RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Module – II Management

Lecture - 1

Topics are:

Functions
Software Planning
Productivity Metrics
Cost Estimation – COCOMO 81, COCOMO – Intermediate, COCOMO II

Presented by,
M. Viju Prakash., M.E., Ph.D., MISTE.,
Assistant Professor,
Rajagiri School of Engineering and Technology, Kochi.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

What is Software Engineering?

Program vs. Software

- Program is set of instructions written for a specific purpose.
- Software is the combination of program(s), documentation (documents produced during development) and operating procedure manuals (delivered with programs to customer at the time of release).

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Introduction
(Taken from Software Engineering – Sommerville)

Software project management is the art and science of planning and leading software projects. It is a sub-discipline of project management in which software projects are planned, implemented, monitored and controlled.

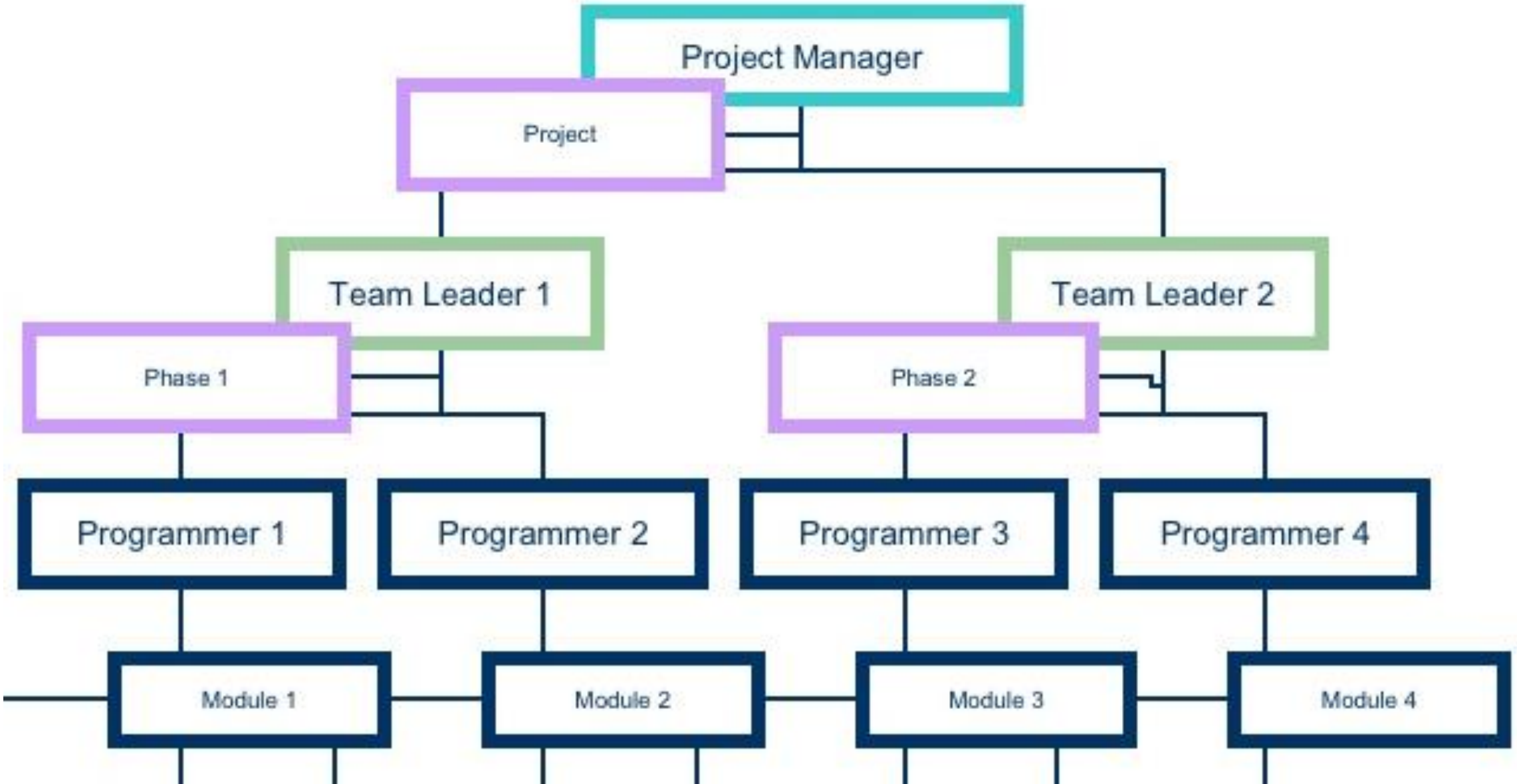
Interesting fact is, good management too cannot guarantee project success.

However, bad management usually results in project failure: The software is delivered late, costs more than estimated and fails to meet customer requirements.

