**Table of Content**

[1. Economics 1](#_Toc45723763)

[2. Software Engineering Economics 1](#_Toc45723764)

[3. COCOMO Model (Constructive Cost Model) 1](#_Toc45723765)

[3.1 Basic COCOMO Model 1](#_Toc45723766)

[3.1.1 Basic COCOMO Equations 1](#_Toc45723767)

[3.1.2 The Development Modes: Project Characteristics 2](#_Toc45723768)

[i. Organic Projects 2](#_Toc45723769)

[ii. Semi-detached Projects 2](#_Toc45723770)

[iii. Embedded Projects 2](#_Toc45723771)

[3.1.3 Example 2](#_Toc45723772)

[i. Effort Estimation 2](#_Toc45723773)

[ii. Duration Estimation 2](#_Toc45723774)

[iii. Persons Estimation 2](#_Toc45723775)

[3.2 Intermediate Model 3](#_Toc45723776)

[3.2.1 Classification of Cost Drivers and their attributes: 3](#_Toc45723777)

[3.2.2 Example 5](#_Toc45723778)

[3.3 The Detailed/Advanced Model 5](#_Toc45723779)

[*Step 1:* You have to compute the count-total which will be used to define the complexity of a project. You will do that by completing the table below: 6](#_Toc45723780)

[*Step 2:*  You have to find the complexity adjustment values based on responses to the questions below: 6](#_Toc45723781)

[*Step 4:* Final Step is to select complexity of the software project: 8](#_Toc45723782)

[4.1 Counting Function Point (FP): 9](#_Toc45723783)

[4.2 Example 10](#_Toc45723784)

[5. Object Point 10](#_Toc45723785)

[5.1 Example 10](#_Toc45723786)

[5.2 New Object Point 11](#_Toc45723787)

[6. Story Point 11](#_Toc45723788)

[7. COCOMO-II 13](#_Toc45723789)

[7.1 Sub-Models 13](#_Toc45723790)

[7.2 Stages of COCOMO II: 14](#_Toc45723791)

[7.3 Cost Drivers 14](#_Toc45723792)

[7.4 Scale Drivers 15](#_Toc45723793)

[References 15](#_Toc45723794)

# Economics

It is the **study of production, distribution and consumption of goods and services.** In short it is how people make decisions in resource-limited situations.

# Software Engineering Economics

Software Engineering Economics is concerned with **making decisions within the business context to align technical decisions with the business goals of an organization to develop the required software**.

COCOMO Calculator is at <http://groups.umd.umich.edu/cis/tinytools/cocomo.html>

Baker-Mike Calculator - <http://groups.umd.umich.edu/cis/course.des/cis525/js/f00/baker/cocomo.html>

Kutcher Calculator - <http://groups.umd.umich.edu/cis/course.des/cis525/js/f00/kutcher/kutcher.html>

Mohamed Calculator - <http://groups.umd.umich.edu/cis/course.des/cis525/js/f00/gamel/cocomo.html>

# COCOMO Model (Constructive Cost Model)

It is a **cost estimation model** proposed by **Berry Boehm in 1981**. COCOMO predicts the effort and schedule for a software product development based on inputs relating to the size of the software and a number of cost drivers that affect productivity. COCOMO has three different models that reflect the complexity:

* 1. **The Basic Model**
  2. **The Intermediate Model**
  3. **The Detailed Model**

## Basic COCOMO Model

Basic COCOMO estimates the software development effort using **LOC (Lines of Codes).**It is based on size of the project. The size of the project may vary depending upon the function points.It is good for quick, early, rough order of magnitude estimates of software costs. It does not account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and other project attributes known to have a significant influence on software costs, which limits its accuracy.

### Basic COCOMO Equations

The basic COCOMO equations are:

E=ab (**KLOC** or KDSI) b b

D=cb (E) db

P=E/D

Where,

**E**- Effort applied in person-months,

**D-** Development time in months,

**P-** Number of people required,

**KLOC/KDSI-** estimated delivered lines of code for the projects,

**ab, bb, cb** and **db**- Coefficients

Figure 1: Coefficients

Figure 1 shows the constant values that have been calculated from various projects. The basic cocomo gives the magnitude of cost of the project. It varies depending upon size of the project. The various classes of software projects are :

### The Development Modes: Project Characteristics

COCOMO applies to the three classes of software projects:

## Organic Projects

It is related to small projects with small development team. Team members should have an experience to work with less rigid requirements. Such as Payroll project.

## Semi-detached Projects

It is related to the projects of intermediate size and complexity with mixed experience team. Such projects may have mix of rigid and less than rigid requirements. Example: Banking System

## Embedded Projects

It is concerned with the projects that must be developed under tight hardware, software and operational constraints.

### Example

Consider a software project using semi-detached mode with 30,000 lines of code. We will obtain estimation for this project as follows

### Effort Estimation

E=ab (KLOC)bb

E= 3.0(30)1.12 where LOC= 30000= 30KLOC

E= 135 person-month

### Duration Estimation

D=Cb(E)db

D= 2.5(135)0.35

D= 14 months

### Persons Estimation

P=E/D

P=135/14

P=10 persons approx.

