Software Process and Project Metrics:

Software metric is a measure of software characteristics which are measurable or countable. Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.

Within the software development process, many metrics are that are all connected. Software metrics are similar to the four functions of management: Planning, Organization, Control, or Improvement.

Classification of Software Metrics

Software metrics can be classified into two types as follows:

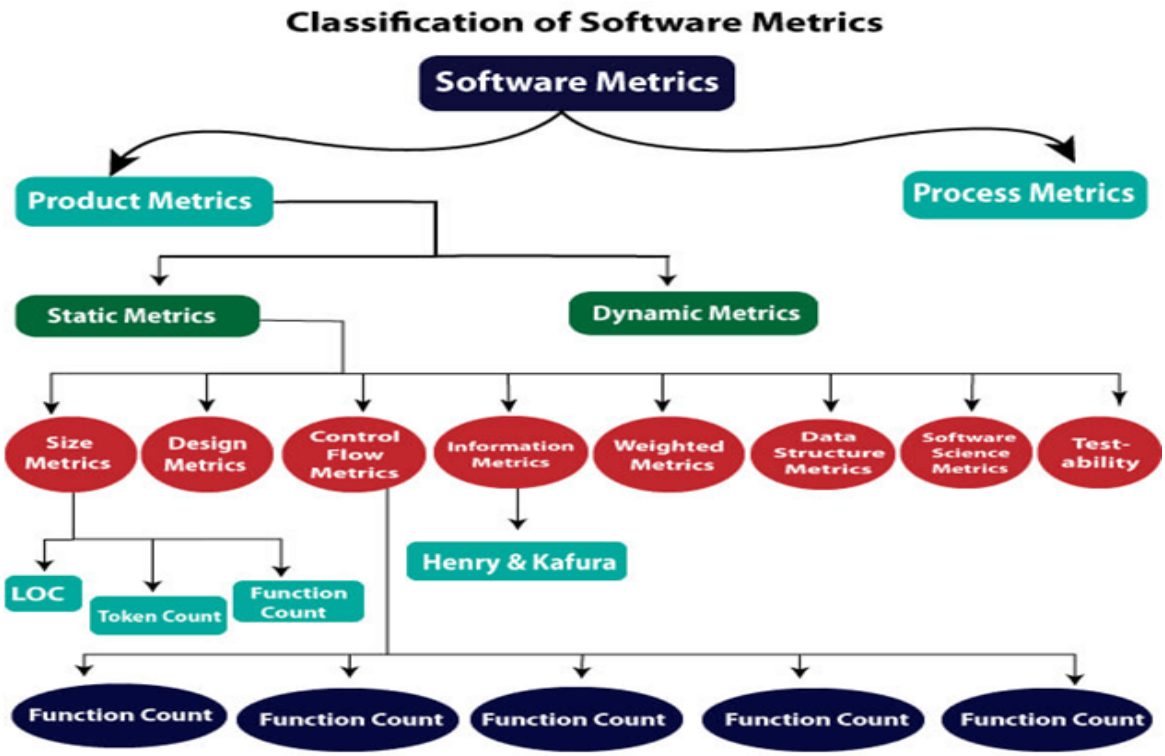
1. **Product Metrics:** These are the measures of various characteristics of the software product. The two important software characteristics are:

a. Size and complexity of software.

b. Quality and reliability of software.

These metrics can be computed for different stages of SDLC.

2. **Process Metrics:** These are the measures of various characteristics of the software development process. For example, the efficiency of fault detection. They are used to measure the characteristics of methods, techniques, and tools that are used for developing software.



Types of Metrics

Internal metrics: Internal metrics are the metrics used for measuring properties that are viewed to be of greater importance to a software developer. For example, Lines of Code (LOC) measure.

External metrics: External metrics are the metrics used for measuring properties that are viewed to be of greater importance to the user, e.g., portability, reliability, functionality, usability, etc.

Hybrid metrics: Hybrid metrics are the metrics that combine product, process, and resource metrics. For example, cost per FP where FP stands for Function Point Metric.

Project metrics: Project metrics are the metrics used by the project manager to check the project's progress. Data from the past projects are used to collect various metrics, like time and cost; these estimates are used as a base of new software. Note that as the project proceeds, the project manager will check its progress from time-to-time and will compare the effort, cost, and time with the original effort, cost and time. Also understand that these metrics are used to decrease the development costs, time efforts and risks. The project quality can also be improved. As quality improves, the number of errors and time, as well as cost required, is also reduced.

Advantage of Software Metrics

1. Comparative study of various design methodology of software systems. For analysis, comparison, and critical study of different programming language concerning their characteristics.

2. In comparing and evaluating the capabilities and productivity of people involved in software development.

3. In the preparation of software quality specifications.

4. In the verification of compliance of software systems requirements and specifications.

5. In making inference about the effort to be put in the design and development of the software systems.

6. In getting an idea about the complexity of the code.

7. In taking decisions regarding further division of a complex module is to be done or not.

8. In guiding resource manager for their proper utilization.

9. In comparison and making design trade-offs between software development and maintenance cost.

10. In providing feedback to software managers about the progress and quality during various phases of the software development life cycle.

