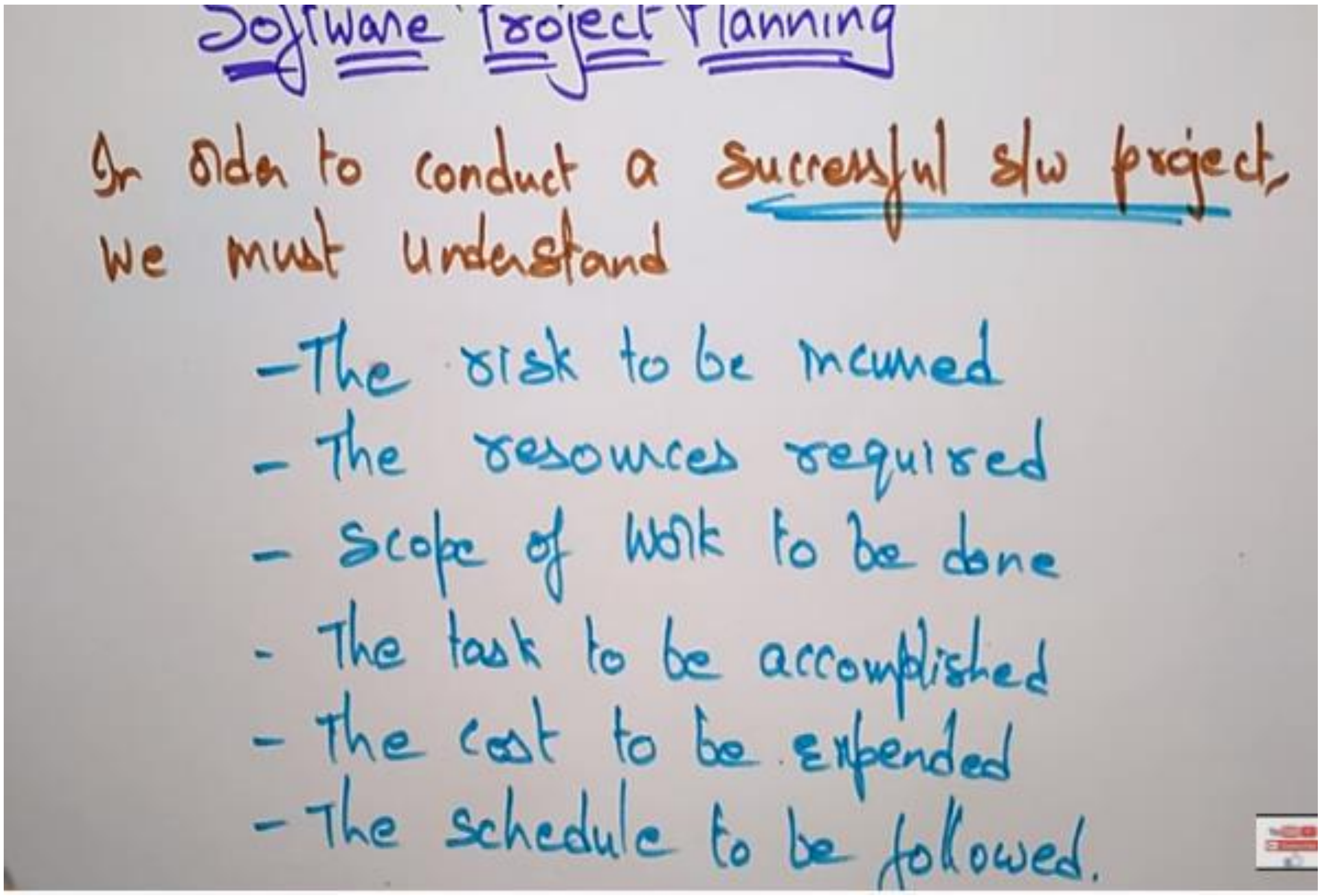
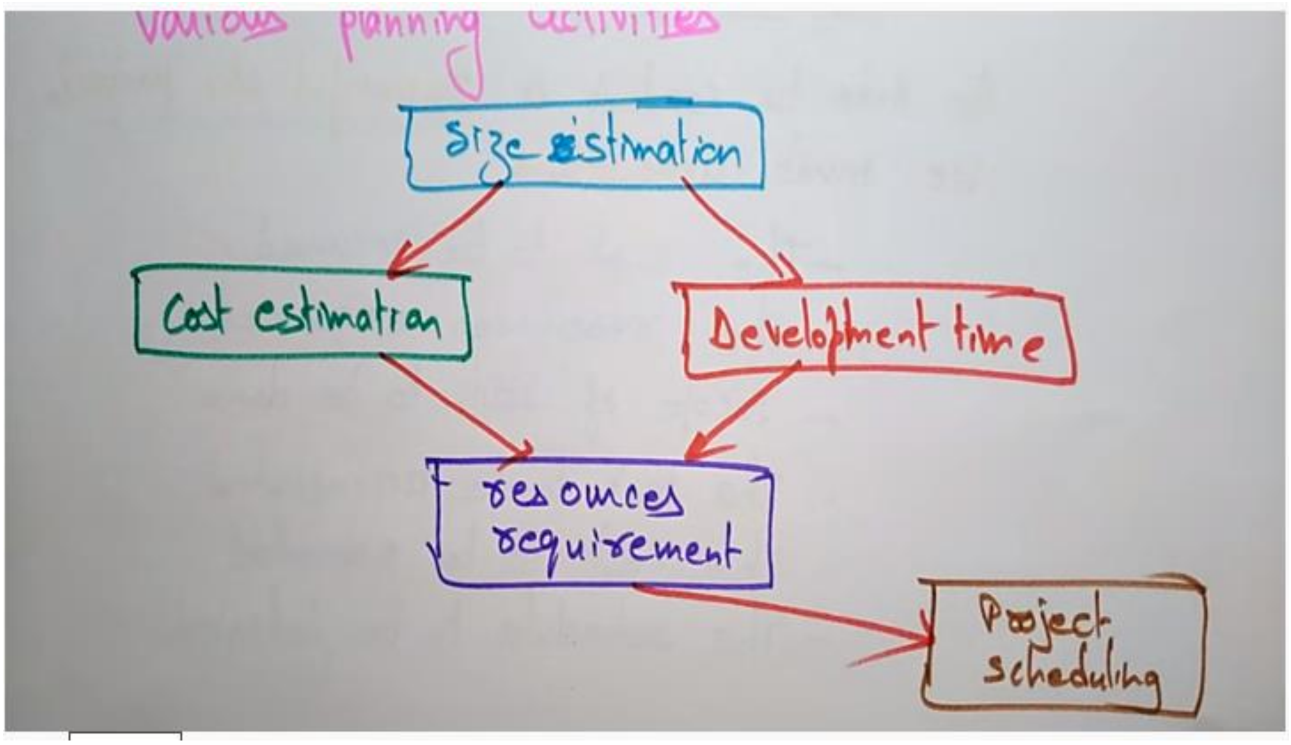
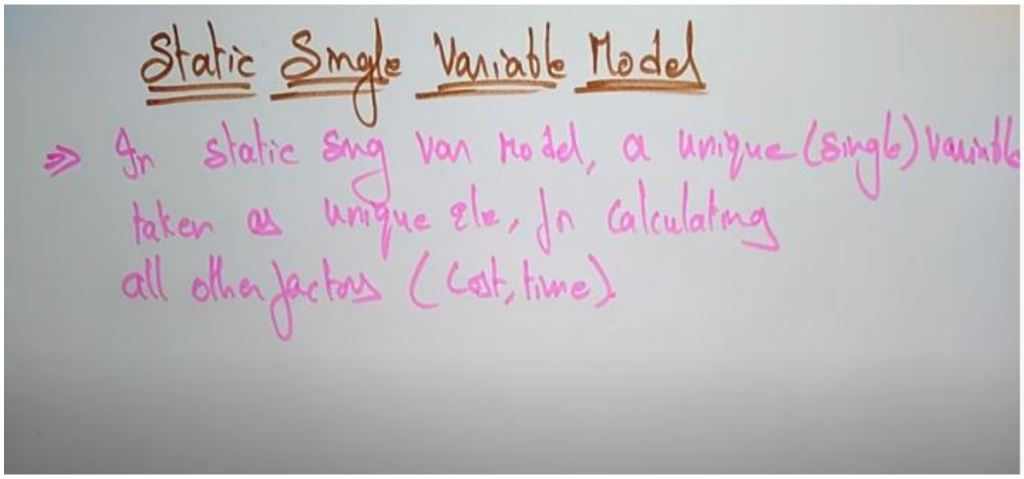


