**MCA (Management )**

**(2020-22)**

**Course Code:- IT 42**

**Course Name :- Software Project Management**

**Chapter 2 :- Linear Software Project Estimation**

**Software project planning involves estimation** - your attempt to determine how much money, how much effort, how many resources, and how much time it will take to build a specific software-based system or product.

Estimation of factors such as cost, effort, risks, and resources is crucial. It gives you a fair idea of the size of the project. You can use the information about size to estimate the cost, effort, and duration of the project. This further helps plan for resources and schedule the project.

To achieve reliable cost and effort estimates, a number of options arise:

1. Delay estimation until late in the project (obviously, we can achieve 100% accurate estimates after

the project is complete!).

2. Base estimates on similar projects that have already been completed.

3. Use relatively simple decomposition techniques to generate project cost and effort estimates.

4. Use one or more empirical models for software cost and effort estimation.

Following are some points related to project estimation:

• Estimation is very difficult to do, but is often needed

• It is created, used or refined during

– Strategic planning

– Feasibility study

– Proposals

– Vendor and sub-contractor evaluation

– Project planning (iteratively)

• Basic process involves:

– Estimate the size of the product

– Estimate the effort (man-months)

– Estimate the schedule

– NOTE: Not all of these steps are always explicitly performed

Estimation – A Critical factor

In a software project, unless you are sure that your estimation is accurate, you cannot make much progress.

Estimation of following factors are essential:

– Cost,

– Effort,

– Risks

– Resources

a**) Project complexity** :- has a strong effect on the uncertainty, inherent in planning. Complexity, however, is a relative measure that is affected by familiarity with past effort. The first-time developer of a sophisticated e-commerce application might consider it to be exceedingly complex. However, a software team developing its tenth e-commerce Web site would consider such work run of the mill. A number of quantitative software complexity measures can be applied as per the need of project. Such measures are applied at the design or code level and are therefore difficult to use during Software planning (before a design and code exist). However, other, more subjective assessments of complexity (e.g., the function point complexity adjustment factors) can be established early in the planning process.

**b) Project size:-**  is another important factor that can affect the accuracy and efficacy of estimates. As size increases, the interdependency among various elements of the software grows rapidly. Problem decomposition, an important approach to estimating, becomes more difficult because decomposed elements may still be alarming. To paraphrase Murphy's Law: "What can go wrong will go wrong"- and if there are more things that can fail, more things will fail.

**c) The degree of structural uncertainty** :- has an effect on estimation risk. In this context, structure refers to the degree to which requirements have been solidified, the ease with which functions can be compartmentalized and the hierarchical nature of the information that must be processed.

**d) The availability of historical information:**  has a strong influence on estimation risk. By looking back, we can emulate things that worked and improve areas where problems arose.

**e) Risk** is measured by the degree of uncertainty in the quantitative estimates

established for resources, cost, and schedule. If project scope is poorly

understood or project requirements are subject to change, uncertainty and risk

become dangerously high.

* **2.1] Different Methods Of Cost Estimation :-**

**2.1.1]** **The Constructive Cost Model (COCOMO)**:- It is an algorithmic software cost estimation model developed by Barry W. Boehm. The model uses a basic regression formula with parameters that are derived from historical project data and current as well as future project characteristics.

Basic COCOMO computes software development effort (and cost) as a function of program size. Program size is expressed in estimated thousands of source lines of code ([SLOC](http://en.wikipedia.org/wiki/Source_lines_of_code), [KLOC](http://en.wikipedia.org/wiki/Source_lines_of_code#Related_terms)).

COCOMO applies to three classes of software projects:

**Organic projects** - "small" teams with "good" experience working with "less than rigid" requirements.

**Semi-detached projects** - "medium" teams with mixed experience working with a mix of rigid and less than rigid requirements.

**Embedded projects** - developed within a set of "tight" constraints. It is also combination of organic and semi-detached projects.(hardware, software, operational, ...)

The basic COCOMO equations take the form

Effort Applied (E) = ab(KLOC)bb[ [person-months](http://en.wikipedia.org/w/index.php?title=Person-month&action=edit&redlink=1) ]

Development Time (D) = cb(Effort Applied)db [months]

People required (P) = Effort Applied / Development Time [count]

where, KLOC is the estimated number of delivered lines (expressed in thousands ) of code for project. The coefficients & Exponent  ab, bb,  cb and db are given in the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Software project | ab | bb | cb | db |
| 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 |

Intermediate COCOMO computes software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel and project attributes. This extension considers a set of four "cost drivers", each with a number of subsidiary attributes:-

* Product attributes
  + Required software reliability
  + Size of application database
  + Complexity of the product
* Hardware attributes
  + Run-time performance constraints
  + Memory constraints
  + Volatility of the virtual machine environment
  + Required turnabout time
* Personnel attributes
  + Analyst capability
  + Software engineering capability
  + Applications experience
  + Virtual machine experience
  + Programming language experience
* Project attributes
  + Use of software tools
  + Application of software engineering methods
  + Required development schedule

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

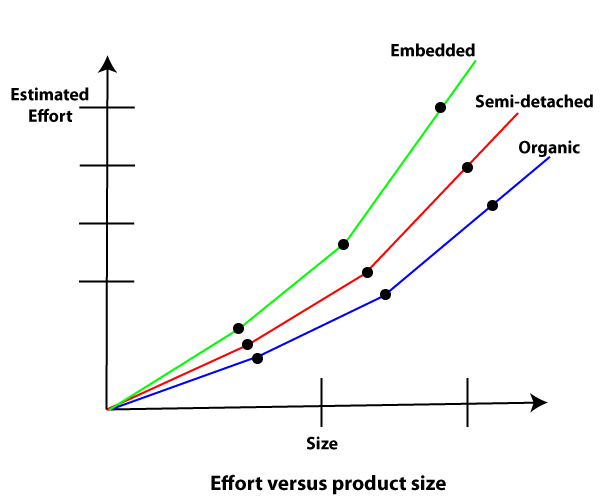
EAF is the factor calculated above.

The coefficient ai and the exponent bi are given in the next table.

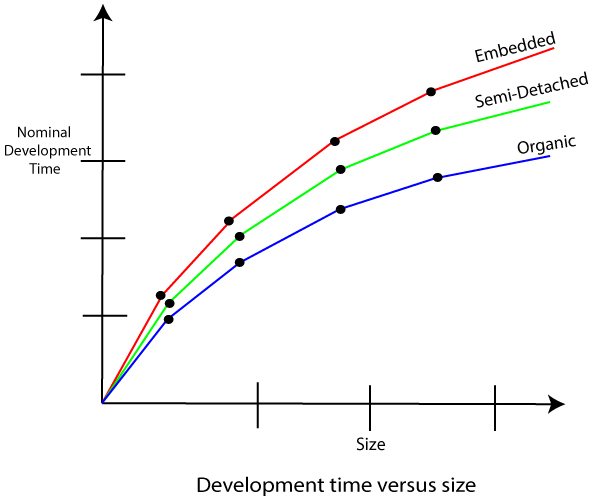
|  |  |  |
| --- | --- | --- |
| Software project | ai | bi |
| Organic | 3.2 | 1.05 |
| Semi-detached | 3.0 | 1.12 |
| Embedded | 2.8 | 1.20 |

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

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.

****

The development time versus the product size in KLOC is plotted in fig. From fig it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.

****

From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

**COCOMO- II**

**CO**nstructive **CO**st **MO**del II (COCOMO II) is a model that allows one to estimate the cost, effort, and schedule when planning a new software development activity. COCOMO II is the latest major extension to the original COCOMO (COCOMO 81) model published in 1981. It consists of three sub models, each one offering increased fidelity the further along one is in the project planning and design process. Listed in increasing fidelity, these sub models are called the Applications Composition, Early Design, and Post-architecture models.

The primary objectives of the COCOMO II effort are:

* To develop a software cost and schedule estimation model tuned to the life cycle practices of the 1990’s and 2000’s.
* To develop software cost database and tool support capabilities for continuous model improvement.
* To provide a quantitative analytic framework, and set of tools and techniques for evaluating the effects of software technology improvements on software life cycle costs and schedules.

**COCOMO - II can be used for the following major decision situations**

* Making investment or other financial decisions involving a software development effort
* Setting project budgets and schedules as a basis for planning and control
* Deciding on or negotiating tradeoffs among software cost, schedule, functionality, performance or quality factors
* Making software cost and schedule risk management decisions
* Deciding which parts of a software system to develop, reuse, lease, or purchase
* Making legacy software inventory decisions: what parts to modify, phase out, outsource, etc
* Setting mixed investment strategies to improve organization's software capability, via reuse, tools, process maturity, outsourcing, etc
* Deciding how to implement a process improvement strategy, such as that provided in the SEI- CMM

The four main elements of the COCOMO II strategy are:

* Preserve the openness of the original COCOMO;
* Key the structure of COCOMO II to the future software marketplace sectors described above;
* Key the inputs and outputs of the COCOMO II submodels to the level of information available;
* Enable the COCOMO II submodels to be tailored to a project’s particular process strategy

COCOMO II provides the following three-stage series of models for estimation of Application Generator,

System Integration, and Infrastructure software projects:

1. The earliest phases or spiral cycles will generally involve prototyping, using the Application Composition model capabilities. The COCOMO II Application Composition model supports these phases, and any other prototyping activities occurring later in the life cycle.

