Software Engineering | Software Project Management (SPM)

Software Project Management (SPM) is a proper way of planning and leading software projects. It is a part of project management in which software projects are planned, implemented, monitored and controlled.

Need of Software Project Management:

Software is an non-physical product. Software development is a new stream in business and there is very little experience in building software products. Most of the software products are made to fit client's requirements. The most important is that the basic technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. Such type of business and environmental constraints increase risk in software development hence it is essential to manage software projects efficiently.

It is necessary for an organization to deliver quality product, keeping the cost within client's budget constrain and deliver the project as per scheduled. Hence in order, software project management is necessary to incorporate user requirements along with budget and time constraints.

Software Project Management consists of several different type of managements:

1. **Conflict Management:**
   Conflict management is the process to restrict the negative features of conflict while increasing the positive features of conflict. The goal of conflict management is to improve learning and group results including efficacy or performance in an organizational setting. Properly managed conflict can enhance group results.

2. **Risk Management:**
   Risk management is the analysis and identification of risks that is followed by synchronized and economical implementation of resources to minimize, operate and control the possibility or effect of unfortunate events or to maximize the realization of opportunities.

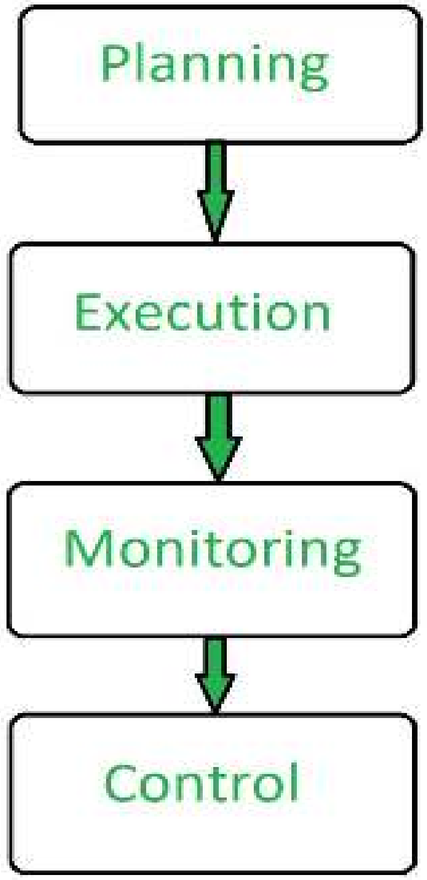
3. **Requirement Management:**
   It is the process of analyzing, prioritizing, tracing and documenting on requirements and then supervising change and communicating to pertinent stakeholders. It is a continuous process during a project.

4. **Change Management:**
   Change management is a systematic approach for dealing with the transition or transformation of an organization's goals, processes or technologies. The purpose of change management is to execute strategies for effecting change, controlling change and helping people to adapt to change.

5. **Software Configuration Management:**
   Software configuration management is the process of controlling and tracing changes in the software, part of the larger cross-disciplinary field of configuration management. Software configuration management include revision control and the inauguration of baselines.

6. **Release Management:**
   Release Management is the task of planning, controlling and scheduling the build in deploying releases. Release management ensures that organization delivers new and enhanced services required by the customer, while protecting the integrity of existing services.

Aspects of Software Project Management:



Advantages of Software Project Management:

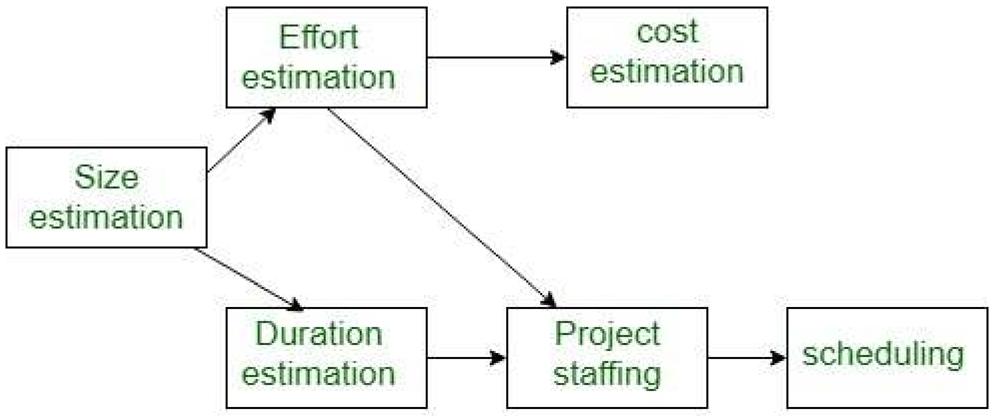
- It helps in planning of software development.
- Implementation of software development is made easy.
- Monitoring and controlling are aspects of software project management.
- It overall manages to save time and cost for software development.