The accuracy of this model is limited because it does not consider certain factors for software cost estimation. These factors are hardware constraints, personal quality and experience, modern techniques and tools. The estimates are within a factor of 1.3 only 29% of the time and within the factor of 2 only 60 % of time.

## Intermediate Model

It is an extension of Basic COCOMO. This make use of set of cost driver attributes to compute software cost. It includes subjective assessments of product, hardware, personnel and project attributes. It can be applied to entire software product for easy and rough cost estimation during early stage and for obtaining more accurate cost estimation.

## Classification of Cost Drivers and their attributes:

**(i) Product attributes**

* Required software reliability extent (RELY)
* Size of the application database (DATA)
* The complexity of the product (CPLX)

**(ii) Hardware/ Computer attributes**

* Run-time performance constraints (TIME)
* Memory constraints (STOR)
* The volatility of the virtual machine environment (VIRT)
* Required turnaround time (TURN)

**(iii) Personnel attributes**

* Analyst capability (ACAP)
* Software engineering capability (PCAP)
* Applications experience (AEXP)
* Virtual machine experience (VEXP)
* Programming language experience (LEXP)

**(iv) Project attributes**

* Use of software tools (TOOL)
* Modern programming practices (MODP)
* Required development schedule (SCED)

Each of the 15 attributes receives a rating on a six-point scale that ranges from "very low" to "extra high" (in importance or value). An effort multiplier from the table below applies to the rating. The product of all effort multipliers results in an *effort adjustment factor (EAF)*. Typical values for EAF range from 0.9 to 1.4.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Cost Drivers | **Ratings** | | | | | |
| Very Low | Low | Nominal | High | Very High | Extra High |
| **Product attributes** |  |  |  |  |  |  |
| Required software reliability | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 |  |
| Size of application database |  | 0.94 | 1.00 | 1.08 | 1.16 |  |
| Complexity of the product | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 | 1.65 |
| **Hardware attributes** |  |  |  |  |  |  |
| Run-time performance constraints |  |  | 1.00 | 1.11 | 1.30 | 1.66 |
| Memory constraints |  |  | 1.00 | 1.06 | 1.21 | 1.56 |
| Volatility of the virtual machine environment |  | 0.87 | 1.00 | 1.15 | 1.30 |  |
| Required turnabout time |  | 0.87 | 1.00 | 1.07 | 1.15 |  |
| **Personnel attributes** |  |  |  |  |  |  |
| Analyst capability | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 |  |
| Applications experience | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 |  |
| Software engineer capability | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 |  |
| Virtual machine experience | 1.21 | 1.10 | 1.00 | 0.90 |  |  |
| Programming language experience | 1.14 | 1.07 | 1.00 | 0.95 |  |  |
| **Project attributes** |  |  |  |  |  |  |
| Application of software engineering methods | 1.24 | 1.10 | 1.00 | 0.91 | 0.82 |  |
| Use of software tools | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 |  |
| Required development schedule | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 |  |

The Intermediate Cocomo formula now takes the form:

**E=*ai*(KLOC)*(bi)*(EAF)**

where E is the effort applied in person-months, **KLOC** is the estimated number of thousands of delivered lines of code for the project, and **EAF** is the factor calculated above. The coefficient **ai** and the exponent **bi** are given in the next table.

|  |  |  |
| --- | --- | --- |
| **Software project** | **ai** | **bi** |
| Organic | 3.2 | 1.05 |
| Semi-detached | 3.0 | 1.12 |
| Embedded | 2.8 | 1.20 |

The Development time **D** calculation uses **E** in the same way as in the Basic COCOMO.

The intermediate COCOMO equation is:

E = *ai*(KLOC)*bi \** (EAF) person-month

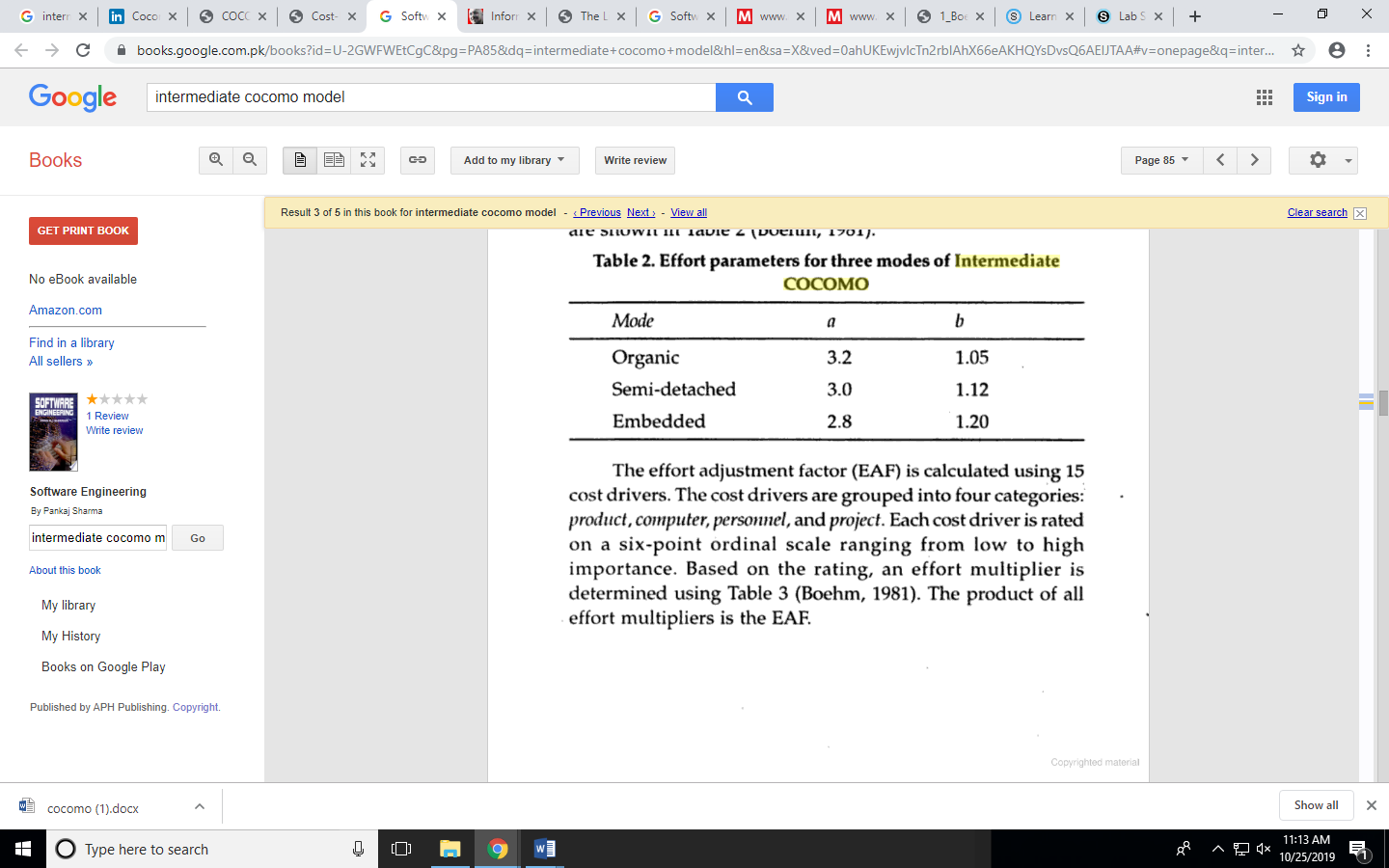


Figure 2: Effort Parameters for three modes of Intermediate COCOMO

The duration and person estimate are same as in basic COCOMO model i.e.

D=Cb(E)db months

P=E/D persons

This model estimation is limited within 20% of actual and 68% of the time. Its effort multiplies are not dependent on phases and a product with many components is difficult to estimate.

## Example

Consider a project having 30,000 lines of code which is an embedded software with critical area hence reliability is high. The estimation can be

E=ai(KLOC)bi \* EAF

As reliability is high, EAF=1.15 (product attribute), DATA=1.16, CPLX=1.65, ACAP=0.71

ai=2.8 , bi= 1.20 (for embedded software)

E=2.8(30)1.20 x 1.15 x 1.16 x 1.65 x 0.71

=191 person-month

D = cb(E)db = 2.5(191)0.32

= 13 months

P=E/D =>191/13 => 15 persons approx.

## The Detailed/Advanced Model

The detail model uses the same equations for estimation as the intermediate model. It can estimate the effort (E), duration (D) and persons (P) of each development phases, subsystems and modules. This model allows the experimentation with different development strategies. Four phases used in it are:

1. Requirements planning and product design (RPD)
2. Detailed design (DD)
3. Code and unit test (CUT)
4. Integrate and Test (IT)