2. The next phases or spiral cycles will generally involve exploration of architectural alternatives or incremental development strategies. To support these activities, COCOMO II provides an early estimation model called the Early Design model. This level of detail in this model is consistent with the general level of information available and the general level of estimation accuracy needed at this stage.

3. Once the project is ready to develop and sustain a fielded system, it should have a life-cycle architecture, which provides more accurate information on cost driver inputs, and enables more accurate cost estimates. To support this stage, COCOMO II provides the Post-Architecture model.

The **Application Composition model** involves prototyping efforts to resolve potential high-risk issues such as user interfaces, software/system interaction, performance, or technology maturity. The costs of this type of effort are best estimated by the Applications Composition model.

The **Early Design model** involves exploration of alternative software/system architectures and concepts of operation. At this stage, not enough is generally known to support fine-grain cost estimation.

The **Post-Architecture model** involves the actual development and maintenance of a software product. This stage proceeds most cost-effectively if a software life-cycle architecture has been developed; validated with respect to the system’s mission, concept of operation, and risk; and established as the framework for the product.

[**COCOMO I Model**](https://www.geeksforgeeks.org/software-engineering-cocomo-model/)**:**

The Constructive Cost Model was first developed by Barry W. Boehm. The model is for estimating effort, cost, and schedule for software projects. It is also called as Basic COCOMO. This model is used to give an approximate estimate of the various parameters of the project. Example of projects based on this model is business system, payroll management system and inventory management systems.

[**COCOMO II Model**](https://www.geeksforgeeks.org/software-engineering-cocomo-ii-model/)**:**

The COCOMO-II is the revised version of the original Cocomo (Constructive Cost Model) and is developed at the University of Southern California. This model calculates the development time and effort taken as the total of the estimates of all the individual subsystems. In this model, whole software is divided into different modules. Example of projects based on this model is Spreadsheets and report generator.

**Difference between COCOMO I and COCOMO II:**

| COCOMO I | COCOMO II |
| --- | --- |
| COCOMO I is useful in the waterfall models of the software development cycle. | COCOMO II is useful in non-sequential, rapid development and reuse models of software. |
| It provides estimates pf effort and schedule. | It provides estimates that represent one standard deviation around the most likely estimate. |
| This model is based upon the linear reuse formula. | This model is based upon the non linear reuse formula |
| This model is also based upon the assumption of reasonably stable requirements. | This model is also based upon reuse model which looks at effort needed to understand and estimate. |
| Effort equation’s exponent is determined by 3 development modes. | Effort equation’s exponent is determined by 5 scale factors. |
| Development begins with the requirements assigned to the software. | It follows a spiral type of development. |
| Number of submodels in COCOMO I is 3 and 15 cost drivers are assigned | In COCOMO II, Number of submodel are 4 and 17 cost drivers are assigned |
| Size of software stated in terms of Lines of code | Size of software stated in terms of Object points, function points and lines of code |

**Let’s have some Example :-**

**Example** :-Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

**Solution:** The basic COCOMO equation takes the form:

  Effort=a1\*(KLOC) a2 PM  
                Tdev=b1\*(efforts)b2 Months  
                Estimated Size of project= 400 KLOC

**(i)Organic Mode**

                E = 2.4 \* (400)1.05 = 1295.31 PM  
                D = 2.5 \* (1295.31)0.38=38.07 PM

**(ii)Semidetached Mode**

                E = 3.0 \* (400)1.12=2462.79 PM  
                D = 2.5 \* (2462.79)0.35=38.45 PM

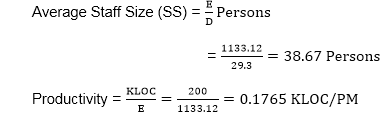
**(iii) Embedded Mode**

                E = 3.6 \* (400)1.20 = 4772.81 PM  
                D = 2.5 \* (4772.8)0.32 = 38 PM

**Another Example :** A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

**Solution:** The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence       E=3.0(200)1.12=1133.12PM  
                D=2.5(1133.12)0.35=29.3PM



            P = 176 LOC/PM

* **Example for COCOMO :-**

**Example 1]** A Project Estimated for 251 KLOC has to be developed ,for development project also requires:-

1. Software Reliability is High (1.15)
2. Product Complexity is High (1.15)
3. Analyst capability is high (0.86)
4. Programming Language Experience is low (1.07)
5. Remaining all drivers are treated ,as nominal

Calculate Effort, Development Time period ,Staff Size & Productivity

**Answer 1:-** a) Project Size is =251 KLOC

b) Cost Driver given as

Software Reliability is High = 1.15

Product Complexity is High =1.15

Analyst capability is high =0.86

Programming Language Experience is low = 1.07

Remaining all drivers are treated, as nominal =1.0

c) EAF = Effort Adjustment factor = Multiplication of all cost drivers

EAF = 1.15 X 1.15 X 0.86 X 1.07 X 1.0

**EAF = 1.2169**

d) Effort = E

As Size is 251 KLOC , we use “Intermediate Model & Semidetached Model “

**E= ai (size)bi \* EAF**

E= 3.0 (251) 1.12 \* 1.2169

⸫ Effort **E = 1778.32** Person Month or Man Months

e)Development Time Period ( D) :-

**D= ci ( E) di**

D= 2.5 (1778.32) 0.35

⸫Development Time Period **D= 34.31** Months

f) Staff Size (SS) :-

**SS = E/D**

**( Staff Size = Effort / Development Time Period )**

SS = 1778.32 / 34.81

SS = 51.83 ≈ 52

**SS = 52 Persons**

g) Productivity P :-

P = Size / Effort

P= Size /E

P= 251 / 1778.32

P= 0.1411

⸫ **Productivity P= 0.1411 KLOC/ PM**

**Example 2]** A Project Estimated for 350 KLOC has to be developed, for development project also requires:-

1. Software Reliability is High (1.15)
2. Product Complexity is High (1.15)
3. Analyst capability is high (0.86)
4. Programming Language Experience is low (1.07)
5. Remaining all drivers are treated ,as nominal

Calculate Effort, Development Time period ,Staff Size & Productivity

**Answer 2]** :- a) Project Size is = 350 KLOC

b) Cost Driver given as

Software Reliability is High = 1.15

Product Complexity is High =1.15

Analyst capability is high =0.86

Programming Language Experience is low = 1.07

Remaining all drivers are treated ,as nominal =1.0

c) EAF = Effort Adjustment factor = Multiplication of all cost drivers

EAF = 1.15 X 1.15 X 0.86 X 1.07 X 1.0

**EAF = 1.2169**

Let’s Try to solve it in by **Organic Mode** The Size = 350 KLOC

d) Effort = E= ai (size)bi \* EAF

E= 3.2 (350)1.05 X 1.2169

E= 3.2 X 469.106 X 1.2169

**E= 1826.73 Person Month**

e) Development Time Period ( D) :-

**D= ci ( E) di**

D= 2.5 (1826.73)0.38

D= 2.5 X (17.355)

**D= 43.38 months**

f) Staff Size (SS) :-

**SS = E/D**

**( Staff Size = Effort / Development Time Period )**

SS= 1826.73 / 43.48

**SS = 42.10 ≈ 42 Person**

g) Productivity P :-

P = Size / Effort

P= Size /E

P= 350 / 1826.73

**P= 0.1915** KLOC/ PM

Let’s Try to solve it(same Problem ) in by **Embedded Mode** & The Size = 350 KLOC

We already calculate Effort Adjustment Factor **(EAF) = 1.2169**

Effort = **E= ai (size)bi \* EAF**

E = 2.8(350)1.20 X 1.2169

E= 2.8 X 1129.488 X 1.2169

**E= 3848.52 Person Month**

Development Time Period ( D) :-

**D= ci ( E) di**

D= 2.5 (3848.52)0.32

D= 2.5 X 14.0376

**D= 35.09 Months**

Staff Size (SS) :-

**SS = E/D**

**( Staff Size = Effort / Development Time Period )**

SS= 3848.52 / 35.09

**SS= 109.67 ≈ 110 Person**

Productivity P :-

P = Size / Effort

P= Size /E

P= 350 / 3848.52

**P= 0.090944 KLOC/ PM**

**Example 3]** You as Project Manager are required to give efforts estimation for the project of size 340 KLOC for all classes of project .For development project also requires