Software Engineering | Project Planning

Once a project is found to be possible, computer code project managers undertake project designing. Project designing is undertaken and completed even before any development activity starts. Project designing consists of subsequent essential activities:

Estimating the subsequent attributes of the project:

- **Project size:**
  What's going to be downside quality in terms of the trouble and time needed to develop the product?
- **Cost:**
  What proportion is it reaching to value to develop the project?
- **Duration:**
  However long is it reaching to want complete development?
- **Effort:**
  What proportion effort would be required?
  - **Precedence ordering among project planning activities:**
    The different project connected estimates done by a project manager have already been mentioned. The below diagram shows the order during which vital project coming up with activities is also undertaken. It may be simply discovered that size estimation is that the 1st activity. It's conjointly the foremost basic parameter supported that all alternative coming up with activities square measure dispensed, alternative estimations like the estimation of effort, cost, resource, and project length also are vital elements of the project coming up with.
  -



Precedence ordering among planning activities

Line of Code

The basis of the Measure LOC is that program length can be used as a predictor of program characterictics such as effort and ease of maintenance. The LOC measure is used to measure size of the software.

There are versions of LOC :

DSI (Delivered Source Instructions)
It is used in COCOMO'81 as KDSI (Means thousands of Delivered Source Instructions). DSI is defined as follows :

- Only Source lines that are DELIVERED as part of the product are included -- test drivers and other support software is excluded
- SOURCE lines are created by the project staff -- code created by applications generators is excluded
- One INSTRUCTION is one line of code or card image
- Declarations are counted as instructions
- Comments are not counted as instructions

Advantages of LOC

1. Simple to measure

Drawbacks of LOC

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

Because of the critics above there have been extensive efforts to characterise other prodeuct size attributes, notably complexity and functionality.

Function Points (FP)

Function points [Albrecth 1979] is basic data from which productivity metrics could be computed.
FP data is used in two ways:

1. as an estimation variable that is used to "size" each element of the software,
2. as baseline metrics collected from past projects and used in conjuction with estimation variables to develop cost and effort projections.

The approach is to identify and count a number of unique function types:

- external inputs (e.g. file names)
- external outputs (e.g. reports, messages)
- queries (interactive inputs needing a response)
- external files or interfaces (files shared with other software systems)
- internal files (invisible outside the system)

Each of these is then individually assessed for complexity and given a weighting value which varies from 3 (for simple external inputs) to 15 (for complex internal files).

Unadjusted function points (UFP) is calculated as follows :
The sum of all the occurrences is computed by multiplying each function count with a weighting and then adding up all the values. The weights are based on the complexity of the feature being counted. Albrecht's original method classified the weightings as:

Function Type	**Low**	**Average**	**High**
External Input	x3	x4	x6
External Input	x4	x5	x7
Logical Internal File	x7	x10	x15
External Interface File	x5	x7	x10
External Inquiry	x3	x4	x6

Low, average and high decision can be determined with this table :

	1-5 Data element types	**6-19 Data elemet types**	**20+ Data elemet types**
0-1 File types referenced	Low	Low	Average
2-3 File types referenced	Low	Average	High
4+ File types referenced	Average	High	High

In order to find adjusted FP, UFP is multiplied by technical complexity factor (TCF) which can be calculated by the formula :

TCF = 0.65 + (sum of factors) / 100

There are 14 technical complexity factor. Each complexity factor is rated on the basis of its degree of influence, from no influence to very influencial :

1. Data communications
2. Performance
3. Heavily used configuration
4. Transaction rate
5. Online data entry
6. End user efficiency
7. Online update
8. Complex processing
9. Reusability
10. Installation ease
11. Operations ease
12. Multiple sites
13. Facilitate change
14. Distributed functions

Then `FP = UFP x TCF`

Function points are recently used also for real time systems.

