the files. There is a single internal file (ILF) maintained by the application. The file maintains 11 pieces of information. According to the function point complexity guidelines this is a simple ILF, so it will account for 7 function points. There is a single EIF maintained by the security application. It also qualifies as a simple EIF, so it will account for 5 function points. There are three EI processes: add employee information, update employee information, and delete employee information. Per the complexity guidelines, they all qualify as simple, so they will account for a total 9 function points. There is one EO, a user browsing employee information. It is also simple, so it will account for 4 function points. There are two EQs, one for the weekly report and one for a logon function that must query the security application. They are both low complexity, so they account for a total of 6 function points.

Component	Simple	Average	Complex	Points (P_i)
External Inputs (EI)	3 x 3	___ x 4	___ x 6	9
External Outputs (EO)	1 x 4	___ x 5	___ x 7	4
External Queries (EQ)	6 x 3	___ x 4	___ x 6	6
Internal Logical Files (ILF)	1 x 7	___ x 10	___ x 15	7
External Interface Files (EIF)	1 x 5	___ x 7	___ x 10	5
			Sum	31

605.401
Foundations of Software Engineering

Function Point & COCOMO II Estimating Examples

The function point counts for each of the components is then summed to get a total unadjusted function point count of 31. This is where the function point process ends for some organizations. Then, if the organization has productivity statistics on how many function points per person-month can be developed, an effort estimate can be developed. For example, a productivity rate of 16 function points per person-month would yield an effort estimate of about two person-months for this project. In practice, productivity rates can vary substantially based upon type, size, and complexity of a project, so different productivity rates would need to be used based on these characteristics.

The fourth step in the process, optionally used to refine the basic function point count, is to calculate a Value Adjustment Factor (VAF) that is based on fourteen characteristics of the product to be developed. These include things like performance requirements, whether the product will be installed at multiple facilities, and so forth.

The fifth step is to multiply the unadjusted function point count by the VAF to get a refined count.

For purposes of this document these two steps won't be discussed.

COCOMO II Early Design Model

In the COCOMO II Early Design Model, the nominal effort, in units of person-months, is estimated based upon some size estimate for the software product, raised to some power, B, as illustrated in the following equation:

$$\text{Effort}_{\text{nominal}} = 2.94 * \text{Size}^{\text{B}}$$

The size estimate is in units of thousands of lines of source code. In practice, COCOMO II uses an initial size estimate based upon an unadjusted function point count, and then converts the function point measure to equivalent lines of code, according to the following table:

605.401
Foundations of Software Engineering

Function Point & COCOMO II Estimating Examples

Language	SLOC / UFP
Ada	71
AI Shell	49
APL	32
Assembly	320
Assembly (Macro)	213
ANSI/Quick/Turbo Basic	64
Basic - Compiled	91
Basic - Interpreted	128
C	128
C++	29
ANSI Cobol 85	91
Fortan 77	105
Forth	64
Jovial	105
Lisp	64
Modula 2	80
Pascal	91
Prolog	64
Report Generator	80
Spreadsheet	6

SOURCE: COCOMO II Model Definition Manual

For example, an application that will be developed in the C++ language and has an unadjusted function point count of 500 will have an estimated size of 14,500 lines of code.

The exponent, B, is a parameter that is based on five product and process characteristics, called *scale factors*:

- PREC: Precedentedness of the product
- FLEX: Development flexibility
- RESL: Architecture/Risk resolution
- TEAM: Project team cohesion
- PMAT: Process maturity

The value of B is computed from the following formula:

$$B = 1.01 + .01 * \sum_{i=1}^{i=5} W_i$$

605.401
Foundations of Software Engineering

Function Point & COCOMO II Estimating Examples

Each scale factor, W_i, is given a rating from *Very Low* to *Extra High*, and then is assigned a weight from 0-5 based upon its rating. A rating of *Extra High* is assigned a weight of 0 and a rating of *Very Low* is assigned a weight of 5. Ratings are determined according to the table below.

Scale Factors (*Wi*)	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	throughly familiar
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
RESL[a]	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
PMAT	Weighted average of "Yes" answers to CMM Maturity Questionnaire					

SOURCE: COCOMO II Model Definition Manual

[a] % of significant module interfaces specified, % of significant risks eliminated

Let's take an example. Our software product has an unadjusted function point count of 500 and will be developed using the C++ language. That equates to a size estimate of 14,500 lines of code. We'll assume nominal ratings for the five scale factors, so $\sum_{i=1}^{i=5} W_i$ = 15, and B = 1.16. Therefore, nominal effort is

$$\text{Effort}_{\text{nominal}} = 2.94 * (14.5)^{1.16} = 65.4 \text{ person-months}$$

This effort estimate includes work to be done from the design phase in a project to the end of the project. Using this estimate a nominal schedule estimate can be made using the following formula:

$$\text{Duration} = 3*\text{Effort}^{.33 + .2*(B - 1.01)} = 3*(65.4)^{.36} = 13.5 \text{ months}$$

If desired, to refine the effort estimate, the nominal effort can be adjusted by a multiplier, M, based on seven project and process attributes:

$$M = PERS * RCPX * RUSE * PDIF * PREX * FCIL * SCHED$$

where

PERS:	Personnel capability
RCPX:	Product reliability & complexity
RUSE:	Reuse required
PDIF:	Platform difficulty
PREX:	Personnel experience

Function Point & COCOMO II Estimating Examples

FCIL:	Support Facilities
SCHED:	Schedule

The multiplier, M, is then applied to the nominal effort estimate as follows:

$$Effort_{adjusted} = Effort_{nominal} * M$$

An estimate of adjusted project duration, in calendar months, can then be made by applying the same duration formula illustrated above, substituting the adjusted effort for the nominal effort.

References

COCOMO II Model Definition Manual, University of Southern California. Available online at http://sunset.usc.edu/research/COCOMOII/Docs/modelman.pdf

Function Point Counting Practices Manual, International Function Point User Group (IFPUG)