Software Project Planning

In order to conduct a successful s/w project, we must understand

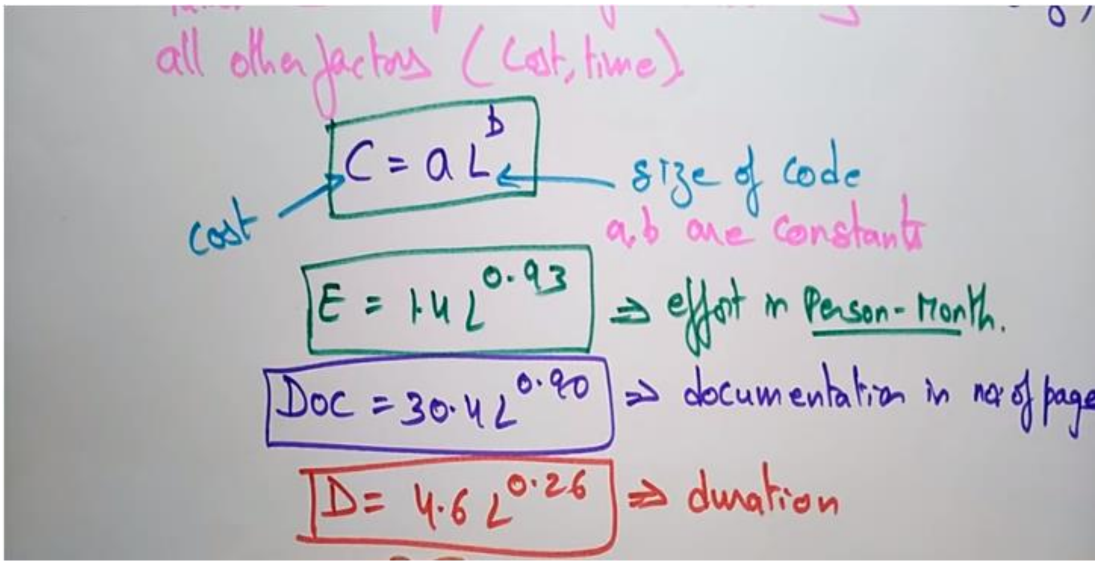
- The risk to be incurred
- The resources required
- scope of work to be done
- The task to be accomplished
- The cost to be expended
- The schedule to be followed.