Advantages of FP

i. It is not rescricted to code
ii. Language independent
iii. The necessary data is avaiable early in a project. We need onşy a detailed specification.
iv. More accurate than estimated LOC

Drawbacks of FP

i. Subjective counting
ii. Hard to automate and diffucult to compute
iii. Ignores quality of output
iv. Oriented to traditional data porcessing applications

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

$$E_i=a*(KDLOC)b$$

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 provide 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

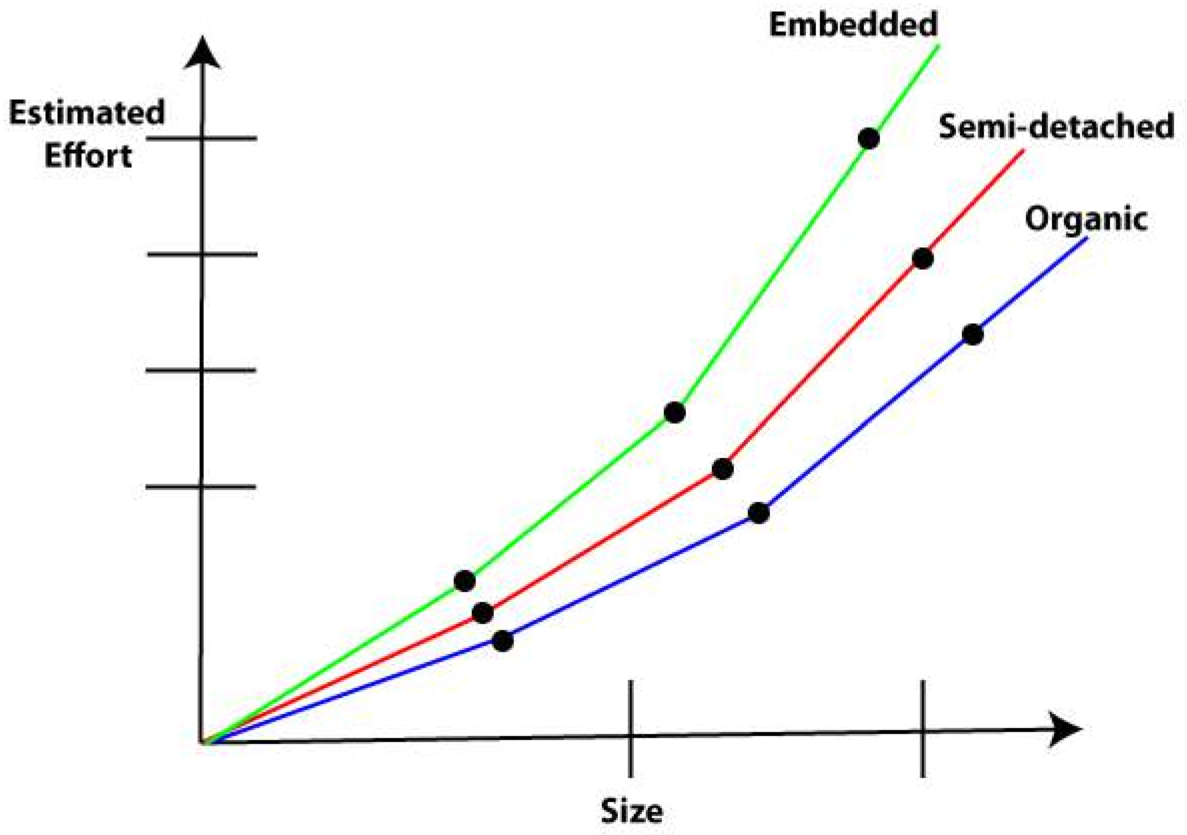
For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: Tdev = 2.5(Effort) 0.38 Months