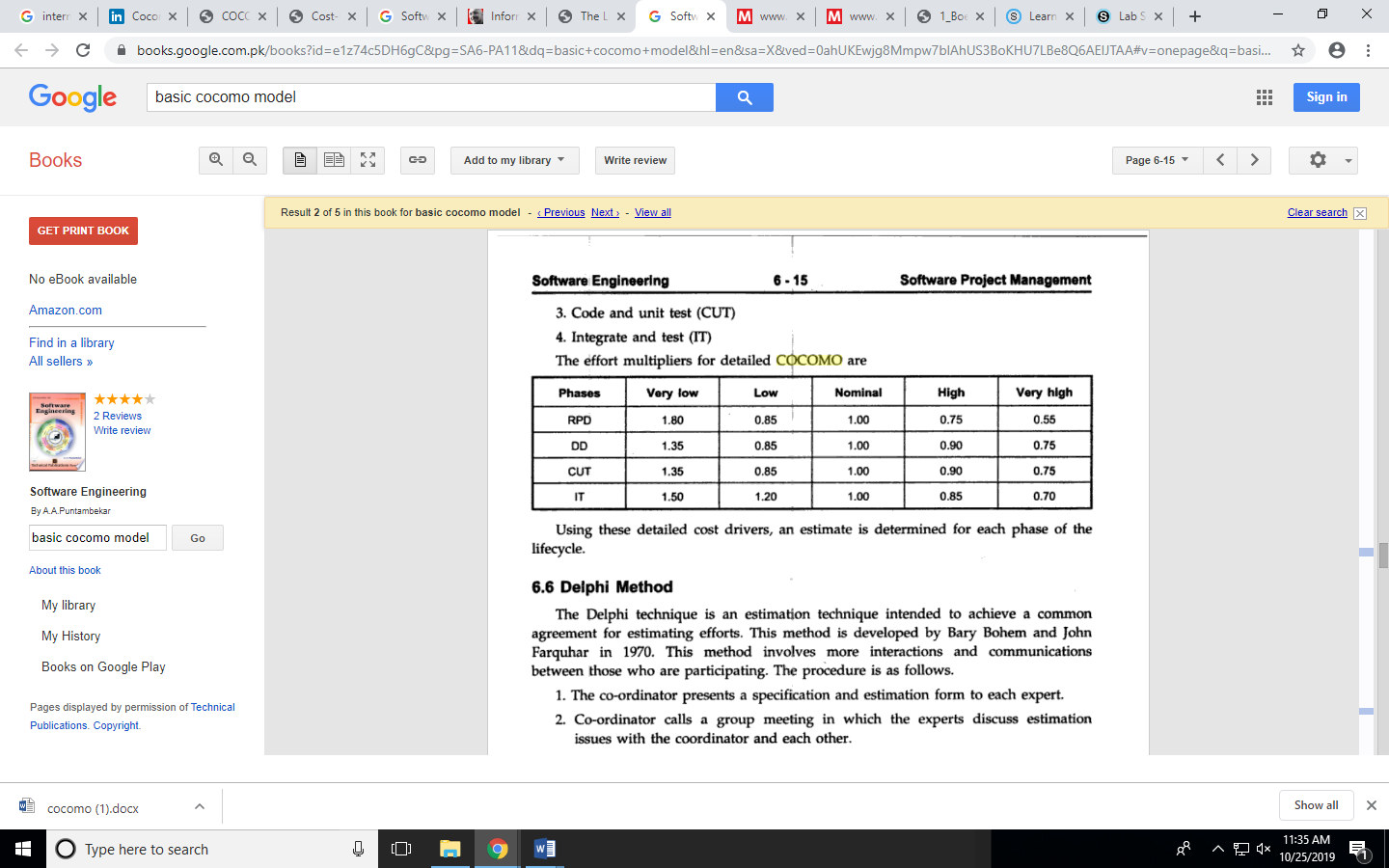


Figure 3: Effort Multipliers for Detailed COCOMO

**Function Point**

## *Step 1:* You have to compute the count-total which will be used to define the complexity of a project. You will do that by completing the table below:

Top of Form

**Information Domain Values**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Measurement Parameter** | **Count** |  | **Simple** | **Average** | **Complex** |  | **Total** |
| **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 external interfaces** |  | **X** | **5** | **7** | **10** | **=** |  |
| **Count=Total** |  |  |  |  |  |  |  |

Bottom of Form

## *Step 2:*  You have to find the complexity adjustment values based on responses to the questions below:

**Complexity Weighting Factors   
// heading of the second table Rate each factor on a scale of 0 to 5:   
(0 = No influence,   1 = Incidental,   2 = Moderate,   3 = Average,   4 = Significant,   5 = Essential):**

|  |
| --- |
| **Question** |
| **1. Does the system require reliable backup and recovery?** |
| **2. Are data communications required?** |
| **3. Are there distributed processing functions?** |
| **4. Is performance critical?** |
| **5. Will the system run in an existing, operational heavily utilized environment?** |
| **6. Does the system require on-line data entry?** |
| **7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?** |
| **8. Are the master file updated on-line?** |
| **9. Are the inputs, outputs, files, or inquiries complex?** |
| **10. Is the internal processing complex?** |
| **11. In 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?** |
| **Total** |

**The Function Points is:** 

*Step 3:* You have to find LOC (Lines of Code), and you do this by choosing a programming language that you will using when developing a project:

|  |  |  |
| --- | --- | --- |
| **Programming Language** | **LOC/FP (average)** | **Select** |
| **Assembly Language** | **320** |  |
| **C** | **128** |  |
| **COBOL** | **105** |  |
| **Fortran** | **105** |  |
| **Pascal** | **90** |  |
| **Ada** | **70** |  |
| **Object-Oriented Languages** | **30** |  |
| **Fourth Generation Languages (4GLs)** | **20** |  |
| **Code Generators** | **15** |  |
| **Spreadsheets** | **6** |  |
| **Graphical Languages (icons)** | **4** |  |