1. Execution time constraints is high :- 1.11
2. Database size is high :- 1.08
3. Programmer capability is very High :- 0.70
4. Modern Programming practices very low :- 1.10
5. Remaining factors drivers are Nominal

**Answer 3] :-** a) Project Size is =340 KLOC

b) Cost Driver given as

Execution time constraints is high = 1.11

Database size is high = 1.08

Programmer capability is very High = 0.70

Modern Programming practices very low = 1.10

Remaining all drivers are treated ,as nominal =1.0

c) EAF = Effort Adjustment factor = Multiplication of all cost drivers

EAF = 1.11 X 1.08 X 0.70 X 1.10 X 1.0 = 0.923076

**EAF = 0.923076**

d) Effort = E

As Size is 340 KLOC , we use “Intermediate Model & Semidetached Model “

**E= ai (size)bi \* EAF**

E= 3.0 (340) 1.12 \* 0.923076

E= 3.0 X 684.315 X 0.923076

⸫ Effort **E = 1895.02** Person Month or Man Months

e)Development Time Period ( D) :-

**D= ci ( E) di**

D= 2.5 (1895.02) 0.35

D= 2.5 X 14.033 =35.083

⸫Development Time Period **D= 35.083** Months

f) Staff Size (SS) :-

**SS = E/D**

**( Staff Size = Effort / Development Time Period )**

SS = 1895.02 / 35.083

SS = 54.01 ≈ 54

**SS = 54 Persons**

g) Productivity P :-

P = Size / Effort

P= Size /E

P= 340 / 1895.02

P= 0.1794

⸫ **Productivity P= 0.1794 KLOC/ PM**

**Example 4]** Consider a project with 400 KLOC and the cost drivers are given below

Low Reliability = 0.88

High Product Complexity = 1.15

Low Application experience = 1.13

High Programming Language Experience = 0.95

Remaining factors drivers are Nominal

Calculate Effort, Development Time period ,Staff Size & Productivity

**Answer 4] :-** a) Project Size is =400 KLOC

Cost Driver given as :-

Low Reliability = 0.88

High Product Complexity = 1.15

Low Application experience = 1.13

High Programming Language Experience = 0.95

Remaining factors drivers are Nominal =1.0

c) EAF = Effort Adjustment factor = Multiplication of all cost drivers

EAF = 0.88 X 1.15 X 1.13 X 0.95 X 1.0 = 0.94468

**EAF = 0.94468**

d) Effort = E

As Size is 400 KLOC , we use “Intermediate Model & Semidetached Model “

**E= ai (size)bi \* EAF**

E= 3.0 (400) 1.12 \* 0.94468

E= 3.0 X 820.932 X 0.94468

⸫ Effort **E = 2326.554** Person Month or Man Months

e)Development Time Period ( D) :-

**D= ci ( E) di**

D= 2.5 (2326.554) 0.35

D= 2.5 X 15.0782 =37.6955

⸫Development Time Period **D= 37.6955** Months

f) Staff Size (SS) :-

**SS = E/D**

**( Staff Size = Effort / Development Time Period )**

SS = 2326.554 / 37.6955

SS = 61.71 ≈ 62

**SS = 62 Persons**

g) Productivity P :-

P = Size / Effort

P= Size /E

P= 400 / 2326.554

P= 0.17192

⸫ Productivity **P= 0.17192 KLOC/ PM**

* **2.1.2] Delphi Cost Estimation:-**

This technique is based on soft skills and relies more on human factors, such as collecting information during group discussions.

The Delphi technique is a Human-based estimation technique. Human-based estimation techniques use human experience and analytical skills to estimate the size, productivity, and effort required for a project. This is a trusted technique and is widely used in many established organizations to facilitate practical and reasonable estimation. The rationale of using the Delphi technique is that when many experts independently arrive at the same estimate on the basis of similar assumptions, the estimate is likely to be correct.

The Delphi technique has eight basic steps:

1. Identify the terms that need to perform the estimation activity. In an estimation

activity meeting, three distinct groups of people need to be present.

### Estimation experts: They usually consist of groups of five or six experienced project managers. The estimation values provided by the project managers are based on past project history and their knowledge. However, only those project managers should be invited for estimation whose experience of a past project matches that of the current project. Otherwise, estimation values may turn out to be far from realistic.

* **Estimation coordinator**: An estimation coordinator is very similar to a moderator in a usual meeting. The coordinator facilitates the meeting and ensures that the goals of the meeting are fully achieved.
* **Author**: An author is similar to a recorder of minutes in a meeting.

2. The author presents the project details including clients’ needs and system requirements to the group of experts. The author also describes the expectations from the group. The author and experts jointly identify the tasks that need to be estimated. They also identify the valid assumptions that they need to consider while estimating. For example, while estimating the effort needed to create a high-level design, they can assume that the SRS document is approved by the client.

3. The author and experts arrive at a consensus that any estimation with a specific variance value will not be accepted. For example, they may decide that any variance above 25 percent will not be accepted as an estimation value for computing the project effort or the productivity.

4. The coordinator prepares a list of tasks jointly decided by the team and distributes the list to all experts. These tasks comprise a project plan.

5. The experts independently make their estimates for each task. After recording their estimates, they hand over their estimates to the coordinator. This is a critical step. While making estimates, no discussions or consultations are permitted because a mutual discussion may influence the estimation logic of the fellow experts. The coordinator and the author jointly ensure this.

6. The coordinator prepares a summary of estimates for each task in a table .After calculating the percentage of variance, the coordinator marks each task as accepted or not accepted based on the agreed accepted value.

7. The coordinator hands over the summary to the group of experts and the author. The group of experts and the author discuss tasks and assumptions where the percentage of variance is more than the acceptable level. The maximum and minimum estimates of tasks are not disclosed or discussed.

8. Revert to step 5 and repeat the steps. You do this until all tasks are assigned estimates that have an acceptable percentage of variance value. Figure summarizes the steps of the Delphi technique in the form of a flowchart.

**Identify the teams that will estimate.**

**Present project details to the expert group.**

**Finalize the acceptable variance value.**

**Prepare a list of tasks.**

**Estimate done by the expert group**

**Prepare a summary of estimates for each task.**

**Discuss tasks and assumptions for not acceptable estimates.**

**Repeat steps until all estimates are finalized**

**Figure : Steps of the Delphi Technique**

The Delphi technique is a simple and subjective method of estimation. However, it is a very effective method because most of the estimates are tried and tested. You can use this method if the project is small or if you have the data and expertise that can enable unambiguous estimates.

* **2.2] Function Point Analysis (Problem Statement ) :-**

A function point is a unit of measurement to express the amount of business functionality an information system (as a product) provides to a user. Function points measure software size. The cost (in dollars or hours) of a single unit is calculated from past projects.

Function points were defined in 1979 by Allan Albrecht at IBM. The functional user requirements of the software are identified and each one is categorized into one of five types: outputs, inquiries, inputs, internal files, and external interfaces. Once the function is identified and categorized into a type, it is then assessed for complexity and assigned a number of function points. Each of these functional user requirements maps to an end-user business function, such as a data entry for an Input or a user query for an Inquiry.

Function points are a standard unit of measure that represents the functional size of a software application. In the same way that a house is measured by the square feet it provides, the size of an application can be measured by the number of function points it delivers to the users of the application.

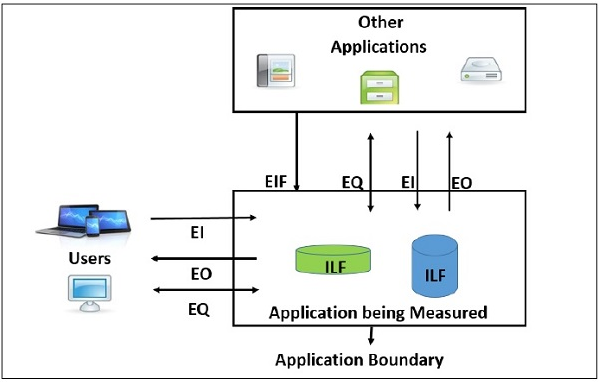
 One of the primary goals of Function Point Analysis is to evaluate a system's capabilities from a user's point of view. To achieve this goal, the analysis is based upon the various ways users interact with computerized systems. From a user's perspective a system assists them in doing their job by providing five (5) basic functions. Two of these address the data requirements of an end user and are referred to as Data Functions. The remaining three addresses the user's need to access data and are referred to as Transactional Functions.

Elementary Process (EP)

Elementary Process is the smallest unit of functional user requirement that −

* Is meaningful to the user.
* Constitutes a complete transaction.
* Is self-contained and leaves the business of the application being counted in a consistent state.

**The Five Components of Function Points**



http://www.qpmg.com/images/arrowblt.gif **Data Functions**

* Internal Logical Files
* External Interface Files
* **Transactional Functions**
* External Inputs
* External Outputs
* External Inquiries

**Internal Logical Files** –

Internal Logical File (ILF) is a user identifiable group of logically related data or control information that resides entirely within the application boundary. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted. An ILF has the inherent meaning that it is internally maintained, it has some logical structure and it is stored in a file.