Static Single Variable Model

⇒ In static sing var model, a unique (single) variable taken as unique ele, In calculating all other factors (cost, time)



all other factors (cost, time)

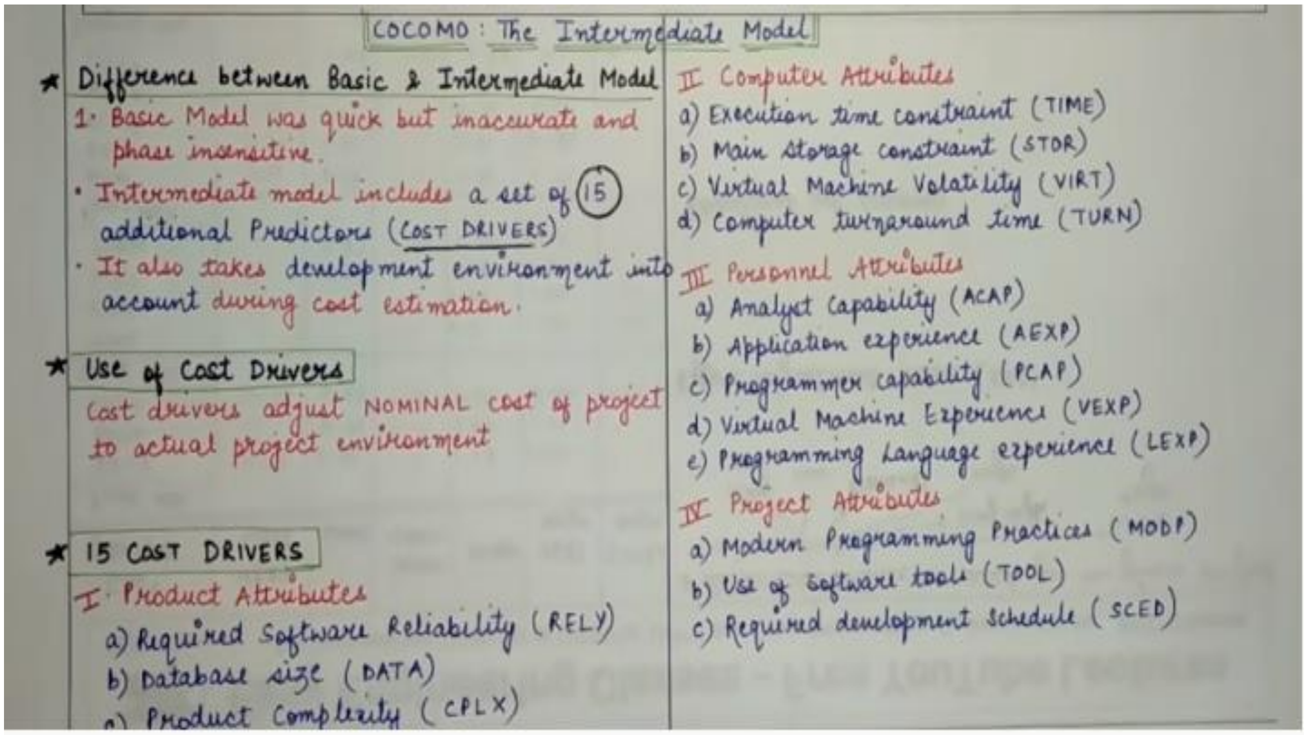
$\boxed{C = aL^{b}}$ cost; size of code

a,b are constants

$\boxed{E = 1.4L^{0.93}}$ ⇒ effort in Person-Month.

$\boxed{Doc = 30.4L^{0.90}}$ ⇒ documentation in no. of pages

$\boxed{D = 4.6L^{0.26}}$ ⇒ duration



COCOMO : The Intermediate Model

★ Difference between Basic & Intermediate Model

1. Basic Model was quick but inaccurate and phase insensitive.

• Intermediate model includes a set of 15 additional Predictors (COST DRIVERS)

• It also takes development environment into account during cost estimation.

★ Use of Cost Drivers

Cost drivers adjust NOMINAL cost of project to actual project environment

★ 15 COST DRIVERS

I. Product Attributes

a) Required Software Reliability (RELY)
b) Database size (DATA)
c) Product Complexity (CPLX)