      **LOC/F** **P:** 

## *Step 4:* Final Step is to select complexity of the software project:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Software Project** | **ab** | **bb** | **cb** | **db** | **Select** |
| **Organic** | **2.4** | **1.05** | **2.5** | **0.38** |  |
| **Semi-detached** | **3.0** | **1.12** | **2.5** | **0.35** |  |
| **Embedded** | **3.6** | **1.20** | **2.5** | **0.32** |  |

1. **Effort (E) = ab(KLOC)bb =**      **Duration (D) = cb(E)db =**   
   [Function Point (FP)](https://www.geeksforgeeks.org/software-engineering-functional-point-fp-analysis/)

Function point analysis (FPA) is a methodology for measuring software productivity and the cost associated with the development and maintenance. One function point (FP) is one end-user requested business function. The following defines the five characteristics of function points:

* + - * **External Inputs**: these are end-user actions such as putting in a login or executing a mouse click.
      * **External Outputs**: the system provides the end-user output or interface such as a GUI display or items in a report.
      * **Logical Internal Files**: these files are the master or transaction files that the system interacts with during its session.
      * **External Interface Files**: unlike logical internal files, where the application uses solely for its purpose, these files are or databases are shared with other applications or systems.
      * **External Inquiries**: this function is initiated by the end-user. For example, the end-user wishes to submit a query to a database or requests on-line help. In any case the developer provides a means for the end-user to "search" for answers.

## ****Counting Function Point (FP):****

* **Step-1:**

F = 14 \* scale

Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF). Below table shows scale:

0 - No Influence

1 - Incidental

2 - Moderate

3 - Average

4 - Significant

5 - Essential

* **Step-2:** Calculate Complexity Adjustment Factor (CAF).

CAF = 0.65 + ( 0.01 \* F ) = 0.65 + (0.01\*42) = 0.65+0.42= 1.07 .65+.7=1.35

* **Step-3:** Calculate Unadjusted Function Point (UFP).

**Table 1: Function Point Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function Units** | **Low** | **Average** | **High** |
| EI | 3 | 4 | 6 |
| EO | 4 | 5 | 7 |
| EQ | 3 | 4 | 6 |
| ILF | 7 | 10 | 15 |
| EIF | 5 | 7 | 10 |

Total Count (UFP) = 314 424 616

Multiply each individual function point to corresponding values in TABLE.

* **Step-4:** Calculate Function Point.
* FP = UFP \* CAF = 314 \* 1.07 = 335.9 314\*1.35 = 423.9 (Simple)

FP = UFP \* CAF = 424 \* 1.07 = 453.7 424\*1.35 = 572.4 (Average)

FP = UFP \* CAF = 616 \* 1.07 = 659.1 616\*1.35 = 831.6 (Complex)

* 1. Example

Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.

User Input = 50

User Output = 40

User Inquiries = 35

User Files = 6

External Interface = 4

**Explanation:**

* **Step-1:** As **complexity adjustment factor** is **average** (given in question), hence,
* Scale = 3.

F = **14 \* 3 = 42**

* **Step-2:**

CAF = 0.65 + ( 0.01 \* 42 ) = 1.07

* **Step-3:** As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.

UFP = (50\*4) + (40\*5) + (35\*4) + (6\*10) + (4\*7) = 628

* **Step-4:**

Function Point = **628 \* 1.07 = 671.96**

**We have 45 inputs, 30 outputs, 24 inquiries. 9 files and 27 interfaces. Such that Complexity is distributed. If we have 5 Complexity Adjustment Factors to No Influence and remaining are Average.**

**Table 1: Function Point Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Units** | **Low** | **Average** | **High** | **Total** |
| EI | 15 x 3 | 15 x 4 | 15 x6 | 195 |
| EO | 10 x 4 | 10 x 5 | 10 x 7 | 160 |
| EQ | 8 x 3 | 8 x 4 | 8 x 6 | 104 |
| ILF | 3 x 7 | 3 x 10 | 3 x 15 | 96 |
| EIF | 9 x 5 | 9 x 7 | 9 x 10 | 198 |

Total Count (UFP) = 753

Multiply each individual function point to corresponding values in TABLE.

F = 5 x 0 + 9 x 3 = 27

CAF = 0.65 + (0.01 x F) = 0.65 + 0.01 x 27 = 0.65 + 0.27 = 0.92

* **Step-4:** Calculate Function Point.
* FP = UFP \* CAF = **753 \* 0.92 = 692.76**
* **If we have 5 Complexity Adjustment Factors to No Influence and remaining are Essential.**

Total Count (UFP) = 753

Multiply each individual function point to corresponding values in TABLE.