The first data function allows users to utilize data they are responsible for maintaining. For example, a pilot may enter navigational data through a display in the cockpit prior to departure. The data is stored in a file for use and can be modified during the mission. Therefore the pilot is responsible for maintaining the file that contains the navigational information. Logical groupings of data in a system, maintained by an end user, are referred to as Internal Logical Files (ILF).   
  
**External Interface Files** –

External Interface File (EIF) is a user identifiable group of logically related data or control information that is used by the application for reference purposes only. The data resides entirely outside the application boundary and is maintained in an ILF by another application. An EIF has the inherent meaning that it is externally maintained, an interface has to be developed to get the data from the file.

The second Data Function a system provides an end user is also related to logical groupings of data. In this case the user is not responsible for maintaining the data. The data resides in another system and is maintained by another user or system. The user of the system being counted requires this data for reference purposes only. For example, it may be necessary for a pilot to reference position data from a satellite or ground-based facility during flight. The pilot does not have the responsibility for updating data at these sites but must reference it during the flight. Groupings of data from another system that are used only for reference purposes are defined as External Interface Files (EIF).

The remaining functions address the user's capability to access the data contained in ILFs and EIFs. This capability includes maintaining, inquiring and outputting of data. These are referred to as Transactional Functions.   
  
**External Input**–

External Input (EI) is a transaction function in which Data goes “into” the application from outside the boundary to inside. This data is coming external to the application.

* Data may come from a data input screen or another application.
* An EI is how an application gets information.
* Data can be either control information or business information.
* Data may be used to maintain one or more Internal Logical Files.
* If the data is control information, it does not have to update an Internal Logical File.

The first Transactional Function allows a user to maintain Internal Logical Files (ILFs) through the ability to add, change and delete the data. For example, a pilot can add, change and delete navigational information prior to and during the mission. In this case the pilot is utilizing a transaction referred to as an External Input (EI). An External Input gives the user the capability to maintain the data in ILF's through adding, changing and deleting its contents.   
  
**External Output** –

External Output (EO) is a transaction function in which data comes “out” of the system. Additionally, an EO may update an ILF. The data creates reports or output files sent to other applications

The next Transactional Function gives the user the ability to produce outputs. For example a pilot has the ability to separately display ground speed, true air speed and calibrated air speed. The results displayed are derived using data that is maintained and data that is referenced. In function point terminology the resulting display is called an External Output (EO).   
  
**External Inquiries** –

External Inquiry (EQ) is a transaction function with both input and output components that result in data retrieval.

The final capability provided to users through a computerized system addresses the requirement to select and display specific data from files. To accomplish this a user inputs selection information that is used to retrieve data that meets the specific criteria. In this situation there is no manipulation of the data. It is a direct retrieval of information contained on the files. For example if a pilot displays terrain clearance data that was previously set, the resulting output is the direct retrieval of stored information. These transactions are referred to as External Inquiries (EQ).

FP Counting Process involves the following steps −

* **Step 1** − Determine the type of count.
* **Step 2** − Determine the boundary of the count.
* **Step 3** − Identify each Elementary Process (EP) required by the user.
* **Step 4** − Determine the unique EPs.
* **Step 5** − Measure data functions.
* **Step 6** − Measure transactional functions.
* **Step 7** − Calculate functional size (unadjusted function point count).
* **Step 8** − Determine Value Adjustment Factor (VAF).
* **Step 9** − Calculate adjusted function point count.

⸫Function Point (FP)= Unadjusted Function Points (UFP) X

Complexity or Value Adjustment Factor (CAF/VAF)

**Unadjusted function points (UFP):-**

The FPA measures functionality that the user requires. The specific user functionality is a measurement of the functionality delivered by the application as for user request. The 5 function types identified are

* + external input which receives information from outside the application boundary,
  + external output which presents information of the information system,
  + external enquiries which is special kind of an external output.  
    An external inquiry presents information of the information system based on a uniquely identifying search criterion, without applying additional processing (such as calculations).
  + internal logical files contains permanent data that is relevant to the user The information system references and maintains the data and
  + external interface files also contains permanent data that is relevant to the user. The information system references the data, but the data is maintained by another information system

For each function identified above the function is further classified as

 Low, Average, High  and a weight is given to each. The sum of the weights quantifies the size of information processing and is referred to as the Unadjusted Function points.

| **FUNCTION UNITS** | **LOW** | **AVG** | **HIGH** |
| --- | --- | --- | --- |
| EI | 3 | 4 | 6 |
| EO | 4 | 5 | 7 |
| EQ | 3 | 4 | 6 |
| ILF | 7 | 10 | 15 |
| EIF | 5 | 7 | 10 |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Component | Complexity of Components | | | |
|  | Low | Average | High | Total |
| External Inputs (EI) | \_\_\_\_ X 3= EI1 | \_\_\_\_\_ X 4 =EI2 | \_\_ X 6= EI3 |  |
| External Outputs (EO) | ----- X 4= EO1 | ----- X 5= EO2 | ----- X 7 = EO3 |  |
| External Inquires (EQ) | ----- X 3= EQ1 | ----- X 4 = EQ2 | ----- X 6 = EQ3 |  |
| Internal Logical files(ILF) | ----- X 7 = ILF1 | ----- X 10 = ILF2 | ----- X 15 = ILF3 |  |
| External Interface file (EIF) | ----- X 5 = EIF1 | ----- X 7 = EIF2 | ----- X 10 = EIF3 |  |
| Total Number of Unadjusted Function Points | | | |  |

**Value or Complexity Adjustment Factor (VAF/CAF**) :- It is based on 14 General System Characteristics (GSC’s) that rate the general functionality of the application being counted. Each characteristic has associated descriptions to determine the degrees of influence.

**Rating:**

The degrees of influence range on a scale of zero to five, from no influence to strong influence. Each characteristic is assigned the rating. They ratings are:

0 - Not present, or no influence

1 - Incidental influence

2 - Moderate influence

3 - Average influence

4 - Significant influence

5 - Strong influence throughout

**GSC’s at a Glance:**

|  |  |  |
| --- | --- | --- |
| **General System Characteristic** | | **Brief Description** |
| 1 | Data communications | How many communication facilities are there to aid in the transfer or exchange of information with the application or system? |
| 2 | Distributed data processing(Distributed Function) | How are distributed data and processing functions handled? |
| 3 | Performance Objective | Did the user require response time or throughput? |
| 4 | Heavily used configuration | How heavily used is the current hardware platform where the application will be executed? |
| 5 | Transaction rate | How frequently are transactions executed daily, weekly, monthly, etc.? |
| 6 | On-Line data entry | What percentage of the information is entered On-Line? |
| 7 | End-user efficiency | Was the application designed for end-user efficiency? |
| 8 | On-Line update | How many ILF’s are updated by On-Line transaction? |
| 9 | Complex processing | Does the application have extensive logical or mathematical processing? |
| 10 | Reusability | Was the application developed to meet one or many user’s needs? |
| 11 | Installation ease | How difficult is conversion and installation? |
| 12 | Operational ease | How effective and/or automated are start-up, back up, and recovery procedures? |
| 13 | Multiple sites | Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations? |
| 14 | Facilitate change | Was the application specifically designed, developed, and supported to facilitate change? |

**CAF Or VAF = 0.65 + (Sum of Degrees of Influence of the Fourteen GSC’s ) /100**

**CAF or VAF**  = 0.65 + [( )/100 ]

= 0.65 + 0.01 \*

***Example 1:-*** Consider the project with following functional units

* 1. Number of user input, low or simple = 3 , Average =4 , and high or complex =1
  2. Number of User output , Average = 6 , High or Complex = 2
  3. Number of user Inquires ,Average = 8
  4. Number of user Files, High or complex = 5
  5. Number of external Interface ,Average = 3 , Complex= 4

In addition to above , system requires -

1. Complex Internal processing = 3
2. Code to be reusable = 2
3. High Performance = 4
4. Multiple sites =3
5. Distribute Processing = 5

Other Complexity factors are treated as “Average” , compute Function Point for Project

***Solutions 1:-*** Total number of Unadjusted Function Point (UFP)

UFP = (EI1+EI2+EI3) + (EO1+EO2+EO3) + (EQ1+EQ2+EQ3)+(ILF1+ILF2+ILF3)+(EIF1+EIF2+EIF3)

⸫ UFP =[(3X3)+(4X4)+(1X6)]+ [(0X4)+(6X5)+(2X7) ] + [(0X3)+(8X4)+(0X6)]+[(0X7)+(0X10)+(5X15)]+

[(0X5)+(3X7)+(4X10)]

⸫ UFP= [9+16+6] + [0+30+14] +[0+32+0]+[0+0+75]+[0+21+40]