11. In the allocation of testing resources for testing the code.

Disadvantage of Software Metrics

1. The application of software metrics is not always easy, and in some cases, it is difficult and costly.

2. The verification and justification of software metrics are based on historical/empirical data whose validity is difficult to verify.

3. These are useful for managing software products but not for evaluating the performance of the technical staff.

4. The definition and derivation of Software metrics are usually based on assuming which are not standardized and may depend upon tools available and working environment.

5. Most of the predictive models rely on estimates of certain variables which are often not known precisely.

Size Oriented Metrics

LOC Metrics

It is one of the earliest and simpler metrics for calculating the size of the computer program. It is generally used in calculating and comparing the productivity of programmers. These metrics are derived by normalizing the quality and productivity measures by considering the size of the product as a metric.

Following are the points regarding LOC measures:

1) In size-oriented metrics, LOC is considered to be the normalization value.
2) It is an older method that was developed when FORTRAN and COBOL programming were very popular.
3) Productivity is defined as KLOC / EFFORT, where effort is measured in person-months.
4) Size-oriented metrics depend on the programming language used.
5) As productivity depends on KLOC, so assembly language code will have more productivity.
6) LOC measure requires a level of detail which may not be practically achievable.
7) The more expressive is the programming language, the lower is the productivity.
8) LOC method of measurement does not apply to projects that deal with visual (GUI-based) programming. As already explained, Graphical User Interfaces (GUIs) use forms basically. LOC metric is not applicable here.
9) It requires that all organizations must use the same method for counting LOC. This is so because some organizations use only executable statements, some useful comments, and some do not. Thus, the standard needs to be established.

10) These metrics are not universally accepted.

Based on the LOC/KLOC count of software, many other metrics can be computed:

1. Errors/KLOC.
2. $/ KLOC.
3. Defects/KLOC.
4. Pages of documentation/KLOC.
5. Errors/PM.
6. Productivity = KLOC/PM (effort is measured in person-months).
7. $/ Page of documentation.

Advantages of LOC

Simple to measure

Disadvantage of LOC

1. It is defined on the code. For example, it cannot measure the size of the specification.
2. It characterizes only one specific view of size, namely length, it takes no account of functionality or complexity
3. Bad software design may cause an excessive line of code
4. It is language dependent
5. Users cannot easily understand

Functional Point (FP) Analysis

Allan J. Albrecht initially developed function Point Analysis in 1979 at IBM and it has been further modified by the International Function Point Users Group (IFPUG). FPA is used to make estimate of the software project, including its testing in terms of functionality or function size of the software product. However, functional point analysis may be used for the test estimation of the product. The functional size of the product is measured in terms of the function point, which is a standard of measurement to measure the software application.

Objectives of FPA

The basic and primary purpose of the functional point analysis is to measure and provide the software application functional size to the client, customer, and the stakeholder on their request. Further, it is used to measure the software project development along with its maintenance, consistently throughout the project irrespective of the tools and the technologies.

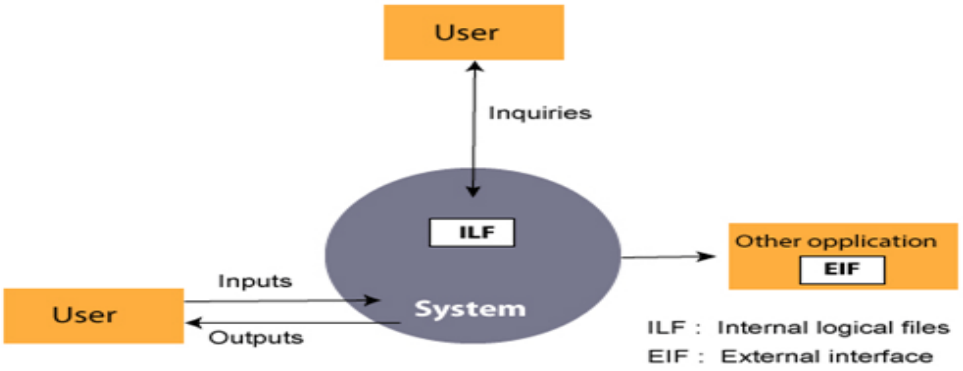
Following are the points regarding FPs

1. FPs of an application is found out by counting the number and types of functions used in the applications. Various functions used in an application can be put under five types, as shown in Table:

Types of FP Attributes

Measurements Parameters	Examples
1.Number of External Inputs(EI)	Input screen and tables
2. Number of External Output (EO)	Output screens and reports
3. Number of external inquiries (EQ)	Prompts and interrupts.
4. Number of internal files (ILF)	Databases and directories
5. Number of external interfaces (EIF)	Shared databases and shared routines.

All these parameters are then individually assessed for complexity.



FPAs Functional Units System

2. FP characterizes the complexity of the software system and hence can be used to depict the project time and the manpower requirement.