F = 5 x 0 + 9 x 5 = 45

CAF = 0.65 + (0.01 x F) = 0.65 + 0.01 x 45 = 0.65 + 0.45 = 1.10

* **Step-4:** Calculate Function Point.
* FP = UFP \* CAF = **753 \* 1.10 = 828.3**

**If we have 3 types of Domain Values – Pessimistic, Optimistic and Average**

**We have [35, 45, 55] inputs, [24, 30, 35] outputs, [20, 24, 25] inquiries. [8, 9, 11] files and [20, 27, 33] interfaces. Such that Complexity is distributed. If we have 5 Complexity Adjustment Factors to No Influence and remaining are Average.**

**Domain Value = (Op + 4 x Avg + Pess) / 6 =**

**Input Domain Value = (35 + 4 x 45 + 55) / 6 = (35 + 180 + 55)/6 = 270/6 = 45**

**Output Domain Value = (24 + 4 x 30 + 35) / 6 = (24 + 120 + 35)/6 = 179/6 = 29.83**

**Inquiries Domain Value = (20 + 4 x 24 + 25) / 6 = (20 + 96 + 25)/6 = 141/6 = 23.5**

**Files Domain Value = (8 + 4 x 9 + 11) / 6 = (8 + 36 + 11)/6 = 55/6 = 9.16**

**Interfaces Domain Value = (20 + 4 x 27 + 33) / 6 = (20 + 108 + 33)/6 = 161/6 = 26.83**

**Table 1: Function Point Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Units** | **Low** | **Average** | **High** | **Total** |
| EI | 15 x 3 | 15 x 4 | 15 x6 | 195 |
| EO | 9.94 x 4 | 9.94 x 5 | 9.94 x 7 | 159.05 |
| EQ | 7.83 x 3 | 7.83 x 4 | 7.83 x 6 | 101.8 |
| ILF | 3.05 x 7 | 3.05 x 10 | 3.05 x 15 | 97.6 |
| EIF | 8.94 x 5 | 8.94 x 7 | 8.94 x 10 | 196.7 |

Total Count (UFP) = 750.15

Multiply each individual function point to corresponding values in TABLE.

F = 5 x 0 + 9 x 3 = 27

CAF = 0.65 + (0.01 x F) = 0.65 + 0.01 x 27 = 0.65 + 0.27 = 0.92

* **Step-4:** Calculate Function Point.
* FP = UFP \* CAF = **750.15 \* 0.92 = 690.14**

## Object Point

Object Point is similar to FP. It computes the **number of screens and classify them as simple, medium, complex**. It computes the **number of reports and classify them as simple, medium, complex**. It counts **the number of modules that have to be developed** **as complex**. It uses weight matrices to sum the values above, taking into account reused code.

**Table 2: Object Point Data**

|  |  |  |  |
| --- | --- | --- | --- |
| **Object Type** | **Complexity Weights** | | |
| **Simple** | **Medium** | **Difficult** |
| **Screen** | 1 | 2 | 3 |
| **Report** | 2 | 5 | 8 |
| **3GL Components** | - | - | 10 |

## Example

Screen = 15 Screens (5 each)

Reports = 10 Reports (Medium)

3GL Components = 15 3GL components

Evaluating the above values in Table 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Object Type** | **Complexity Weights** | | |  |
|  | **Simple** | **Medium** | **Difficult** | Total |
| **Screen** | 1 x 5 + | 2 x 5 + | 3 x 5 | 30 |
| **Report** | 2 x 0 + | 5x 10 + | 8x 0 | 50 |
| **3GL Components** | - | - | 10 x 15 | 150 |
| **Object Point =** | | | | **230** |

## New Object Point

New object point (NOP) = (Object Point) x (100 – Reuse Percentage/ 100)

Having **Reuse Percentage: 33%,** Therefore:

New object point (NOP) = (230) x (100-33%x) / 100)

New object point (NOP) = (230) x (0.67)

**New object point (NOP) = 154.1**

**We have 45 inputs, 30 outputs, 24 inquiries. 9 files and 27 interfaces. Such that Complexity is distributed. Reuse % = 40%.**

Screens = 75 Screens (25 each)

Reports = 33 Reports (11 each)

3GL Components = 27 3GL components

Evaluating the above values in Table 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Object Type** | **Complexity Weights** | | |  |
|  | **Simple** | **Medium** | **Difficult** | Total |
| **Screen** | 1 x 25 + | 2 x 25 + | 3 x 25 | 150 |
| **Report** | 2 x 11 + | 5 x 11 + | 8x 11 | 165 |
| **3GL Components** | - | - | 10 x 27 | 270 |
| **Object Point =** | | | | **585** |

## New Object Point

New object point (NOP) = (Object Point) x (100 – Reuse Percentage/ 100)

Having **Reuse Percentage: 40%,** Therefore:

New object point (NOP) = (585) x (100-40% / 100)

New object point (NOP) = (585) x (0.6)