⸫ UFP = [31] + [44] + [32] + [75] + [61] = 243

⸫ UFP =243

Now Complexity Adjustment Factor (CAF)

CAF= 0.65 + 0.01 \*

The given Complexity Adjustment are as :-

1. Complex Internal processing = 3
2. Code to be reusable = 2
3. High Performance = 4
4. Multiple sites =3
5. Distribute Processing = 5

As per Given problem statement **“five”** CAF Rating value are given and Other Complexity factors are treated as **“Average”** that means remaining **“Nine”** with rating as “**Average**” (⸪ 14 GSC = 5 given + 9 Remaining )

= 3+2+4+3+5 +( 9 X 3) =44

⸫ = 44

⸫ CAF= 0.65 + 0.01 X

CAF = 0.65 + 0.01 X 44

CAF = 0.65+0.44

⸫ CAF = 1.09

Now ,Function Point ( FP ) = UFP X CAF

⸫ FP = 243 X 1.09

⸫ FP = 264.87

***Example 2*** :- Consider the project with following functional units

1. Number of user inputs = 40
2. Number of user outputs =30
3. Number of user enquiries =25
4. Number of user files = 04
5. Number of external Interface =04

In Addition to above systems requires

1. Significant data communication (4)

ii) Performance is very critical (5)

iii) Designed code may be moderately reusable(2)

iv) System is not designed for multiple installation (0)

Other complexity factors are treated as Average .Compute function point for project.

***Solution 2***:-

EI = 40 , EO = 30 , EQ= 25, EIF = 04 , ILF =04

1. UFP = Unadjusted Function point we consider Average Weighting factors

UFP = (40 X 4 ) + (30 X 5) + (25 X 4) + (4 X 10) + (4 X 7)

UFP = 160 + 150 + 100 + 40 +28

UFP = 478

1. Complexity Adjustment factor given as :-

Significant data communication = 4

Performance is very critical = 5

Designed code may be moderately reusable = 2

System is not designed for multiple installation = 0

As per Given problem statement **“four”** CAF Rating value are given and Other Complexity factors are treated as **“Average”** that means remaining **“Ten”** with rating as “**Average**” (⸪ 14 GSC = 4 given + 10 Remaining )

= 4+5+2+0 +( 10 X 3) =11 + 30 = 41

= 41

⸫ CAF= 0.65 + 0.01 X

CAF = 0.65 + 0.01 X 41

CAF = 0.65+0.41

⸫ CAF = 1.06

Now ,Function Point ( FP ) = UFP X CAF

⸫ FP = 478 X 1.06

⸫ FP = 506.68

***Example 3*** :- Consider the project with following functional units

1. Number of user inputs = 11
2. Number of user outputs =11
3. Number of user enquiries =07
4. Number of user files = 22
5. Number of external Interface =06

In Addition to above systems requires

1. Significant data communication (4)

ii) Performance is very critical (5)

iii) Designed code may be moderately reusable(2)

Other complexity factors are treated as Average .Compute function point for project.

***Solution 3***:-

EI = 11 , EO = 11, EQ= 07, EIF = 22 , ILF =06

UFP = Unadjusted Function point we consider Average Weighting factors

UFP = (11 X 4 ) + (11 X 5) + (07 X 4) + (22 X 10) + (06 X 7)

UFP = 44 + 55 + 28 + 220 +42

UFP = 389

Complexity Adjustment factor given as :-

Significant data communication = 4

Performance is very critical = 5

Designed code may be moderately reusable = 2

As per Given problem statement **“Three”** CAF Rating value are given and Other Complexity factors are treated as **“Average”** that means remaining **“Eleven”** with rating as “**Average**” (⸪ 14 GSC = 3 given + 11 Remaining )

= 4+5+2+( 11 X 3) =11 + 33 = 44

= 44

⸫ CAF= 0.65 + 0.01 X

CAF = 0.65 + 0.01 X 44

CAF = 0.65+0.44

⸫ CAF = 1.09

Now ,Function Point ( FP ) = UFP X CAF

⸫ FP = 389 X 1.09

⸫ FP = 424.01

***Example 4*** :- Consider the project with following functional units

1. Number of user inputs = 42
2. Number of user outputs =42
3. Number of user enquiries =57
4. Number of user files = 06
5. Number of external Interface =06

In Addition to above systems requires

1. Application specifically designed, developed, and supported to be installed at multiple sites (05)

ii) Software conversion and installation (05)

iii)Transactions executed frequently (03)

iv) The application designed for end-user efficiency (05)

v) Percentage of the information is entered On-Line (03)

***Solutions 4 :-***

EI = 42 , EO = 42, EQ= 57, EIF = 06 , ILF =06

UFP = Unadjusted Function point we consider Average Weighting factors

UFP = (42 X 4 ) + (42 X 5) + (57 X 4) + (06 X 10) + (06 X 7)

UFP = 168 + 210 + 228 + 60 +42

UFP = 708

Complexity Adjustment factor given as :-

i)Application specifically designed, developed, and supported to be installed at multiple sites

= 05

ii) Software conversion and installation = 05

iii)Transactions executed frequently =03

iv) The application designed for end-user efficiency = 05

v) Percentage of the information is entered On-Line =03

As per Given problem statement **“Five”** CAF Rating value are given and Other Complexity factors are treated as **“Average”** that means remaining **“Nine”** with rating as “**Average**” (⸪ 14 GSC = 5 given + 9 Remaining )

= 5+5+3+5+3+( 9 X 3) =21 + 27 = 48

= 48

⸫ CAF= 0.65 + 0.01 X

CAF = 0.65 + 0.01 X 48

CAF = 0.65+0.48

⸫ CAF = 1.13

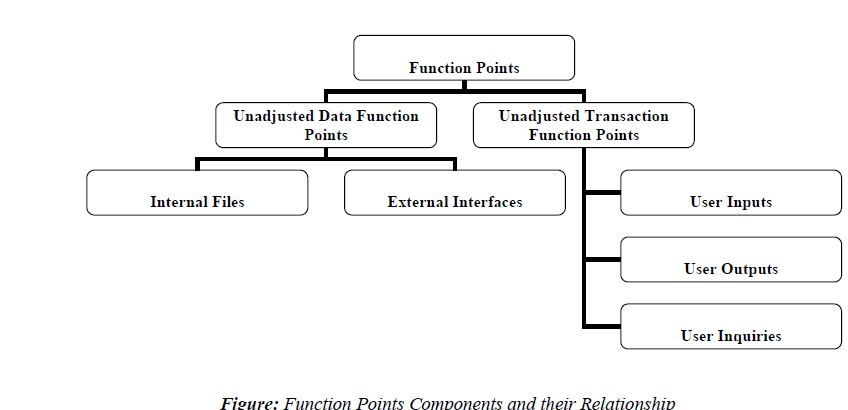
Now ,Function Point ( FP ) = UFP X CAF

⸫ FP = 708 X 1.13

⸫ FP = 800.04

**Features of Function Points**

The total size of a software project is expressed in total function points. It is independent of the computer language, development methodology, technology, or capability of the project team developing the software project. The specific user functionality of the application is evaluated in terms of relation to what is delivered by the application and, not how it is delivered. Only user requested and user-defined components are counted. To calculate FP for a project, some major components are required. The major components and their relationships are represented in following figure.



* **2.3] The SEI & Capability Maturity Model CMM:-**

The **Software Engineering Institute** (**SEI**) is an American research and development center headquartered in [Pittsburgh](https://en.wikipedia.org/wiki/Pittsburgh), [Pennsylvania](https://en.wikipedia.org/wiki/Pennsylvania). Its activities cover cyber security, [software assurance](https://en.wikipedia.org/wiki/Software_assurance), software engineering and acquisition, and component capabilities critical to the Department of Defense

The SEI defines specific initiatives aimed at improving organizations' software engineering capabilities.

SEI in Management Practice Organizations need to effectively manage the acquisition, development, and evolution (ADE) of software-intensive systems. Success in software engineering management practices helps organizations predict and control quality, schedule, cost, cycle time, and productivity. The best-known example of SEI in management practices is the SEI's [Capability Maturity Model (CMM)](https://en.wikipedia.org/wiki/Capability_Maturity_Model) for Software (now [Capability Maturity Model Integration (CMMI)](https://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration)). The CMMI approach consists of models, appraisal methods, and training courses that have been proven to improve process performance

SEI work in engineering practices increases the ability of software engineers to analyze, predict, and control selected functional and non-functional properties of software systems.