3. The effort required to develop the project depends on what the software does.

4. FP is programming language independent.

5. FP method is used for data processing systems, business systems like information systems.

6. The five parameters mentioned above are also known as information domain characteristics.

7. All the parameters mentioned above are assigned some weights that have been experimentally determined and are shown in Table:

Weights of 5-FP Attributes

Measurement Parameter	Low	Average	High
1. Number of external inputs (EI)	7	10	15
2. Number of external outputs (EO)	5	7	10
3. Number of external inquiries (EQ)	3	4	6
4. Number of internal files (ILF)	4	5	7
5. Number of external interfaces (EIF)	3	4	6

The functional complexities are multiplied with the corresponding weights against each function, and the values are added up to determine the UFP (Unadjusted Function Point) of the subsystem.

Computing FPs

Measurement Parameter	Count		Weighing factor			
			Simple	Average	Complex	
1. Number of external inputs (EI)	—	*	3	4	6 =	—
2. Number of external Output (EO)	—	*	4	5	7 =	—
3. Number of external Inquiries (EQ)	—	*	3	4	6 =	—
4. Number of internal Files (ILF)	—	*	7	10	15 =	—
5. Number of external interfaces(EIF)	—	*	5	7	10 =	—
Count-total →						

Here that weighing factor will be simple, average, or complex for a measurement parameter type.

The Function Point (FP) is thus calculated with the following formula.

$FP = \text{Count-total} * [0.65 + 0.01 * \sum(fi)]$

$= \text{Count-total} * CAF$

Where Count-total is obtained from the above Table.

$CAF = [0.65 + 0.01 * \sum(fi)]$

and ∑(fi) is the sum of all 14 questionnaires and show the complexity adjustment value/ factor-CAF (where i ranges from 1 to 14). Usually, a student is provided with the value of ∑(fi)

Also note that ∑(fi) ranges from 0 to 70, i.e.,

$$0 <= \sum(fi) <= 70$$

and CAF ranges from 0.65 to 1.35 because

When ∑(fi) = 0 then CAF = 0.65

When ∑(fi) = 70 then CAF = 0.65 + (0.01 * 70) = 0.65 + 0.7 = 1.35

Based on the FP measure of software many other metrics can be computed:

- **a.** Errors/FP
- **b.** $/FP.
- **c.** Defects/FP
- **d.** Pages of documentation/FP
- **e.** Errors/PM.
- **f.** Productivity = FP/PM (effort is measured in person-months).
- **g.** $/Page of Documentation.

8. LOCs of an application can be estimated from FPs. That is, they are interconvertible. This process is known as backfiring. For example, 1 FP is equal to about 100 lines of COBOL code.

9. FP metrics is used mostly for measuring the size of Management Information System (MIS) software.

10. But the function points obtained above are unadjusted function points (UFPs). These (UFPs) of a subsystem are further adjusted by considering some more General System Characteristics (GSCs). It is a set of 14 GSCs that need to be considered. The procedure for adjusting UFPs is as follows:

a) Degree of Influence (DI) for each of these 14 GSCs is assessed on a scale of 0 to 5. (b) If a particular GSC has no influence, then its weight is taken as 0 and if it has a strong influence then its weight is 5.
b) The score of all 14 GSCs is totalled to determine Total Degree of Influence (TDI).

c) Then Value Adjustment Factor (VAF) is computed from TDI by using the formula: VAF = (TDI * 0.01) + 0.65

Remember that the value of VAF lies within 0.65 to 1.35 because

a) When TDI = 0, VAF = 0.65
b) When TDI = 70, VAF = 1.35
c) VAF is then multiplied with the UFP to get the final FP count: FP = VAF * UFP

Example1: Compute the function point, productivity, documentation, cost per function for the following data:

Number of user inputs = 24
Number of user outputs = 46
Number of inquiries = 8
Number of files = 4
Number of external interfaces = 2
Effort = 36.9 p-m
Technical documents = 265 pages
User documents = 122 pages
Cost = $7744/ month
Various processing complexity factors are: 4, 1, 0, 3, 3, 5, 4, 4, 3, 3, 2, 2, 4, 5.

Solution:

Measurement Parameter	Count		Weighing factor
1. Number of external inputs (EI)	24	*	4 = 96
2. Number of external outputs (EO)	46	*	4 = 184
3. Number of external inquiries (EQ)	8	*	6 = 48
4. Number of internal files (ILF)	4	*	10 = 40
5. Number of external interfaces (EIF) Count-total →	2	*	5 = 10 378

So sum of all fi (i ← 1 to 14) = 4 + 1 + 0 + 3 + 5 + 4 + 4 + 3 + 3 + 2 + 2 + 4 + 5 = 43

FP = Count-total * [0.65 + 0.01 *∑(fi)]
= 378 * [0.65 + 0.01 * 43]
= 378 * [0.65 + 0.43]

= 378 * 1.08 = 408

$$\text{Productivity} = \frac{\text{FP}}{\text{Effort}} = \frac{408}{36.9} = 11.1$$

Total pages of documentation = technical document + user document
= 265 + 122 = 387pages

Documentation = Pages of documentation/FP
= 387/408 = 0.94

$$\text{Cost per function} = \frac{\text{cost}}{\text{productivity}} = \frac{7744}{11.1} = \$700$$

Differentiate between FP and LOC

FP	**LOC**
1. FP is specification based.	1. LOC is an analogy based.
2. FP is language independent.	2. LOC is language dependent.
3. FP is user-oriented.	3. LOC is design-oriented.
4. It is extendible to LOC.	4. It is convertible to FP (backfiring)

Extended Function Point (EFP) Metrics

FP metric has been further extended to compute:

a. Feature points.
b. 3D function points.