**New object point (NOP) = 351**

## Story Point

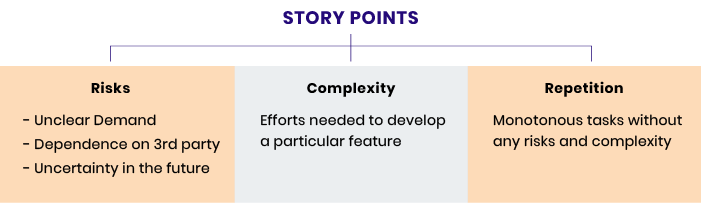
A story point is a metric used in agile project management and development to estimate the difficulty of implementing a given user story, which is an abstract measure of effort required to implement it. In simple terms, a story point is a number that tells the team about the difficulty level of the story. Difficulty could be related to complexities, risks, and efforts involved. The Story Points approach uses historical data to compare features of one project to features of a previous similar project to generate a precise estimate.

The general characteristics of story points include:

* Story points represent the total amount of work required to fully implement a user story.
* The stories are estimated independently by team members and the team drives toward a consensus opinion.
* If a user story is too large to be implemented in one iteration it needs to be broken down into two or more smaller stories.
* Many of the estimating models are designed as games that are interesting and engaging for the project team.

**Step 1 — Identify a Base Story**

Story Points in agile are a complex unit that includes three elements: risk, complexity and repetition.

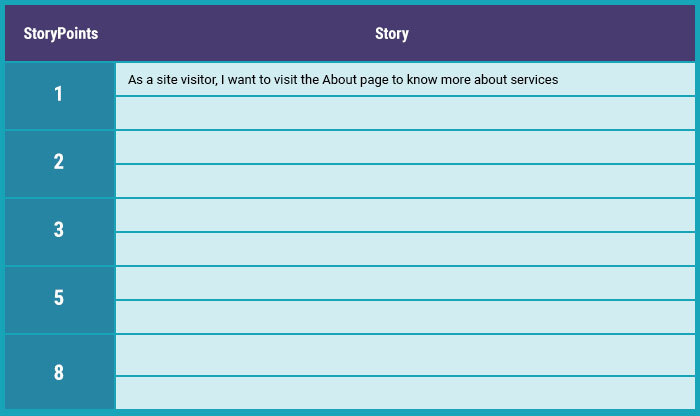


To find our Base Story, we search for one elementary task that corresponds to internal standards of Definition of Done for User Stories and assign it one Story Point. This will be our Base Story.

**Step 2 — Create a Matrix for Estimation**

There are two types of scales used for creating estimation matrices: the linear scale (1,2,3,4,5,6,7…) and Fibonacci sequence numbers (0.5, 1, 2, 3, 5, 8, 13 …), T-Shirt Sizing (X-Small, Small, Medium, Large, Extra-Large).

Base Story is already in this matrix in the first row with a value of one Story Point as shown in Figure 4.



**Figure 4: Story Point Metric**

There can be several stories in one row as shown in Figure 5.

# what are story points

**Figure 5: Story Points with Several Stories**

## COCOMO-II

It is the revised version of the basic Cocomo (Constructive Cost Model) and is developed at University of Southern California. It is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity. It consists of three sub-models:

## Sub-Models

* **End User Programming**

Application generators are used in this sub-model. End user write the code by using these application generators.

**Example –**Spreadsheets, report generator, etc.

* **Intermediate Sector:**

It consists of:

**(a). Application Generators and Composition Aids**

This category will create largely prepackaged capabilities for user programming. Their product will have many reusable components. Typical firms operating in this sector are Microsoft, Lotus,  
Oracle, IBM, Borland, Novell.

**(b). Application Composition Sector**

This category is too diversified and to be handled by prepackaged solutions. It includes GUI, Databases, and domain specific components such as financial, medical or industrial process control packages.

**(c). System Integration**

This category deals with large scale and highly embedded systems.

* **Infrastructure Sector**

This category provides infrastructure for the software development like Operating System, Database Management System, User Interface Management System, Networking System, etc.

## Stages of COCOMO II:

**Stage-I:**  
It supports estimation of prototyping. For this it uses *Application Composition Estimation Model*. This model is used for the prototyping stage of application generator and system integration.

**Stage-II:**  
It supports estimation in the early design stage of the project, when we less know about it. For this it uses *Early Design Estimation Model*. This model is used in early design stage of application generators, infrastructure, and system integration.

**Stage-III:**  
It supports estimation in the post architecture stage of a project. For this it uses *Post Architecture Estimation Model*. This model is used after the completion of the detailed architecture of application generator, infrastructure, and system integration.

## Cost Drivers

COCOMO II has 17 cost drivers used to assess project, development environment, and team to set each cost driver. The cost drivers are multiplicative factors that determine the effort required to complete your software project. For example, if your project will develop software that controls an airplane's flight, you would set the Required Software Reliability (RELY) cost driver to Very High. That rating corresponds to an effort multiplier of 1.26, meaning that your project will require 26% more effort than a typical software project.