II Computer Attributes

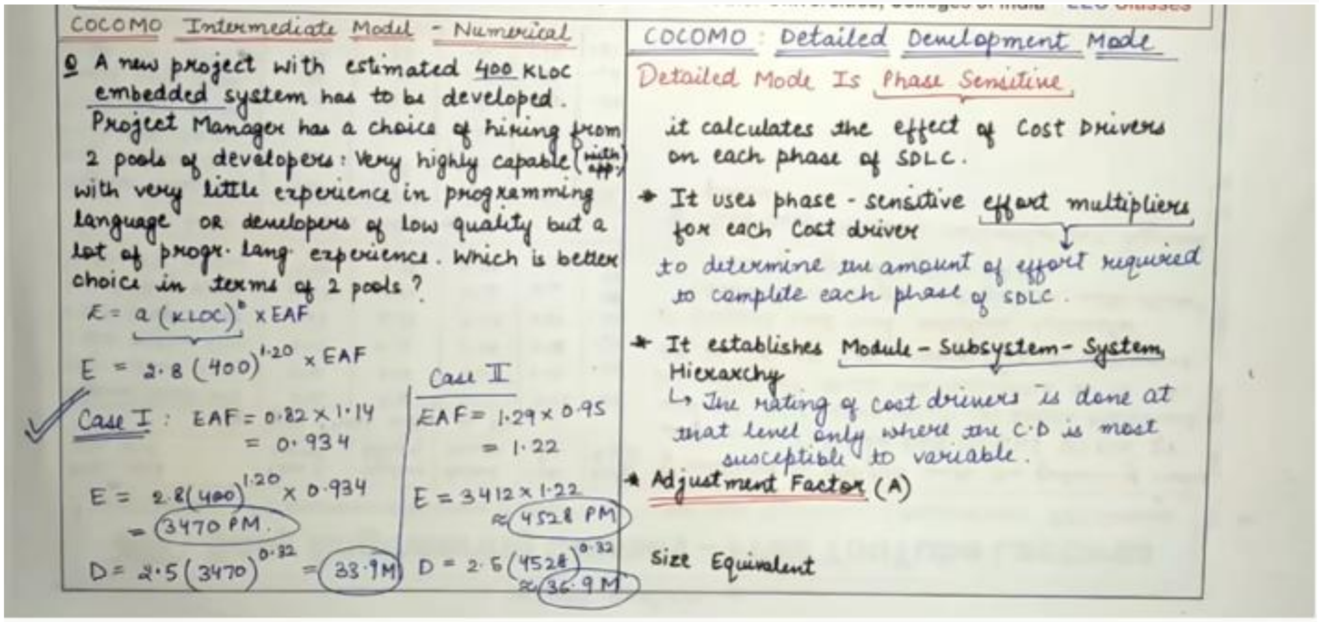
a) Execution time constraint (TIME)
b) Main Storage constraint (STOR)
c) Virtual Machine Volatility (VIRT)
d) Computer turnaround time (TURN)

III Personnel Attributes

a) Analyst Capability (ACAP)
b) Application experience (AEXP)
c) Programmer capability (PCAP)
d) Virtual Machine Experience (VEXP)
e) Programming Language experience (LEXP)

IV Project Attributes

a) Modern Programming Practices (MODP)
b) Use of Software tools (TOOL)
c) Required development schedule (SCED)



COCOMO Intermediate Model - Numerical

Q A new project with estimated 400 KLOC embedded system has to be developed. Project Manager has a choice of hiring from 2 pools of developers: Very highly capable (with app.) with very little experience in programming language OR developers of low quality but a lot of progr. lang. experience. Which is better choice in terms of 2 pools?

$E = \underbrace{a\,(KLOC)^b} \times EAF$

$E = 2.8\,(400)^{1.20} \times EAF$

✓ Case I : $EAF = 0.82 \times 1.14$
$= 0.934$

$E = 2.8(400)^{1.20} \times 0.934$
$= \boxed{3470\ PM}$

$D = 2.5\,(3470)^{0.32} = \boxed{33.9M}$

Case II

$EAF = 1.29 \times 0.95$
$= 1.22$

$E = 3412 \times 1.22$
$\approx \boxed{4528\ PM}$

$D = 2.5\,(4528)^{0.32}$
$\approx \boxed{36.9M}$

COCOMO : Detailed Development Mode

Detailed Mode Is $\underbrace{\text{Phase Sensitive}}$

it calculates the effect of Cost Drivers on each phase of SDLC.

★ It uses phase-sensitive $\underbrace{\text{effort multipliers}}$ for each Cost driver

to determine the amount of effort required to complete each phase of SDLC.

★ It establishes $\underbrace{\text{Module - Subsystem - System}}$ Hierarchy

↳ The rating of cost drivers is done at that level only where the C.D is most susceptible to variable.

★ Adjustment Factor (A)

Size Equivalent

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:

$$\textbf{Effort}=a_1*(KLOC)\ a_2\ PM$$
$$\textbf{Tdev}=b_1*(efforts)b_2\ Months$$