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

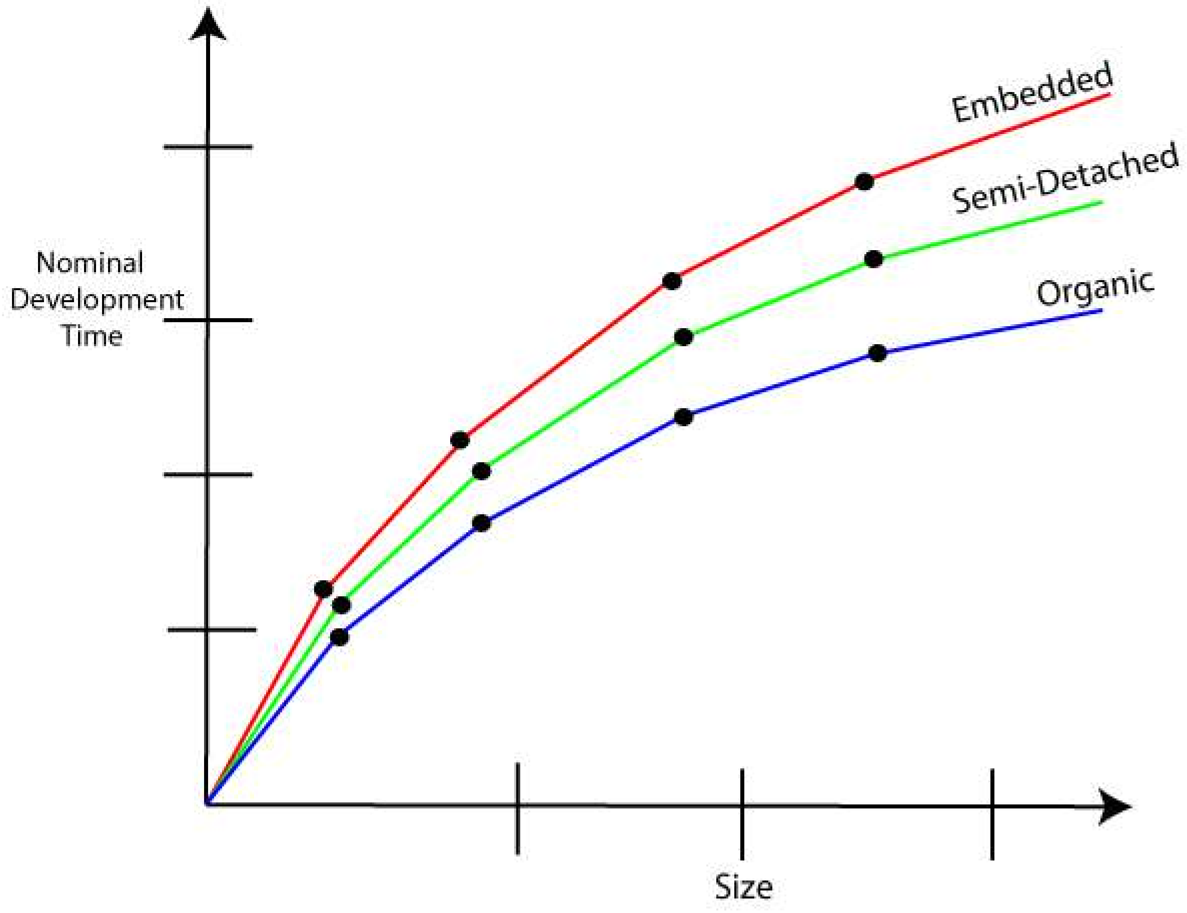
Embedded: Tdev = 2.5(Effort) 0.32 Months

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.



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

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.