The SEI is also the home of the [CERT/CC](https://en.wikipedia.org/wiki/CERT/CC) (CERT Coordination Center), a federally funded computer security organization. The SEI CERT Program's primary goals are to ensure that appropriate technology and systems-management practices are used to resist attacks on networked systems and to limit damage and ensure continuity of critical services in spite of successful attacks, accidents, or failures. The SEI CERT program is working with [US-CERT](https://en.wikipedia.org/wiki/US-CERT) to produce the Build Security In (BSI) website, which provides guidelines for building security into every phase of the [software development lifecycle](https://en.wikipedia.org/wiki/Software_development_lifecycle). The SEI has also conducted research on insider threats and [computer forensics](https://en.wikipedia.org/wiki/Computer_forensics)

* **The Capability Maturity Model CMM** classifies SE processes in to five different levels.

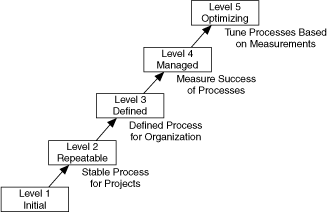
1. Initial Level

2. Repeatable Level

3. Defined Level

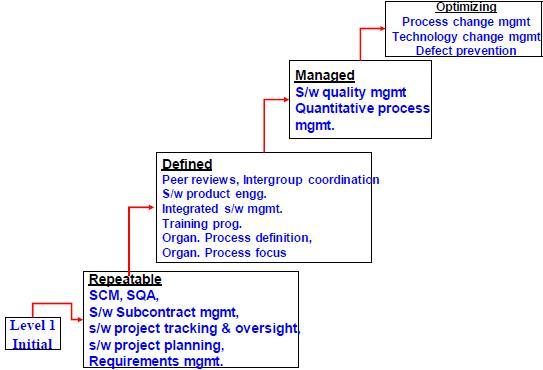
4. Managed Level

5. Optimizing Level



**Figure : Capability Maturity Model**

**SEI CMM: Key process areas of SEI maturity Model (per IEEE):**



**Figure : Capability Maturity Model as per IEEE**

There are five levels defined along the continuum of the model and, according to the SEI: "Predictability, effectiveness, and control of an organization's software processes are believed to improve as the organization moves up these five levels.

The model provides a theoretical continuum along which process maturity can be developed incrementally from one level to the next. Skipping levels is not allowed/feasible.

**Level 1 - Initial (Chaotic)**

It is characteristic of processes at this level that they are (typically) undocumented and in a state of dynamic change, tending to be driven in an *ad hoc*, uncontrolled and reactive manner by users or events. This provides a chaotic or unstable environment for the processes.

**Level 2 - Repeatable**

It is characteristic of processes at this level that some processes are repeatable, possibly with consistent results. Process discipline is unlikely to be rigorous, but where it exists it may help to ensure that existing processes are maintained during times of stress.

**Level 3 - Defined**

It is characteristic of processes at this level that there are sets of defined and documented standard processes established and subject to some degree of improvement over time. These standard processes are in place (i.e., they are the AS-IS processes) and used to establish consistency of process performance across the organization.

**Level 4 - Managed**

It is characteristic of processes at this level that, using process metrics, management can effectively control the AS-IS process (e.g., for software development). In particular, management can identify ways to adjust and adapt the process to particular projects without measurable losses of quality or deviations from specifications. Process Capability is established from this level.

**Level 5 - Optimizing**

It is a characteristic of processes at this level that the focus is on continually improving process performance through both incremental and innovative technological changes/improvements.

At maturity level 5, processes are concerned with addressing statistical *common causes* of process variation and changing the process (for example, to shift the mean of the process performance) to improve process performance. This would be done at the same time as maintaining the likelihood of achieving the established quantitative process-improvement objectives.

* **2.4 Software configuration Management :-**

Configuration management (CM) is a [systems engineering](https://en.wikipedia.org/wiki/Systems_engineering) process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life. The CM process is widely used by military engineering organizations to manage changes throughout the [system lifecycle](https://en.wikipedia.org/wiki/System_lifecycle) of [complex systems](https://en.wikipedia.org/wiki/Complex_system),

CM applied over the life cycle of a system provides visibility and control of its performance, functional, and physical attributes. CM verifies that a system performs as intended, and is identified and documented in sufficient detail to support its projected life cycle. The CM process facilitates orderly management of system information and system changes for such beneficial purposes as to revise capability; improve performance, reliability, or maintainability; extend life; reduce cost; reduce risk and liability; or correct defects. The relatively minimal cost of implementing CM is returned many fold in cost avoidance. The lack of CM, or its ineffectual implementation, can be very expensive and sometimes can have such catastrophic consequences such as failure of equipment or loss of life.

CM emphasizes the functional relation between parts, subsystems, and systems for effectively controlling system change. It helps to verify that proposed changes are systematically considered to minimize adverse effects. Changes to the system are proposed, evaluated, and implemented using a standardized, systematic approach that ensures consistency, and proposed changes are evaluated in terms of their anticipated impact on the entire system. CM verifies that changes are carried out as prescribed and that documentation of items and systems reflects their true configuration. A complete CM program includes provisions for the storing, tracking, and updating of all system information on a component, subsystem, and system basis.

A structured CM program ensures that documentation (e.g., requirements, design, test, and acceptance documentation) for items is accurate and consistent with the actual physical design of the item. In many cases, without CM, the documentation exists but is not consistent with the item itself. For this reason, engineers, contractors, and management are frequently forced to develop documentation reflecting the actual status of the item before they can proceed with a change. This [reverse engineering](https://en.wikipedia.org/wiki/Reverse_engineering) process is wasteful in terms of human and other resources and can be minimized or eliminated using CM.

Software configuration management, SCM is an activity which is used at every level and every part of the process of software Engineering. Every improvement takes the shape of better control. This is a discipline which controls betters and according to client need in software Engineering. With the help of this many types are changes which play an important role in software Engineering and development process.

In the simple way if we define the term configuration of management, this is the tool which makes better control, easy maintenance during the whole process of software development. With the help of software configuration management we can easily find out what modification and controlling required by the developer. SCM have the capacity to control all those effects which comes in software projects. The main objectives of SCM is increase the production by reduce the errors.

When a software development process start then SCM take change by identification, control, alteration, audit and etc. after that the output of total process provided to our customer. We can clarify the action of SCM as:

1. Software configuration identification - Normally software is used in various kinds of programs and documentation and data related to each program is called configuration identification. With the help of C.I we can make a guide line which will be helpful in software development process, several time the requirement of guideline for check the document and design of software. Document related to SCM are the useful item, with the help of this we can make better control and take a basic unit for configuration.
2. Software configuration control - This is the process of deciding with the help of this we make coordination between the changes which is necessary and apply them as per mentioned in guideline. Configuration control board gives the permission for any kind of change or modification which is necessary for the project. Many times CCB take advice of those members which are the part of software development process.
3. Accounting status of Software configuration - The process of maintaining record of all data which is necessary for the software is called accounting status of software. It has all the data related to the old software to new software that what changes are done or required for the fulfilment of the customer need.
4. Auditing of software configuration - Auditing of software configuration is may be defined as an art with the help of this we can understand that the required actions or changes are done by the developer or not. Some of the item involved in the process of verifying or auditing.
   * Function is properly performed by the software.
   * The process of documentation, data is completed or not.

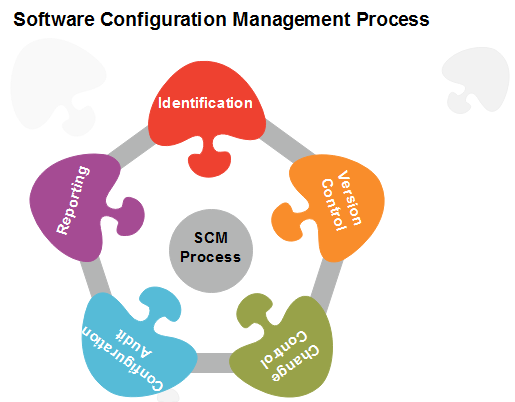
**Benefits**

* + With the help of SCM we can easily control all changes which are done in development process.
  + It gives the surety to check that changes are done on required area.
  + It is helpful to generate the new software with old components.
  + SCM has the capacity to explain everything about the process of software development.

**SCM Process**

It uses the tools which keep that the necessary change has been implemented adequately to the appropriate component. The SCM process defines a number of tasks:

* Identification of objects in the software configuration
* Version Control
* Change Control
* Configuration Audit
* Status Reporting



ChangeManagementis the process by which changes to the Project’s scope,

deliverables, timescales or resources are formally defined, evaluated and

approved prior to implementation. This is achieved by understanding the business and system

drivers requiring the change, documenting the benefits and costs of adopting the change and formulating a structured plan for implementing the change.

The change management process in systems engineering is the process of requesting, determining attainability, planning, implementing, and evaluating of changes to a system. It has two main goals: supporting the processing of changes and enabling traceability of changes, which should be possible through proper execution of the process described.