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,

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 -

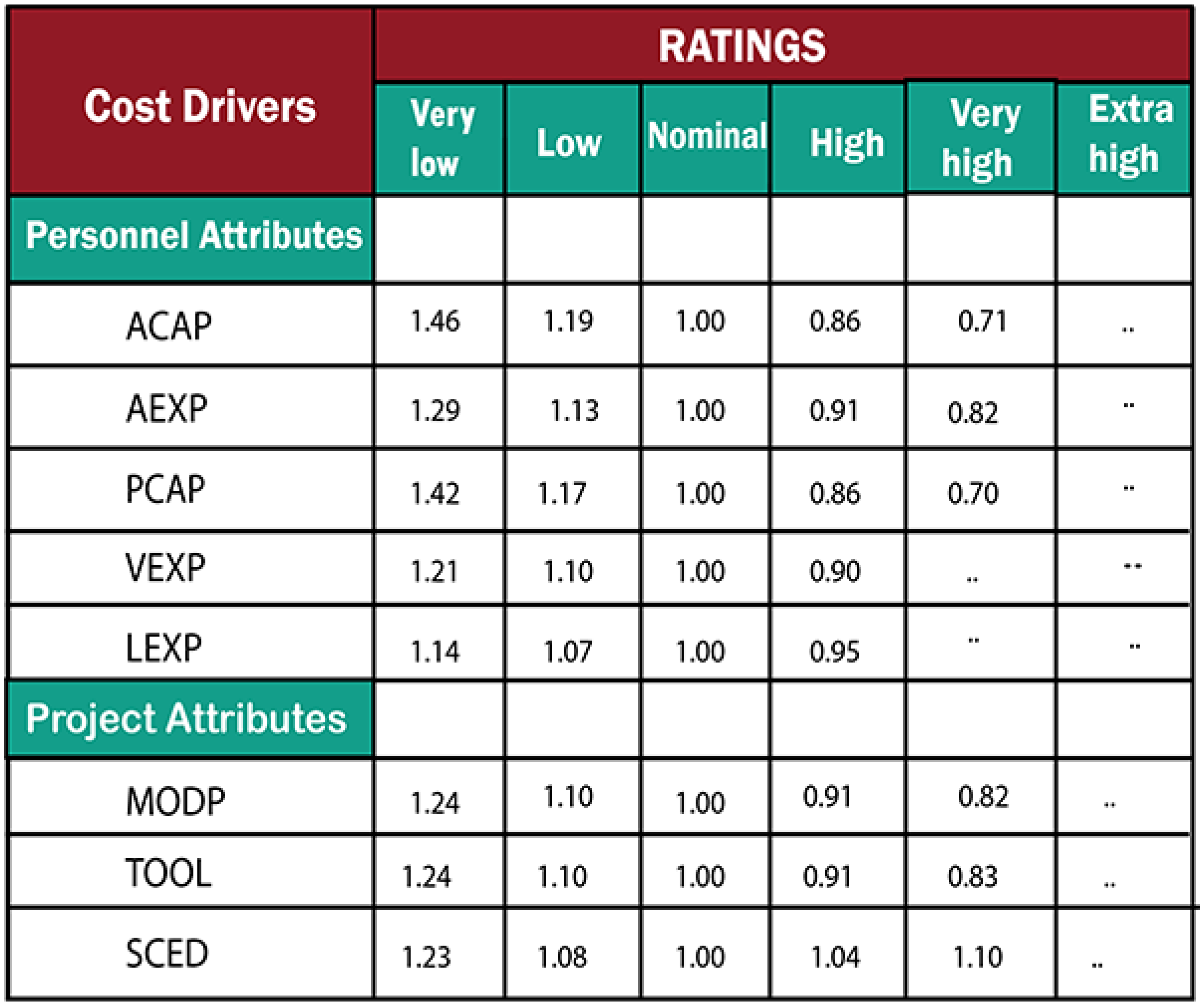
- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

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

Detailed Model

In detailed COCOMO, a large amount of work has been done by Boehm to cover all essential aspects of software development. It offers a medium to process all the project characteristics for calculating the software estimation. The detailed model introduces two more capabilities that are as follows:

Phase-sensitive effort multiplier:

Some phases (like design, programming, integration/test) are more affected than others by factors defined by the cost drivers. The detailed model provides a set of phase-sensitive effort multipliers for each cost driver. This helps in determining the workforce allocation for each phase of the project.

Three-level product hierarchy:

In the three-level product hierarchy, there are a module, subsystem, and system levels. The ratings of the cost driver are done at an appropriate level, i.e., the level at which it is most affected by the variation.

Development phase

Software development is done in four phases:

Plans/requirements:

This is the first phase of the software development life cycle. In this phase, the requirements are analyzed, the product plan is set up, and a full product specification is generated for software development. The developers have to give 6% to 8% of the effort and 10% to 40% of the development time.

Product design:

In this phase, a software developer starts thinking about “how.” How a software UI look like? How do they function? The software requirements required by the customers are kept in the mind of the designer while designing software. There are two types of designs that are designed by the designer; conceptual design and technical design. Conceptual design tells the customer what the system will exactly do. When the customer approves the design, this process is called validation. Then the conceptual design is converted into a more detailed form of technical design, which helps the developer to understand the hardware and software needs to fulfill the customer’s requirement.

Programming:

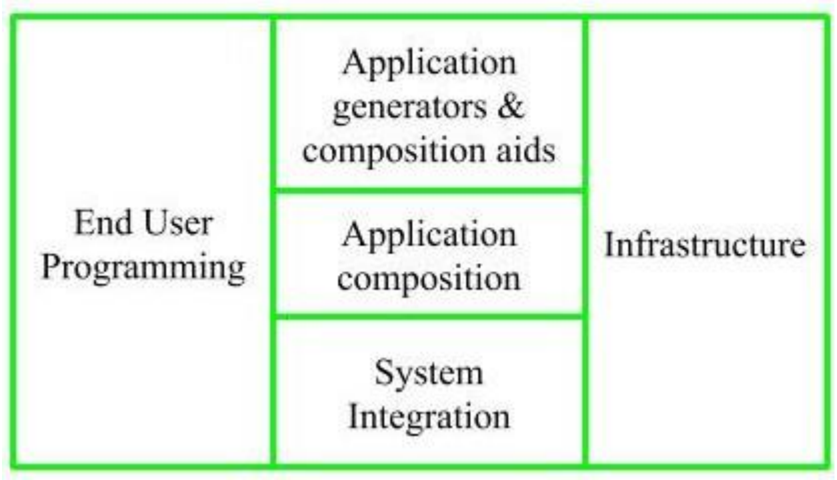
This is the phase where development starts. This phase is divided into two sub-phases: detailed design and code/unit test. This phase requires an effort from 48% to 68% and necessitate 24% to 64% of the development time to finish the coding.