Project managers take major role for planning and scheduling project development.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Team hierarchy



RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Introduction
(Taken from Software Engineering – Sommerville)

Software engineering is different from other types of engineering in a number of ways:

1. The product is intangible (Software managers cannot be progress)
2. There are no standard software processes (Lack of standard methodology)
3. Large software projects are often *'one-off'* projects (Managers may not predict the upcoming problems in large projects. Lessons learned from previous projects may not be transferable to new projects)

Because of these problems, some software projects are late, over budget and behind schedule. Software systems are always new & innovative.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Management - functions

1. Proposal writing
2. Project planning and scheduling
3. Project cost estimation
4. Project monitoring and reviews
5. Personnel selection and evaluation
6. Report writing and presentations

Less-than-ideal

In most cases, managers have to settle for a less-than-ideal project team. The reasons for this are:

1. The project budget may not cover the use of highly paid staff. Less experienced, less well-paid staff may have to be used.

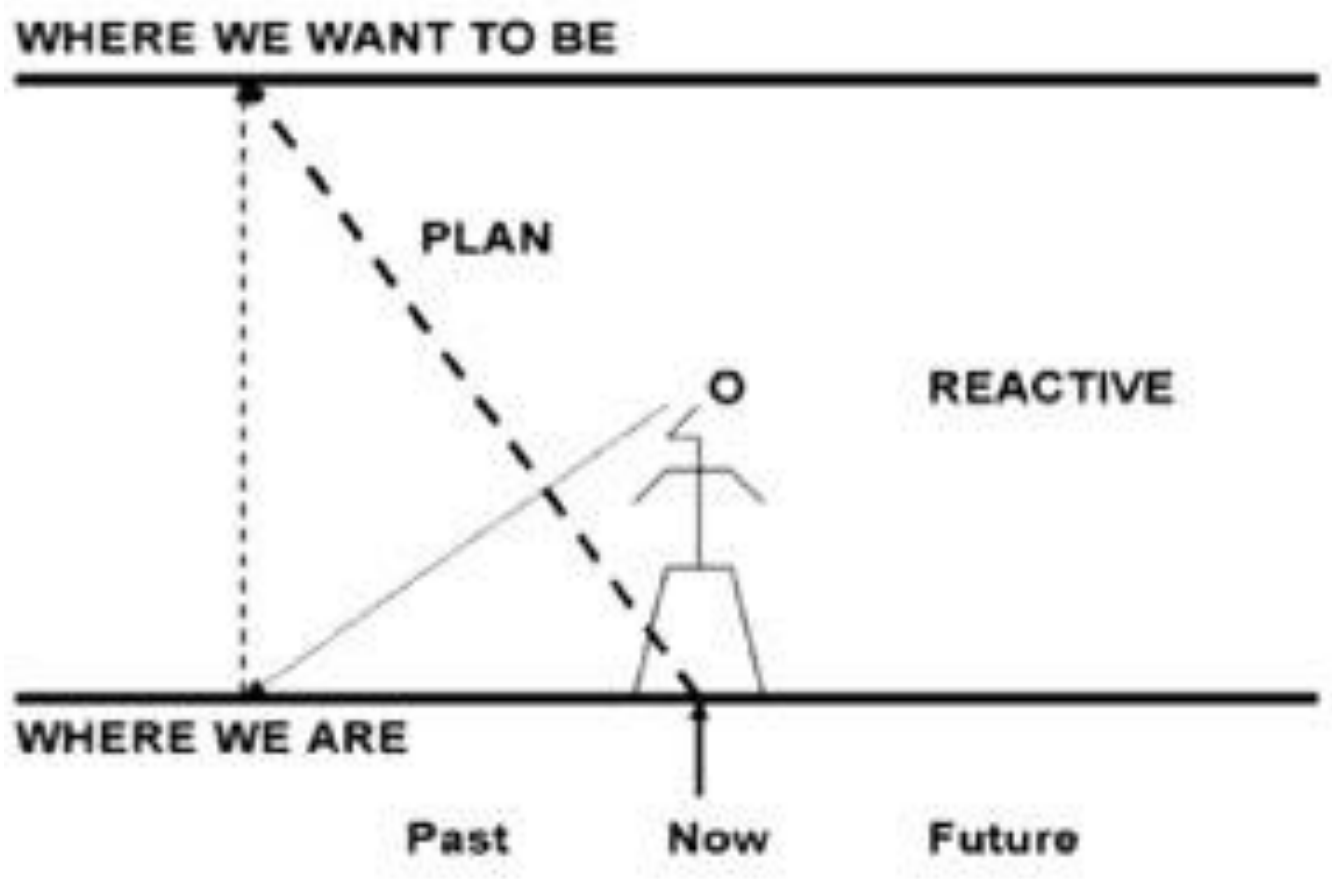
2. Staff with the appropriate experience may not be available either within an organization or externally. It may be impossible to recruit new staff to the project. Within the organization, the best people may already be allocated to other projects.

3. The organization may wish to develop the skills of its employees. Inexperienced staff may be assigned to a project to learn and to gain experience.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Project planning

Effective management of a software project depends on thoroughly planning the progress of the project. A plan, drawn up at the start of a project, should be used as the driver for the project. This initial plan must be best.