Feature Points

1. Feature point is the superset of function point measure that can be applied to systems and engineering software applications.
2. The feature points are used in those applications in which the algorithmic complexity is high like real-time systems where time constraints are there, embedded systems, etc.
3. Feature points are computed by counting the information domain values and are weighed by only single weight.
4. Feature point includes another measurement parameter-ALGORITHM.
5. The table for the computation of feature point is as follows:

Feature Point Calculations

Measurement Parameter	Count		Weighing factor	
1. Number of external inputs (EI)	-	*	4	-
2. Number of external outputs (EO)	-	*	5	-
3. Number of external inquiries (EQ)	-	*	4	-
4. Number of internal files (ILF)	-	*	7	-
5. Number of external interfaces (EIF)	-	*	7	-
6.Algorithms used Count total →	-	*	3	-

The feature point is thus calculated with the following formula:

$$FP = \text{Count-total} * [0.65 + 0.01 * \sum(fi)]$$

$$= \text{Count-total} * CAF$$

Where count-total is obtained from the above table.

$$CAF = [0.65 + 0.01 * \sum(fi)]$$

and $\sum(fi)$ is the sum of all 14 questionnaires and show the complexity adjustment value/factor-CAF (where i ranges from 1 to 14). Usually, a student is provided with the value of $\sum(fi)$.

6. Function point and feature point both represent systems functionality only.

7. For real-time applications that are very complex, the feature point is between 20 and 35% higher than the count determined using function point above.

3D function points

1. Three dimensions may be used to represent 3D function points, data dimension, functional dimension, and control dimension.
2. The data dimension is evaluated as FPs are calculated. Herein, counts are made for inputs, outputs, inquiries, external interfaces, and files.
3. The functional dimension adds another feature-Transformation, that is, the sequence of steps which transforms input to output.
4. The control dimension that adds another feature-Transition that is defined as the total number of transitions between states. A state represents some externally observable mode.

Computing FPs

Measurement Parameter	Count		Weighing factor		
			Simple	Average	Complex
1. Number of external inputs (EI)	32	*	3	4	6 = 128
2. Number of external Output (EO)	60	*	4	5	7 = 300
3. Number of external Inquiries (EQ)	24	*	3	4	6 = 96
4. Number of internal Files (ILF)	8	*	7	10	15 = 80
5. Number of external interfaces(EIF)	2	*	5	7	10 = 14
Count-total →					618

Now fi for average case = 3. So sum of all fi (i ←1 to 14) = 14 * 3 = 42

$$FP = \text{Count-total} * [0.65 + 0.01 * \sum(fi)]$$
$$= 618 * [0.65 + 0.01 * 42]$$
$$= 618 * [0.65 + 0.42]$$
$$= 618 * 1.07 = 661.26$$

and feature point = (32 *4 + 60 * 5 + 24 * 4 + 80 +14) * 1.07 + {12 * 15 *1.07}
= 853.86

Example: Compute the 3D-function point value for an embedded system with the following characteristics:

Internal data structures = 6

External data structures = 3

No. of user inputs = 12

No. of user outputs = 60

No. of user inquiries = 9

No. of external interfaces = 3

Transformations = 36

Transitions = 24

Assume complexity of the above counts is high.

Solution: We draw the Table first. For embedded systems, the weighting factor is complex and complexity is high. So,

Computing FPs

Measurement Parameter	Count		Weighing factor			
			Simple	Average	Complex	
1. Number of external inputs (EI)	12	*	3	4	6 =	72
2. Number of external Output (EO)	60	*	4	5	7 =	420
3. Number of external Inquiries (EQ)	9	*	3	4	6 =	54
4. Number of internal Data structure (ILF)	6	*	7	10	15 =	90
5. Number of external data structure	3	*	5	7	10 =	30
6. Number of external interfaces	3	*	5	7	10 =	30
7. Number of transformation	36	*			———	36
8. Number of transitions	24	*			———	24
Count-total →						756

So, for 3D function points,the required index is 756.

Software Project Planning

A Software Project is the complete methodology of programming advancement from requirement gathering to testing and support, completed by the execution procedures, in a specified period to achieve intended software product.