* Change management is not a stand-alone process for designing a business solution.
* Change management is the processes, tools and techniques for managing the people-side of change.
* Change management is not a process improvement method.
* Change management is a method for reducing and managing resistance to change when implementing process, technology or organizational change.
* Change management is a necessary component for any organizational performance improvement process to succeed, including programs like: Six Sigma, Business Process Reengineering, Total Quality Management, Organizational Development, Restructuring and continuous process improvement.
* Change management is how we drive the adoption and usage we need to realize business results.

|  |  |
| --- | --- |
| **Table 1: Role descriptions for the change management process** | |
| **Role** | **Description** |
| **Customer** | The [customer](http://en.wikipedia.org/wiki/Customer) is the role that requests a change due to problems encountered or new functionality requirements; this can be a person or an organizational entity and can be in- or external to the company that is asked to implement the change. |
| **Project manager** | The [project manager](http://en.wikipedia.org/wiki/Project_manager) is the owner of the [project](http://en.wikipedia.org/wiki/Project) that the CHANGE REQUEST concerns. In some cases there is a distinct change manager, who in that case takes on this role. |
| **Change committee** | The change [committee](http://en.wikipedia.org/wiki/Committee) decides whether a CHANGE REQUEST will be implemented or not. Sometimes this task is performed by the project manager as well. |
| **Change builder** | The change builder is the person who plans and implements the change; it could be argued that the planning component is (partially) taken on by the project manager. |
|  |  |

|  |  |  |
| --- | --- | --- |
| **Table 2: Activity descriptions for the change management process** | | |
| **Activity** | **Sub-activity** | **Description** |
| **Identify potential change** | Require new functionality | A customer desires new functionality and formulates a REQUIREMENT. |
|  | Encounter problem | A customer encounters a problem (e.g. a [bug](http://en.wikipedia.org/wiki/Software_bug)) in the system and this leads to a PROBLEM REPORT. |
|  | Request change | A customer proposes a change through creation of a CHANGE REQUEST. |
| **Analyze change request** | Determine technical feasibility | The project manager determines the technical feasibility of the proposed CHANGE REQUEST, leading to a CHANGE TECHNICAL FEASIBILITY. |
|  | Determine costs and benefits | The project manager determines the costs and benefits of the proposed CHANGE REQUEST, resulting in CHANGE COSTS AND BENEFITS. This and the above sub-activity can be done in any order and they are independent of each other, hence the modeling as unordered activities. |
| **Evaluate change** |  | Based on the CHANGE REQUEST, its CHANGE TECHNICAL FEASIBILITY and CHANGE COSTS AND BENEFITS, the change committee makes the go/no-go decision. This is modeled as a separate activity because it is an important process step and has another role performing it. It is modeled as a sub-activity (without any activity containing it) as recommended by Remko Helms (personal communication). |
| **Plan change** | Analyze change impact | The extent of the change (i.e. what other items the change effects) is determined in a CHANGE IMPACT ANALYSIS. It could be argued that this activity leads to another go/no-go decision, or that it even forms a part of the Analyze change request activity. It is modeled here as a planning task for the change builder because of its relationship with the activity Propagate change. |
|  | Create planning | A CHANGE PLANNING is created for the [implementation](http://en.wikipedia.org/wiki/Implementation) of the change. Some process descriptions (e.g. Mäkäräinen, 2000) illustrate that is also possible to ‘save’ changes and process them later in a [batch](http://en.wikipedia.org/wiki/Batch_production). This activity could be viewed as a good point to do this. |
| **Implement change** | Execute change | The change is ‘programmed’; this activity has a strong relationship with Propagate change, because sometimes the change has to be adapted to other parts of the system (or even other systems) as well. |
|  | Propagate change | The changes resulting from Execute change have to be propagated to other system parts that are influenced by it. Because this and the above sub-activity are highly dependent on each other, they have been modeled as concurrent activities. |
|  | Test change | The change builder tests whether what (s)he has built actually works and satisfies the CHANGE REQUEST. As depicted in the diagram, this can result in an [iterative](http://en.wikipedia.org/wiki/Iterative) process together with the above two sub-activities. |
|  | Update documentation | The DOCUMENTATION is updated to reflect the applied changes. |
|  | Release change | A new SYSTEM RELEASE, which reflects the applied change, is made public. |
| **Review and close change** | Verify change | The implementation of the change in the new SYSTEM RELEASE is verified for the last time, now by the project manager. Maybe this has to happen before the release, but due to conflicting literature sources and diagram complexity considerations it was chosen to model it this way and include this issue. |
|  | Close change | This change [cycle](http://en.wikipedia.org/wiki/Instruction_cycle) is completed, i.e. the CHANGE LOG ENTRY is wrapped up. |

### Versioning and Version control :-

* Versioning is the creation and management of multiple releases of a product, all of which have the same general function but are improved, upgraded or customized. Version control is the practice of ensuring collaborative data sharing and editing among users of systems that employ different versions of a product. The terms "versioning" and "version control" are sometimes used interchangeably even though their technical meanings are different.
* In software versioning, subsequent releases of a specific product receive numerical identifiers consisting of two or three numbers separated by periods. The first number, called the major number, is increased when there are significant improvements or changes in functionality. The second number, called the minor number, is incremented when there are minor feature changes or significant fixes. The third number, if it exists, is called the revision number. It is added or increased when minor bugs are eliminated.
* Version control combines procedures and tools to manage different versions of configuration objects that are created during software product development.
* To control versions, you can use Version Control Register. In Version Control Register, you enter the details of components, such as component identification numbers, their versions, and dates of validity. It is advisable to release a baseline after a version is released. Baseline is a specification or a product that is formally reviewed and agreed upon. This serves as the basis for further development. Baseline can be changed only through formal change control procedures. A baseline consists of a set of SCIs that are logically related to each other. Baselines are established when subsequent changes to the SCIs need to be controlled. Version control is essential so that everybody uses only the latest version. Any kind of version mismatch might result in rework.
* SCM differentiates between baselines and interim versions. A baseline is a tested and certified version of a system. Baselines can be assigned version numbers such as 1.0, 2.0, 3.0, and so on. A baseline usually undergoes intensive testing. Interim versions, on the other hand, have version numbers, such as 1.1 or 1.2. The interim version is a temporary version. Interim versions have a short life and survive only during bug fixing, testing, or debugging.
* Version control is used to manage multiple versions of computer files and programs. A version control system, or VCS (Version Control System ), provides two primary data management capabilities. It allows users to 1) lock files so they can only be edited by one person at a time, and 2) track changes to files.
* If you are the only person editing a document, there is no need to lock a file for editing. However, if a team of developers is working on a project, it is important that no two people are editing the same file at the same time. When this happens, it is possible for one person to

accidentally overwrite the changes made by someone else. For this reason, version control allows users to "check out" files for editing. When a file has been checked out from a shared file server, it cannot be edited by other users. When the person finishes editing the file, he can save the changes and "check in" the file so that other users can edit the file.

Version control also allows users to track changes to files. This type of version control is often used in software development and is also known as "source control" or "revision control." Popular version control systems like Subversion and CVS allow developers to save incremental versions of programs and source code files during the development process. This provides the capability to rollback to an earlier version of the program if necessary. For example, if bugs are found in a new version of a software program, the developer can review the previous version when debugging the code.

Version control software requires that all files are saved in a central location. This location is called the repository and contains all previous and current versions of files managed by the VCS. Whenever a new file is created or a current file is updated, the changes are "committed" to the repository, so the latest version is available to all users.

A good VCS (Version Control System) does the following:

* **Backup and Restore**. Files are saved as they are edited, and you can jump to any moment in time.
* **Synchronization.** Lets people share files and stay up-to-date with the latest version.
* **Track Changes**. As files are updated, you can leave messages explaining why the change happened (stored in the VCS, not the file). This makes it easy to see how a file is evolving over time, and why.
* **Track Ownership.** A VCS tags every change with the name of the person who made it. Helpful for blame storming giving credit.

**Release Management Process:-**

Release Management is the process responsible for planning, scheduling, and controlling the build, in addition to testing and deploying Releases. It aims to plan, schedule and control the movement of releases to test and live environments.

The primary goal of Release Management and Deployment Management is to ensure that the integrity of the live environment is protected and that the correct components are released.

Release Management encompasses the planning, design, build, configuration and testing of hardware and software releases to create a defined set of release components.

Release management is a software engineering process intended to oversee the development, testing, deployment and support of software releases.

Release management usually begins in the development cycle with requests for changes or new features. If the request is approved, the new release is planned and designed. The new design enters the testing or quality assurance phase, in which the release is built, reviewed, tested and tweaked until it is ultimately accepted as a release candidate. The release then enters the deployment phase, where it is implemented and made available. Once deployed, the release enters a support phase, where bug reports and other issues are collected; this leads to new requests for changes, and the cycle starts all over again.

Release management software allows release teams to plan, manage and control the release schedule and track the status of each release to ensure production worthiness.

**Objective:-**

• Increase the number of successful Releases, including reducing the number of Releases with unexpected outcomes.