Integration/Test:

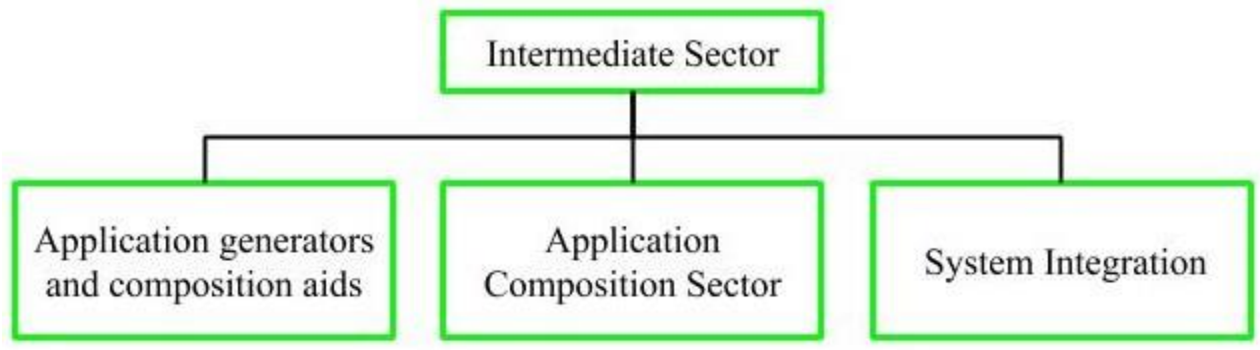
In the integration testing, individual units are combined and tested as a whole. The purpose of this level of testing is to expose faults in the interaction between integrated units.

COCOMO II Model

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



1. End User Programming:
Application generators are used in this sub-model. End user write the code by using these application generators.
Example – Spreadsheets, report generator, etc.
2. Intermediate Sector:


Intermediate Sector
Application generators and composition aids
Application Composition Sector
System Integration

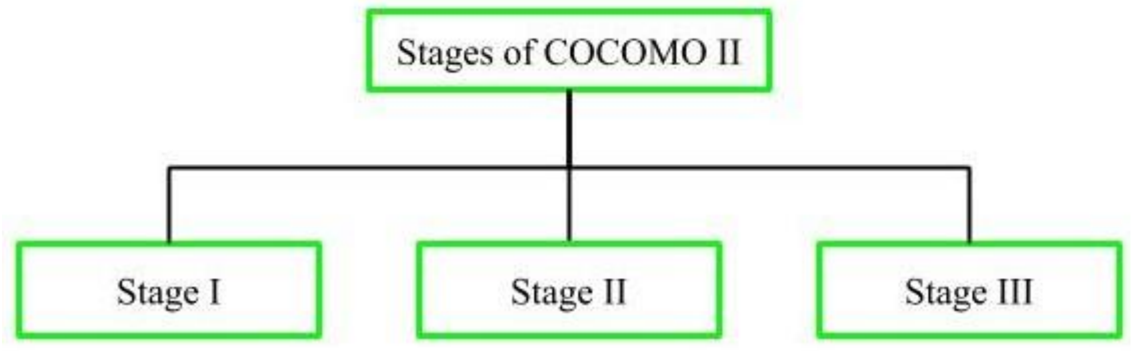
- **(a). Application Generators and Composition Aids –**
  This category will create largely prepackaged capabilities for user programming. Their product will have many reusable components. Typical

firms operating in this sector are Microsoft, Lotus,
Oracle, IBM, Borland, Novell.

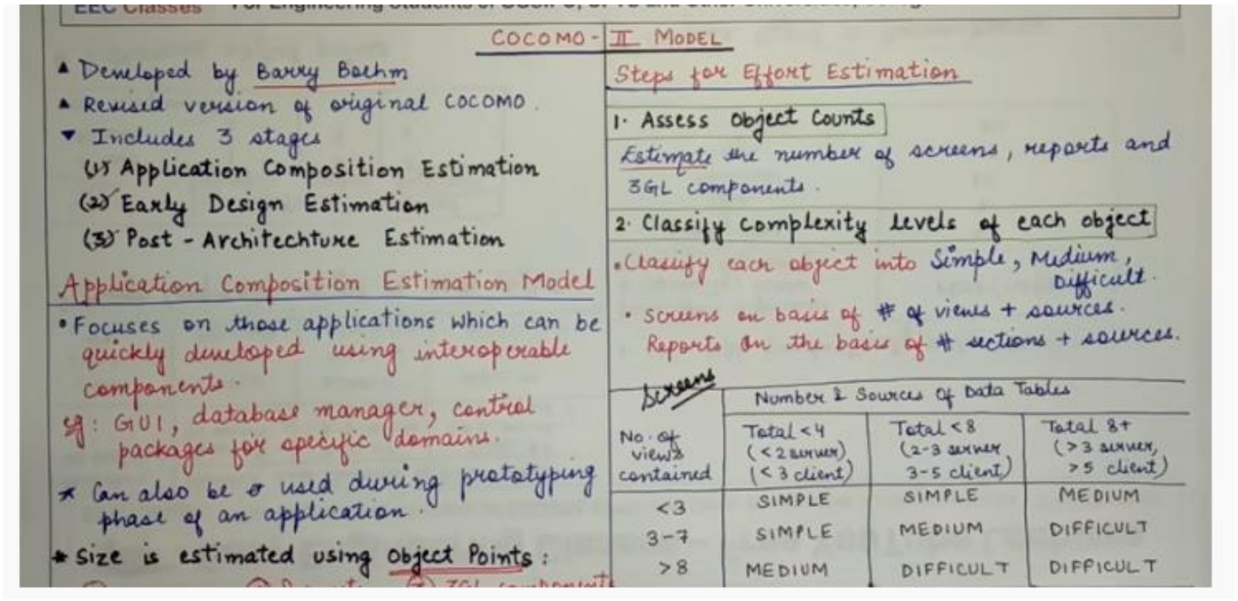
- **(b). Application Composition Sector –**
  This category is too diversified and to be handled by prepackaged solutions. It includes GUI, Databases, domain specific components such as financial, medical or industrial process control packages.
- **(c). System Integration –**
  This category deals with large scale and highly embedded systems.