Need of Software Project Management

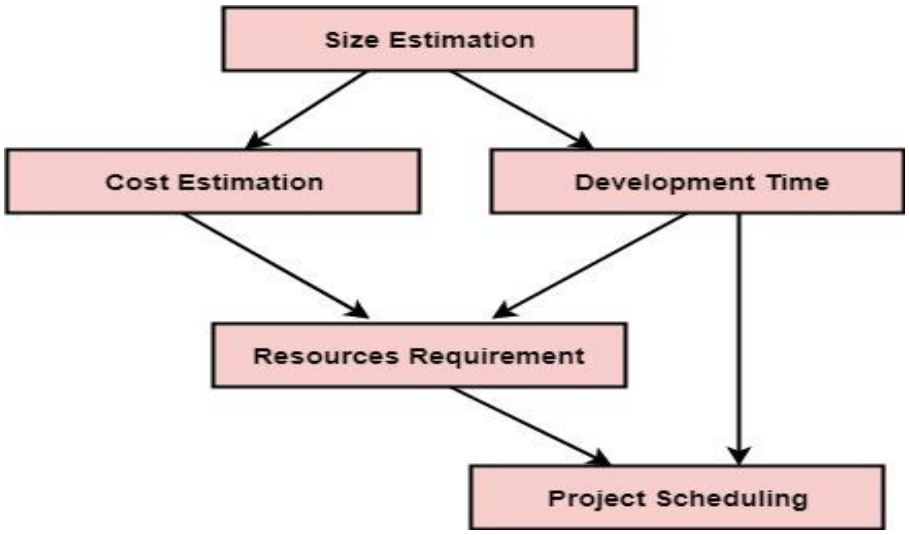
Software development is a sort of all new streams in world business, and there's next to no involvement in structure programming items. Most programming items are customized to accommodate customer's necessities. The most significant is that the underlying technology changes and advances so generally and rapidly that experience of one element may not be connected to the other one. All such business and ecological imperatives bring risk in software development; hence, it is fundamental to manage software projects efficiently.

Software Project Manager

Software manager is responsible for planning and scheduling project development. They manage the work to ensure that it is completed to the required standard. They monitor the progress to check that the event is on time and within budget. The project planning must incorporate the major issues like size & cost estimation scheduling, project monitoring, personnel selection evaluation & risk management. To plan a successful software project, we must understand:

- Scope of work to be completed
- Risk analysis
- The resources mandatory
- The project to be accomplished
- Record of being followed

Software Project planning starts before technical work start. The various steps of planning activities are:



The size is the crucial parameter for the estimation of other activities. Resources requirement are required based on cost and development time. Project schedule may prove to be very useful for controlling and monitoring the progress of the project. This is dependent on resources & development time.

Software Cost Estimation

For any new software project, it is necessary to know how much it will cost to develop and how much development time will it take. These estimates are needed before development is initiated, but how is this done? Several estimation procedures have been developed and are having the following attributes in common.

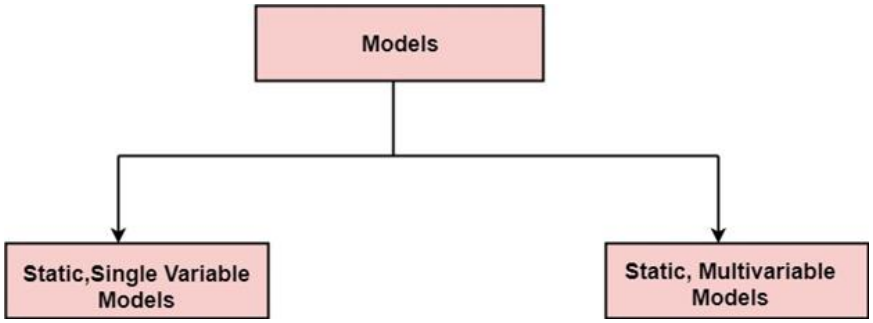
1. Project scope must be established in advanced.
2. Software metrics are used as a support from which evaluation is made.
3. The project is broken into small PCs which are estimated individually. To achieve true cost & schedule estimate, several option arise.
4. Delay estimation
5. Used symbol decomposition techniques to generate project cost and schedule estimates.
6. Acquire one or more automated estimation tools.

Uses of Cost Estimation

1. During the planning stage, one needs to choose how many engineers are required for the project and to develop a schedule.
2. In monitoring the project's progress, one needs to access whether the project is progressing according to the procedure and takes corrective action, if necessary.

Cost Estimation Models

A model may be static or dynamic. In a static model, a single variable is taken as a key element for calculating cost and time. In a dynamic model, all variable are interdependent, and there is no basic variable.



Static, Single Variable Models: When a model makes use of single variables to calculate desired values such as cost, time, efforts, etc. is said to be a single variable model. The most common equation is:

$$C=aL^b$$

Where C = Costs

L= size

a and b are constants

The Software Engineering Laboratory established a model called SEL model, for estimating its software production. This model is an example of the static, single variable model.

$$E=1.4L0.93$$

$$DOC=30.4L0.90$$

$$D=4.6L0.26$$

Where E= Efforts (Person Per Month)

DOC=Documentation (Number of Pages)

D = Duration (D, in months)

L = Number of Lines per code

COCOMO Model

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

The necessary steps in this model are:

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort E_i in person-months the equation used is of the type is shown below

$$\mathbf{E_i=a*(KDLOC)b}$$

The value of the constant a and b are depends on the project type.