• Decrease the number of incidents caused by Releases.

• Create a single documented process for managing all Releases.

• Maintain a single repository for recording all Releases through the lifecycle.

• Ensure that the process is adopted, adhered to, and escalated to management if there are compliance issues

• Improve productivity by establishing standard release processes

* **Extra Reading :-**

**Configuration Management Tools:-**

Configuration management (CM) is a [systems engineering](http://en.wikipedia.org/wiki/Systems_engineering) process for establishing and maintaining consistency of a product's performance, functional and physical attributes with its requirements, design and operational information throughout its life.

**Software Configuration Management (SCM)** is the task of tracking and controlling changes in the software, part of the larger cross-discipline field of configuration management. SCM practices include revision control and the establishment of baselines. If something goes wrong, SCM can determine what was changed and who changed it.

Configuration management (CM) is the detailed recording and updating of information that describes an enterprise's hardware and software. Such information typically includes the versions and updates that have been applied to installed software packages and the locations and network addresses of hardware devices.

Configuration Management is a set of interrelated processes, management techniques, and supporting tools that assure:

1. Our configurations are as they should be, meeting necessary requirements and matching the latest documentation.

2. Changes to our configurations are properly evaluated, authorized, and implemented.   
3. All information necessary to define and manage our configurations & related documentation is :

  (a) kept current and accurate,

(b) structured for users’ needs, and

(c) readily available to all who need to know.

A configuration management system helps development teams control the process of modifying source code, developing documentation, and managing products.

Features or characteristics of configuration management tools are:

* To store information about versions and builds of the software and testware.
* Traceability between software and testware and different versions or variants.
* To keep track of which versions belong with which configurations (e.g. operating systems, libraries, browsers).
* To build and release management.
* Baselining (e.g. all the configuration items that make up a specific release).
* Access control (checking in and out).

A configuration management tool must be able to bring in existing projects from flat and spread-out directory structures and from deeply nested directory structures. When project data is migrated, the tool must provide an easy way to create a baseline. A baseline is project version upon which all subsequent development effort is based.

 Ideally, a configuration management tool is not apparent, allowing developers to work on source code files, insulated from disruptions caused by changes from other developers. At a certain point, however, that insulation must relax so the developers can incorporate other changes from other developers into their own projects. Therefore, a configuration management system must provide developers with insulated areas in which to work, yet also provide an efficient way in which they can work as a team, sharing modifications to their source code.

An important goal of a configuration management tool is to allow several types of projects to take place simultaneously. The tool must provide a way for developers to reuse the code, no matter what directory it was created in.

**Rayleigh Curve :-** In the study of software project scheduling, the Putnam–Norden–Rayleigh curve, also known as the PNR curve, is an equation specifying the relationship between applied effort and delivery time for a software project. A PNR curve can be used to determine the least cost time for delivery  t_o  up to the limit t_{min}, the absolute minimal amount of time required to complete the project no matter how many human resources are added.

**Staffing and the Rayleigh Curve**

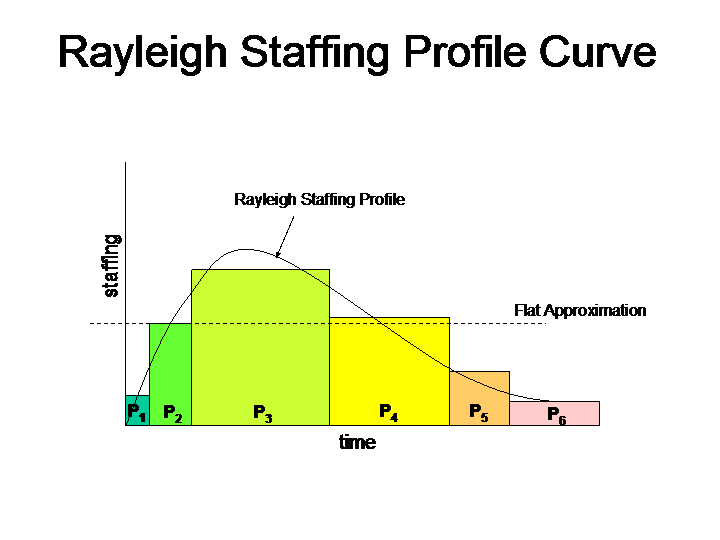
It is very natural for a project manager to want to:

* figure out how many people will be needed to complete a project.
* assign them to it.
* divide the number of estimated staff months by the head-count to get an expected completion time.

Sadly, this isn't how it works:

1. Peter Norden (of IBM, in 1963) observed that a project is not a single monolithic activity, to be accomplished by a single team. Rather, it is a sequence of distinct but overlapping phases, each of which has is own natural team size and composition.
2. Frederick Brooks (in the Mythical Man Month) pointed out that time and man-power are not interchangeable because team size and composition affect productivity.

Every project actually has a *natural* staffing curve, and these curves seem to have a consistent and predictable shape.



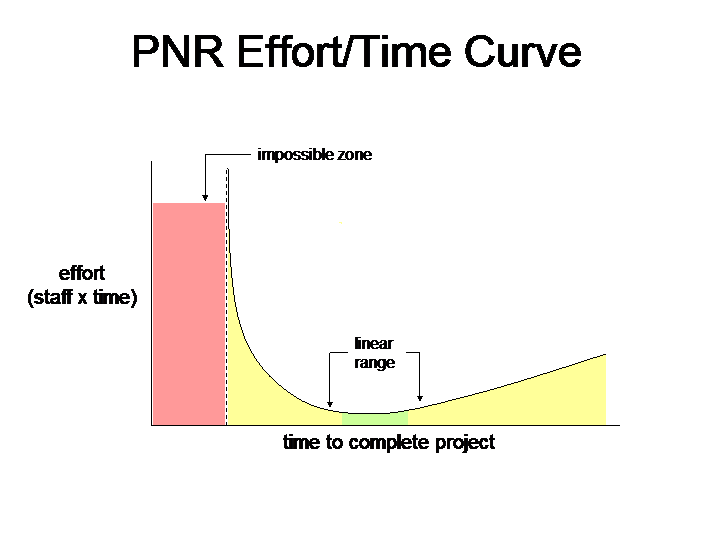
We can attempt to force a flat staffing profile on such a project, but it will result in considerable waste:

* In the early stages, there will be people standing around with nothing to do, because the requirements and architecture development process is more gated by time than by labor.
* In the mid stages, work may be delayed by not having enough people available to perform all of the defined tasks.
* In the late stages, over-staffing will not bring the project to completion any sooner, because the collection of feedback and shaking out of final problems is (again) more gated by time than available labor.

If we want to accomplish a project efficiently, we much understand the distinct phases, along with their respective skill and activity requirements. The notion that we can prepare an estimate in staff months and predict completion time by choosing a staffing level and dividing that into the total project size is a canard.

**3. Time/Effort and the PNR Curve**

Lawrence Putnam (of RADC) applied Norden and Rayleigh's work to the effort to quantitatively predict the work associated with software projects. After studying numerous projects, he concluded that the effort required to deliver a project varied inversely as the fourth power of the time allotted to its completion.



The clear message here is that every project has an optimal staffing level, and (correspondingly) an optimal time in which it will be completed. Going significantly above or below the optimal staffing level will reduce work efficiency ... and there may be a point beyond which adding people actually delays the project.

**This curve breaks down into four zones:**

1. **An impossible zone**

The project cannot be accomplished in less time than this, no matter how many people are applied to the problem. This situation is classically summarized as: "Nine women cannot have a baby in one month".

1. **An "Haste makes waste" zone**

Adding people does accelerate delivery, but not in proportion to the added effort. Each additional person added to the project lowers our productivity (they have to be trained, more time goes into communication and coordination, more misunderstandings). This is a very inefficient way to operate.

1. **A linear range**

This is the range of efficient staffing, and within this range it is possible to trade man-power for time, or vice versa.

1. **An under-staffed/over-staffed zone**

This curve does not yield completion time as a function of staffing, but merely shows the relationship between staffing level and completion times. What we can clearly see on the right of the curve is that productivity is dropping. Why might this be?

* + If the project is critically under-staffed, productivity will suffer because there aren't enough people to deal with the problems.
  + if the project is greatly over-staffed communications overhead will reduce efficiency and misunderstandings will create problems and result in wasted work.

**Norden’s Work :-**

Norden studied the staffing patterns of several R & D projects. He found that the staffing pattern can be approximated by the Rayleigh distribution curve (as shown in fig. 11.6). Norden represented the Rayleigh curve by the following equation:

**E = K/t²d \* t \* e-t² / 2 t²d**

Where E is the effort required at time t. E is an indication of the number of engineers (or the staffing level) at any particular time during the duration of the project, K is the area under the curve, and td is the time at which the curve attains its maximum value. It must be remembered that the results of Norden are applicable to general R & D projects and were not meant to model the staffing pattern of software development projects.