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

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

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

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

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 -

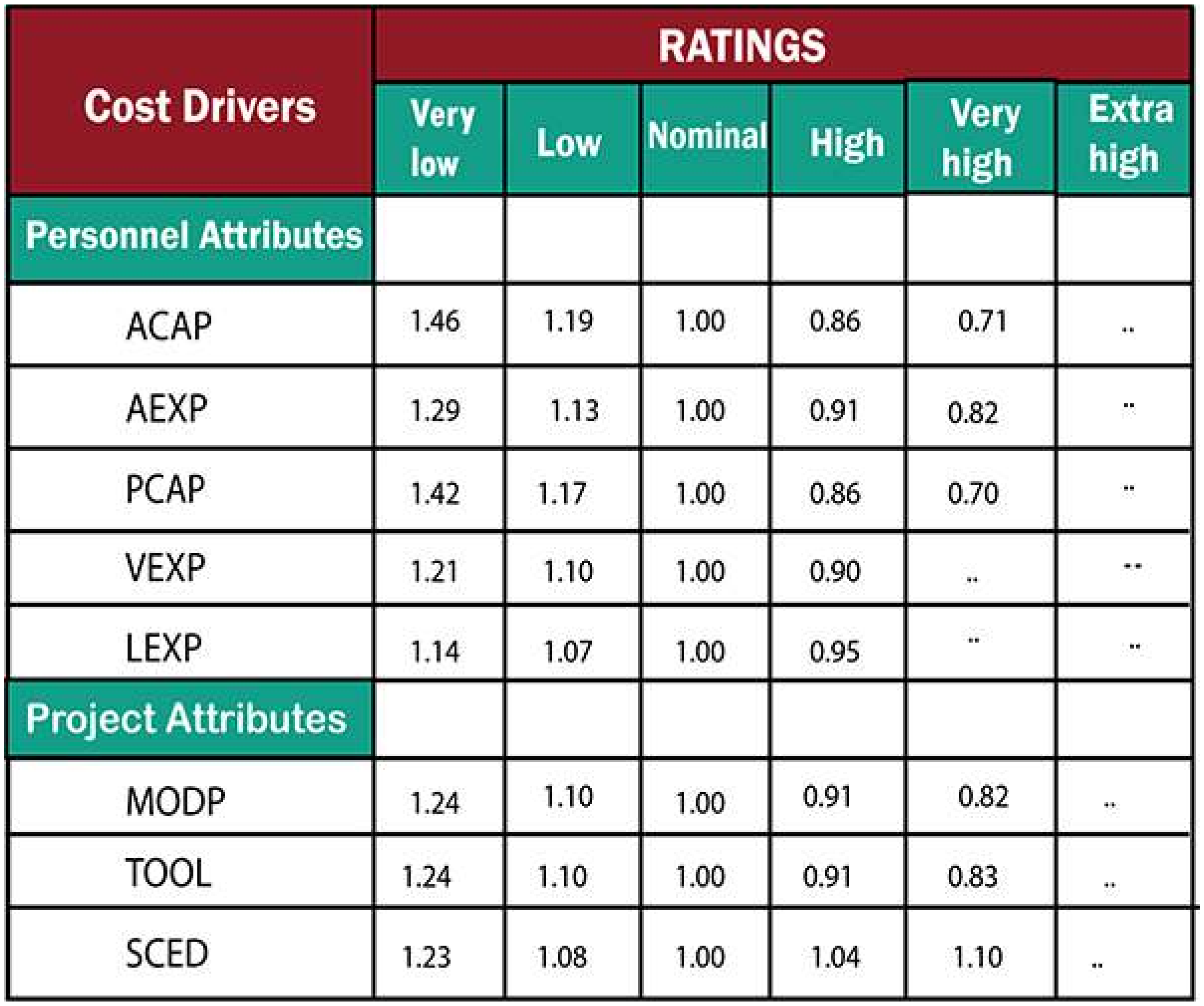
- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

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
Organic	2.4	1.05	2.5
Semidetached	3.0	1.12	2.5
Embedded	3.6	1.20	2.5

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.

Project Risk Management

Risk management involves all activities pertaining to identification, analyzing and making provision for predictable and non-predictable risks in the project. Risk may include the following:

- Experienced staff leaving the project and new staff coming in.
- Change in organizational management.
- Requirement change or misinterpreting requirement.
- Under-estimation of required time and resources.
- Technological changes, environmental changes, business competition.

Risk Management Process

There are following activities involved in risk management process:

- **Identification -** Make note of all possible risks, which may occur in the project.
- **Categorize -** Categorize known risks into high, medium and low risk intensity as per their possible impact on the project.
- **Manage -** Analyze the probability of occurrence of risks at various phases. Make plan to avoid or face risks. Attempt to minimize their side-effects.
- **Monitor -** Closely monitor the potential risks and their early symptoms. Also monitor the effects of steps taken to mitigate or avoid them.