In COCOMO, projects are categorized into three types:

1. Organic
2. Semidetached
3. Embedded

1.Organic: A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. **Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.**

2. Semidetached: A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**

3. Embedded: A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. **For Example:** ATM, Air Traffic control.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

According to Boehm, software cost estimation should be done through three stages:

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

1. Basic COCOMO Model: The basic COCOMO model provides an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model

Effort=a_1*(KLOC) a_2 PM
Tdev=b_1*(efforts)b_2 Months

Where

KLOC is the estimated size of the software product indicate in Kilo Lines of Code,

a_1, a_2, b_1, b_2 are constants for each group of software products,

Tdev is the estimated time to develop the software, expressed in months,

Effort is the total effort required to develop the software product, expressed in **person months (PMs)**.

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic: Effort = 2.4(KLOC) 1.05 PM

Semi-detached: Effort = 3.0(KLOC) 1.12 PM

Embedded: Effort = 3.6(KLOC) 1.20 PM

Estimation of development time

For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: Tdev = 2.5(Effort) 0.38 Months

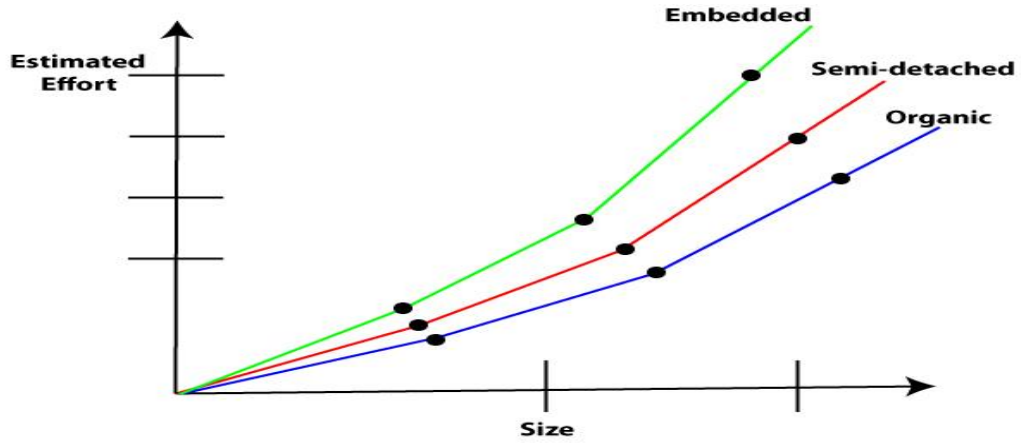
Semi-detached: Tdev = 2.5(Effort) 0.35 Months

Embedded: Tdev = 2.5(Effort) 0.32 Months

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

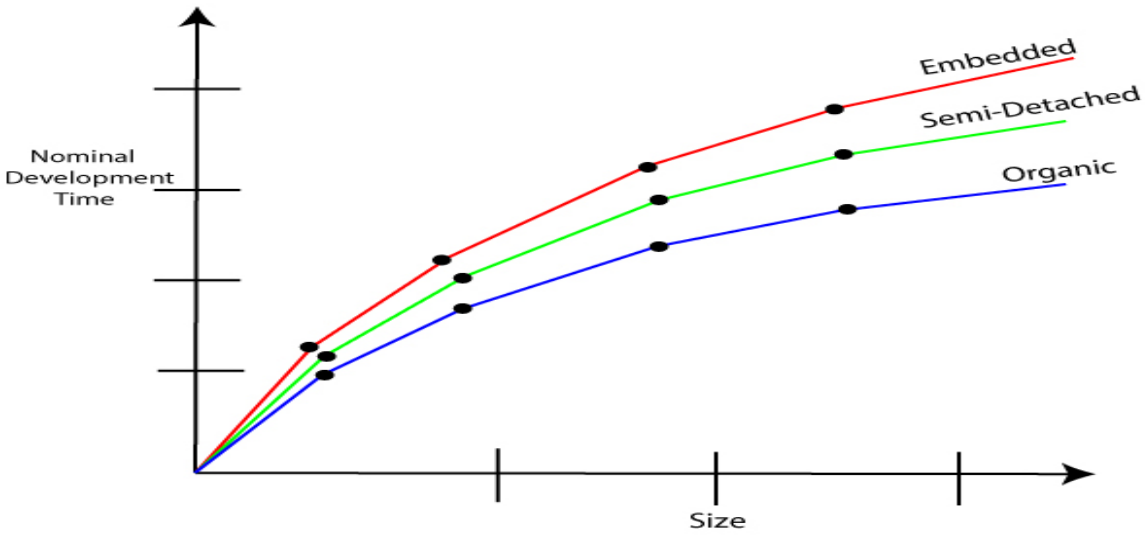
software product. Thus, the effort required to develop a product increases very rapidly with project size.

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.



Effort versus product 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.



Development time versus size

Example1: 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

Example2: 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

$$\text{Average Staff Size (SS)} = \frac{E}{D} \text{ Persons}$$

$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons}$$

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

COCOMO Model

P = 176 LOC/PM

2. Intermediate Model: The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes:

(i) Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The 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

The cost drivers are divided into four categories:

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	..
DATA	..	0.94	1.00	1.08	1.16	..
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME	..	..	1.00	1.11	1.30	1.66
STOR	..	..	1.00	1.06	1.21	1.56
VIRT	..	0.87	1.00	1.15	1.30	..
TURN	..	0.87	1.00	1.07	1.15	..