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

Stages of COCOMO II:



1. **Stage-I:**
   It supports estimation of prototyping. For this it uses *Application Composition Estimation Model*. This model is used for the prototyping stage of application generator and system integration.
2. **Stage-II:**
   It supports estimation in the early design stage of the project, when we less know about it. For this it uses *Early Design Estimation Model*. This model is used in early design stage of application generators, infrastructure, system integration.
3. **Stage-III:**
   It supports estimation in the post architecture stage of a project. For this it uses *Post Architecture Estimation Model*. This model is used after the completion of the detailed architecture of application generator, infrastructure, system integration.



COCOMO - II MODEL

▲ Developed by Barry Boehm
▲ Revised version of original COCOMO.
▼ Includes 3 stages
(1) Application Composition Estimation
(2) Early Design Estimation
(3) Post - Architechture Estimation

Application Composition Estimation Model

• Focuses on those applications which can be quickly developed using interoperable components.
eg: GUI, database manager, control packages for specific domains.
* Can also be ~~o~~ used during prototyping phase of an application.
* Size is estimated using Object Points :

Steps for Effort Estimation

1. Assess Object Counts

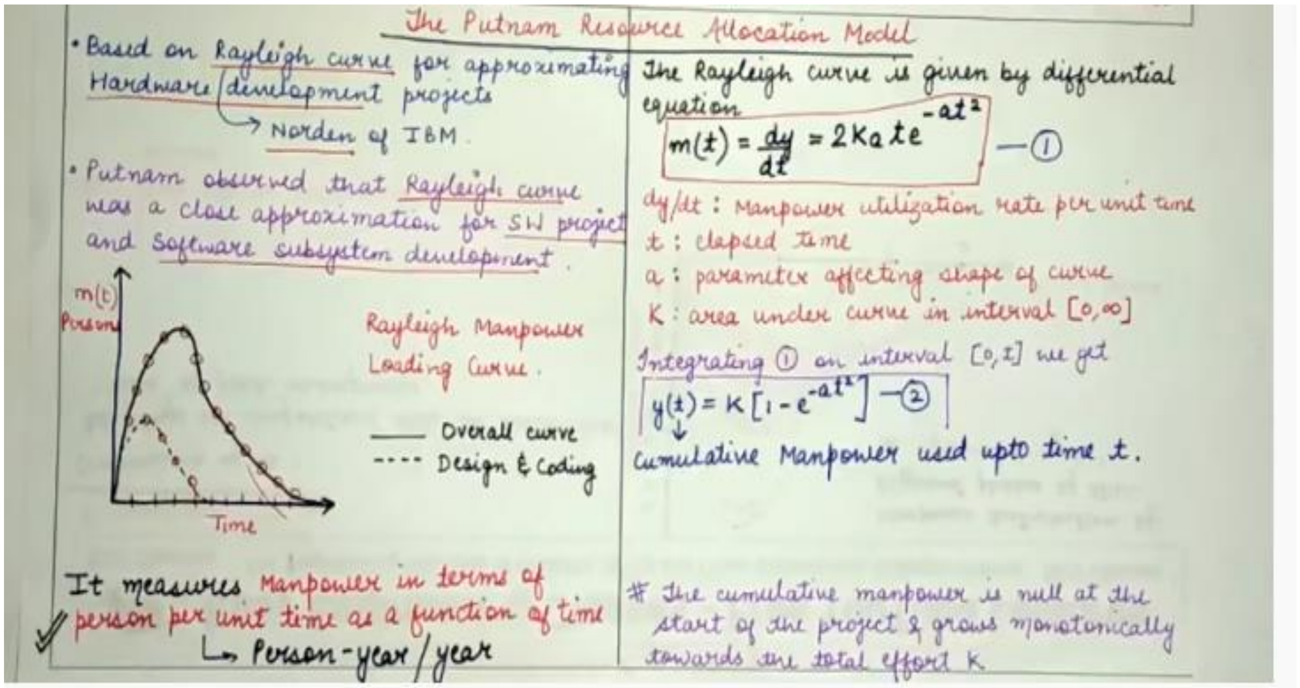
Estimate the number of screens, reports and 3GL components.

2. Classify complexity levels of each object

• Classify each object into Simple, Medium, Difficult.
• Screens on basis of # of views + sources.
Reports on the basis of # sections + sources.

Screens	Number & Sources of Data Tables		
No. of views contained	Total < 4 (< 2 server) (< 3 client)	Total < 8 (2-3 server 3-5 client)	Total 8+ (> 3 server, > 5 client)
< 3	SIMPLE	SIMPLE	MEDIUM
3-7	SIMPLE	MEDIUM	DIFFICULT
> 8	MEDIUM	DIFFICULT	DIFFICULT

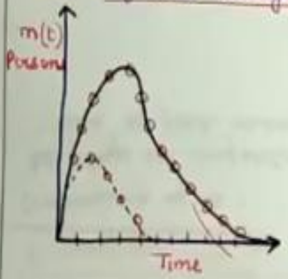
Putnam Resource Allocation Model



The Putnam Resource Allocation Model

• Based on Rayleigh curve for approximating Hardware development projects
→ Norden of IBM.

• Putnam observed that Rayleigh curve was a close approximation for SW project and Software subsystem development.



Rayleigh Manpower Loading Curve.