**Table 3 COCOMO II Cost Drivers**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| RELY | Product | Required system reliability |
| CPLX | Product | Complexity of system modules |
| DOCU | Product | Extent of documentation required |
| DATA | Product | Size of database used |
| RUSE | Product | Required percentage of reusable components |
| TIME | Computer | Execution time constraint |
| PVOL | Computer | Volatility of development platform |
| STOR | Computer | Memory constraints |
| ACAP | Personnel | Capability of project analysts |
| PCON | Personnel | Personnel continuity |
| PCAP | Personnel | Programmer capability |
| PEXP | Personnel | Programmer experience in project domain |
| AEXP | Personnel | Analyst experience in project domain |
| LTEX | Personnel | Language and tool experience |
| TOOL | Project | Use of software tools |
| SCED | Project | Development schedule compression |
| SITE | Project | Extent of multisite working and quality of inter-site communications |

## Software Scale Drivers

In the COCOMO II model, some of the most important factors contributing to a project's duration and cost are the Scale Drivers. You set each Scale Driver to describe your project; these Scale Drivers determine the exponent used in the Effort Equation.The 5 Scale Drivers are:

* Precedentedness
* Development Flexibility
* Architecture / Risk Resolution
* Team Cohesion
* Process Maturity

<http://softwarecost.org/tools/COCOMO/>

|  |  |  |
| --- | --- | --- |
| COCOMO II - Constructive Cost Model | |  | | --- | | Monte Carlo Risk | |

Auto Calculate   
**Software Size**                Sizing Method

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Unadjusted Function Points |  | Language |  | |

|  |  |
| --- | --- |
|  |  |

**Software Scale Drivers**

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | Precedentedness |  | |

|  |  |
| --- | --- |
| Development Flexibility |  |

|  |
| --- |
| **Software Cost Drivers** |
| **Product** |
| Required Software Reliability |  |

|  |  |
| --- | --- |
| Data Base Size |  |
| Product Complexity |  |
| Developed for Reusability |  |
| Documentation Match to Lifecycle Needs |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | |  | |  | | |  |  | | --- | --- | | Architecture / Risk Resolution |  | |

|  |  |
| --- | --- |
| Team Cohesion |  |

|  |
| --- |
|  |
|  |
| **Personnel** |
| Analyst Capability |  |

|  |  |
| --- | --- |
| Programmer Capability |  |
| Personnel Continuity |  |
| Application Experience |  |
| Platform Experience |  |
| Language and Toolset Experience |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | |  | | |  |  | | --- | --- | | Process Maturity |  | |

|  |
| --- |
|  |
|  |
|  |
|  |
| **Platform** |
| Time Constraint |  |

|  |  |
| --- | --- |
| Storage Constraint |  |
| Platform Volatility |  |

|  |
| --- |
| **Project** |
| Use of Software Tools |  |

|  |  |
| --- | --- |
| Multisite Development |  |
| Required Development Schedule |  |

**Maintenance**   
  
**Software Labor Rates**  
Cost per Person-Month (Dollars)

Top of Form

**Results**

Bottom of Form

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Software Development (Elaboration and Construction)**  Effort = 170.8 Person-months Schedule = 20.0 Months Cost = $170773  Total Equivalent Size = 40192 SLOC   **Acquisition Phase Distribution**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Phase | Effort (Person-months) | Schedule (Months) | Average Staff | Cost (Dollars) | | Inception | 10.2 | 2.5 | 4.1 | $10246 | | Elaboration | 41.0 | 7.5 | 5.5 | $40986 | | Construction | 129.8 | 12.5 | 10.4 | $129788 | | Transition | 20.5 | 2.5 | 8.2 | $20493 | | **Staffing Profile** |

**Software Effort Distribution for RUP/MBASE (Person-Months)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phase/Activity | Inception | Elaboration | Construction | Transition |
| Management | 1.4 | 4.9 | 13.0 | 2.9 |
| Environment/CM | 1.0 | 3.3 | 6.5 | 1.0 |
| Requirements | 3.9 | 7.4 | 10.4 | 0.8 |
| Design | 1.9 | 14.8 | 20.8 | 0.8 |
| Implementation | 0.8 | 5.3 | 44.1 | 3.9 |
| Assessment | 0.8 | 4.1 | 31.1 | 4.9 |
| Deployment | 0.3 | 1.2 | 3.9 | 6.1 |

Your output file is <http://softwarecost.org/tools/COCOMO/data/COCOMO_July_29_2020_11_20_48_386732.txt>

# References

[1] A. A. Puntambekar, *Software Engineering*, Third Revi. Technical Publications Pune, 2009.

[2] P. Sharma, *Software Engineering*. APH Pubishing Corporation, 2004.

[3] B. W. Boehm, *Software Engineering Economics*.

[4] COCOMO Tool. http://groups.umd.umich.edu/cis/course.des/cis525/js/f00/kutcher/kutcher.html