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	..
AEXP	1.29	1.13	1.00	0.91	0.82	..
PCAP	1.42	1.17	1.00	0.86	0.70	..
VEXP	1.21	1.10	1.00	0.90	..	..
LEXP	1.14	1.07	1.00	0.95	..	..
Project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	..
TOOL	1.24	1.10	1.00	0.91	0.83	..
SCED	1.23	1.08	1.00	1.04	1.10	..

Intermediate COCOMO equation:

$$E=a_i\ (KLOC)\ b_i*EAF$$
$$D=c_i\ (E)d_i$$

Coefficients for intermediate COCOMO

Project	a_i	b_i	c_i	d_i
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

3. Detailed COCOMO Model: Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost driver's effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed cocomo, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System structure
3. Complete structure
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is determined as a function of program estimate, and a set of cost drivers are given according to every phase of the software lifecycle.

What is Risk?

"Tomorrow problems are today's risk." Hence, a clear definition of a "risk" is a problem that could cause some loss or threaten the progress of the project, but which has not happened yet.

These potential issues might harm cost, schedule or technical success of the project and the quality of our software device, or project team morale.

Risk Management is the system of identifying addressing and eliminating these problems before they can damage the project.

We need to differentiate risks, as potential issues, from the current problems of the project.

Different methods are required to address these two kinds of issues.

For example, staff storage, because we have not been able to select people with the right technical skills is a current problem, but the threat of our technical persons being hired away by the competition is a risk.

Risk Management

A software project can be concerned with a large variety of risks. In order to be adept to systematically identify the significant risks which might affect a software project, it is essential to classify risks into different classes. The project manager can then check which risks from each class are relevant to the project.

There are three main classifications of risks which can affect a software project:

1. Project risks
2. Technical risks
3. Business risks

1. Project risks: Project risks concern differ forms of budgetary, schedule, personnel, resource, and customer-related problems. A vital project risk is schedule slippage. Since the software is intangible, it is very tough to monitor and control a software project. It is very tough to control something which cannot be identified. For any manufacturing program, such as the manufacturing of cars, the plan executive can recognize the product taking shape.

2. Technical risks: Technical risks concern potential method, implementation, interfacing, testing, and maintenance issue. It also consists of an ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence. Most technical risks appear due to the development team's insufficient knowledge about the project.

3. Business risks: This type of risks contain risks of building an excellent product that no one need, losing budgetary or personnel commitments, etc.

Other risk categories

1. Known risks: Those risks that can be uncovered after careful assessment of the project program, the business and technical environment in which the plan is being developed, and more reliable data sources (e.g., unrealistic delivery date)

2. Predictable risks: Those risks that are hypothesized from previous project experience (e.g., past turnover)

3. Unpredictable risks: Those risks that can and do occur, but are extremely tough to identify in advance.

Principle of Risk Management

1. **Global Perspective:** In this, we review the bigger system description, design, and implementation. We look at the chance and the impact the risk is going to have.
2. **Take a forward-looking view:** Consider the threat which may appear in the future and create future plans for directing the next events.
3. **Open Communication:** This is to allow the free flow of communications between the client and the team members so that they have certainty about the risks.
4. **Integrated management:** In this method risk management is made an integral part of project management.
5. **Continuous process:** In this phase, the risks are tracked continuously throughout the risk management paradigm.

Risk Management Activities

Risk management consists of three main activities, as shown in fig:

Risk Assessment

The objective of risk assessment is to division the risks in the condition of their loss, causing potential. For risk assessment, first, every risk should be rated in two methods:

- The possibility of a risk coming true (denoted as r).

- The consequence of the issues relates to that risk (denoted as s).

Based on these two methods, the priority of each risk can be estimated:

$$p = r * s$$

Where p is the priority with which the risk must be controlled, r is the probability of the risk becoming true, and s is the severity of loss caused due to the risk becoming true. If all identified risks are set up, then the most likely and damaging risks can be controlled first, and more comprehensive risk abatement methods can be designed for these risks.

1. Risk Identification: The project organizer needs to anticipate the risk in the project as early as possible so that the impact of risk can be reduced by making effective risk management planning.

A project can be of use by a large variety of risk. To identify the significant risk, this might affect a project. It is necessary to categories into the different risk of classes.

There are different types of risks which can affect a software project:

1. **Technology risks:** Risks that assume from the software or hardware technologies that are used to develop the system.
2. **People risks:** Risks that are connected with the person in the development team.
3. **Organizational risks:** Risks that assume from the organizational environment where the software is being developed.
4. **Tools risks:** Risks that assume from the software tools and other support software used to create the system.
5. **Requirement risks:** Risks that assume from the changes to the customer requirement and the process of managing the requirements change.
6. **Estimation risks:** Risks that assume from the management estimates of the resources required to build the system

2. Risk Analysis: During the risk analysis process, you have to consider every identified risk and make a perception of the probability and seriousness of that risk.

There is no simple way to do this. You have to rely on your perception and experience of previous projects and the problems that arise in them.