—— Overall curve
---- Design & Coding

It measures Manpower in terms of person per unit time as a function of time
↳ Person-year/year

The Rayleigh curve is given by differential equation

$$\boxed{m(t) = \frac{dy}{dt} = 2Kat\,e^{-at^2}} \quad \text{—①}$$

dy/dt : Manpower utilization rate per unit time
t : elapsed time
a : parameter affecting shape of curve
K : area under curve in interval [0, ∞]

Integrating ① on interval [0, t] we get

$$\boxed{y(t) = K\left[1 - e^{-at^2}\right] \text{—②}}$$

↓ Cumulative Manpower used upto time t.

The cumulative manpower is null at the start of the project & grows monotonically towards the total effort K.

Putnam Staffing Estimation/RESOURCE ALLOCATION

After successfully determining the *effort needed to develop the required software product* the focus should be on ***determining the staffing requirements*** or **resource allocation**.

Putnam studied the complete staffing problem to find out the proper solution and pattern for staffing related issues.

In his study he used and extended the ***work of Norden*** who had already investigated the staffing pattern of general research and development (R&D) projects.

Putnam's staffing work

Putnam while studying the problem of *staffing or resource allocation* for software project learned that they have characteristics similar to any other research and development (R&D) projects studied by Norden.

Putnam stated that *Rayleigh-Norden curve* can be used to relate the number of delivered lines of code to the effort and the time required to develop the product.

Given below is *Putnam's expression*:

$L = C_k K^{1/3} t_d^{4/3}$

Here,
K – total effort expended
L – product size in KLOC
t_d – time expended in system integration and testing. It is time required to develop the software.
C_k – state of technology constant and reflects constraints that impede the progress of programmer. (Typically value of Ck is **2** for **poor development environment**, **8** for **good software development environment** and **11** for **excellent development environment**)

*So according to **Putnam estimation model**,*

- *staff build up should follow the Rayleigh curve.*
- *At the very beginning of the project only a small number of developers are needed.*
- *As the project progress the resource allocation requirement starts increasing and reaches at its peak during testing phase.*
- *After the implementation and unit testing is done the staff requirement again starts falling.*

Putnam staffing estimation for *resource allocation* is popularly used model in software engineering.

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

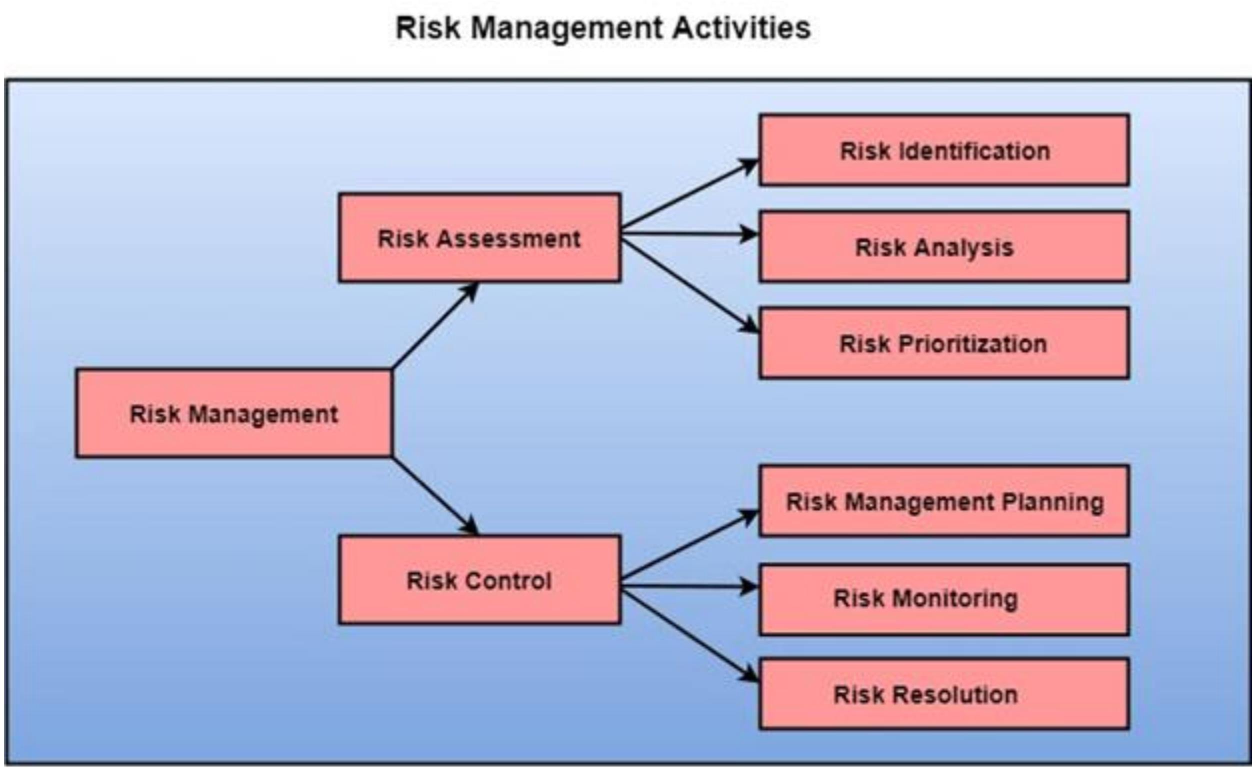
1. **1. Known risks:** Those risks that can be uncovered after careful assessment of the project program, the business and technical environment in which the plan is being developed, and more reliable data sources (e.g., unrealistic delivery date)
2. **2. Predictable risks:** Those risks that are hypothesized from previous project experience (e.g., past turnover)
3. **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).

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.