It is not possible to make an exact, the numerical estimate of the probability and seriousness of each risk. Instead, you should authorize the risk to one of several bands:

1. The probability of the risk might be determined as very low (0-10%), low (10-25%), moderate (25-50%), high (50-75%) or very high (+75%).
2. The effect of the risk might be determined as catastrophic (threaten the survival of the plan), serious (would cause significant delays), tolerable (delays are within allowed contingency), or insignificant.

Risk Control

It is the process of managing risks to achieve desired outcomes. After all, the identified risks of a plan are determined; the project must be made to include the most harmful and the most likely risks. Different risks need different containment methods. In fact, most risks need ingenuity on the part of the project manager in tackling the risk.

There are three main methods to plan for risk management:

1. **Avoid the risk:** This may take several ways such as discussing with the client to change the requirements to decrease the scope of the work, giving incentives to the engineers to avoid the risk of human resources turnover, etc.
2. **Transfer the risk:** This method involves getting the risky element developed by a third party, buying insurance cover, etc.
3. **Risk reduction:** This means planning method to include the loss due to risk. For instance, if there is a risk that some key personnel might leave, new recruitment can be planned.

Risk Leverage: To choose between the various methods of handling risk, the project plan must consider the amount of controlling the risk and the corresponding reduction of risk. For this, the risk leverage of the various risks can be estimated.

Risk leverage is the variation in risk exposure divided by the amount of reducing the risk.

Risk leverage = (risk exposure before reduction - risk exposure after reduction) / (cost of reduction)

1. Risk planning: The risk planning method considers each of the key risks that have been identified and develop ways to maintain these risks.

For each of the risks, you have to think of the behavior that you may take to minimize the disruption to the plan if the issue identified in the risk occurs.

You also should think about data that you might need to collect while monitoring the plan so that issues can be anticipated.

Again, there is no easy process that can be followed for contingency planning. It rely on the judgment and experience of the project manager.

2. Risk Monitoring: Risk monitoring is the method king that your assumption about the product, process, and business risks has not changed.

Software Development Life Cycle (SDLC)

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.

In other words, a life cycle model maps the various activities performed on a software product from its inception to retirement. Different life cycle models may plan the necessary development activities to phases in different ways. Thus, no element which life cycle model is followed, the essential activities are contained in all life cycle models though the action may be carried out in distinct orders in different life cycle models. During any life cycle stage, more than one activity may also be carried out.

Need of SDLC

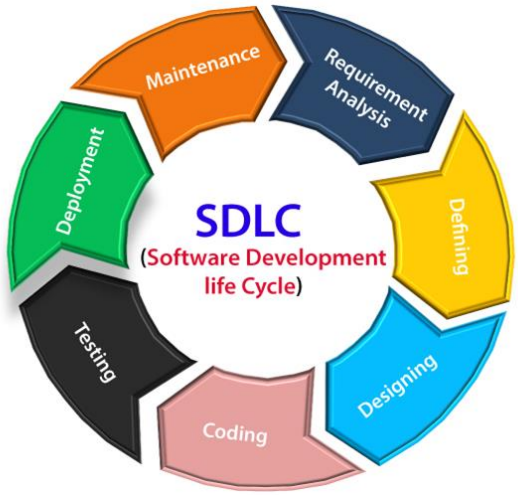
The development team must determine a suitable life cycle model for a particular plan and then observe to it.

Without using an exact life cycle model, the development of a software product would not be in a systematic and disciplined manner. When a team is developing a software product, there must be a clear understanding among team representative about when and what to do. Otherwise, it would point to chaos and project failure. This problem can be defined by using an example. Suppose a software development issue is divided into various parts and the parts are assigned to the team members. From then on, suppose the team representative is allowed the freedom to develop the roles assigned to them in whatever way they like. It is possible that one representative might start writing the code for his part, another might choose to prepare the test documents first, and some other engineer might begin with the design phase of the roles assigned to him. This would be one of the perfect methods for project failure.

A software life cycle model describes entry and exit criteria for each phase. A phase can begin only if its stage-entry criteria have been fulfilled. So without a software life cycle model, the entry and exit criteria for a stage cannot be recognized. Without software life cycle models, it becomes tough for software project managers to monitor the progress of the project.

SDLC Cycle

SDLC Cycle represents the process of developing software. SDLC framework includes the following steps:



The stages of SDLC are as follows:

Stage1: Planning and requirement analysis

Requirement Analysis is the most important and necessary stage in SDLC.

The senior members of the team perform it with inputs from all the stakeholders and domain experts or SMEs in the industry.

Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage.

Business analyst and Project organizer set up a meeting with the client to gather all the data like what the customer wants to build, who will be the end user, what is the objective of the product. Before creating a product, a core understanding or knowledge of the product is very necessary.

For Example, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.

Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.

Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

Stage2: Defining Requirements

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.

This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

Stage3: Designing the Software

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

Stage4: Developing the project

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

Stage5: Testing

After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage. During this stage, unit testing, integration testing, system testing, acceptance testing are done.

Stage6: Deployment

Once the software is certified, and no bugs or errors are stated, then it is deployed. Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.

After the software is deployed, then its maintenance begins.

Stage7: Maintenance

Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time. This procedure where the care is taken for the developed product is known as maintenance.