Task set for project planning

1. Establish project scope
2. Determine feasibility
3. Analyze risks
4. Define required resources
    1. Determine human resources required
    2. Define usable software resources
    3. Identify environmental resources
5. Estimate cost and effort
    1. Decompose the problem
    2. Develop two or more estimates using size, function points, process tasks or use-cases.
    3. Reconcile the estimates.
6. Develop a project schedule
    1. Establish a meaningful task set
    2. Define a task network
    3. Use scheduling tools to develop a timeline chart
    4. Define schedule tracking mechanisms.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

How to describe a project scope?

Software scope describes the functions and features that are to be delivered to end-users.

A software scope can be defined in one of two ways:

1. A narrative description of software scope is developed after communication with all stakeholders.
2. A set of use-cases is developed by end-users.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Few points about resources

The second planning task is estimation of the resources required to accomplish the software development effort.

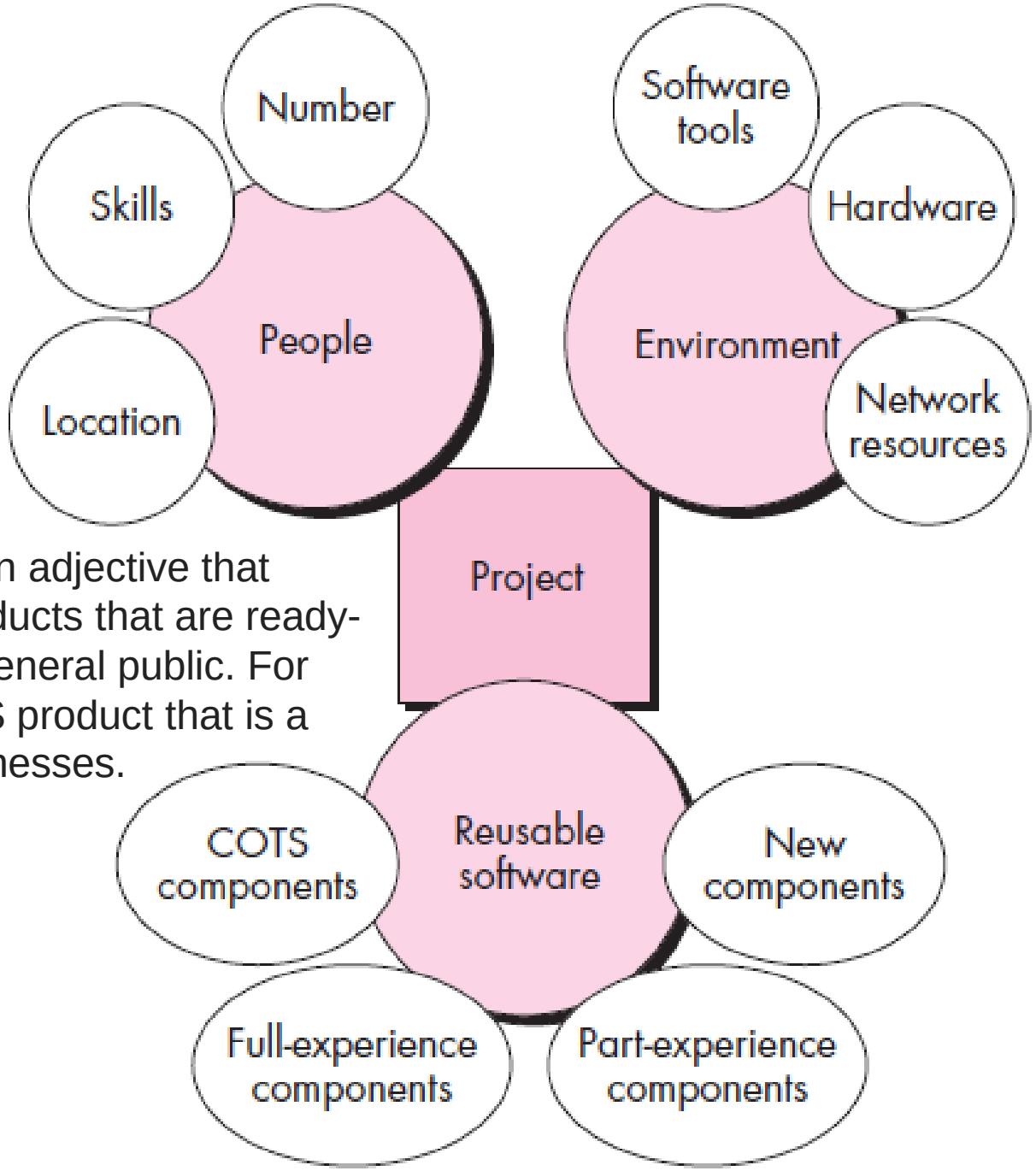
Each resource is specified with four characteristics:

Description of the resource

A statement of availability

Time when the resource will be required

Duration of time that resource will be applied



Short for commercial off-the-shelf, an adjective that describes software or hardware products that are ready-made and available for sale to the general public. For example, Microsoft Office is a COTS product that is a packaged software solution for businesses.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Project planning – Types of plan

Quality plan – Describes the quality procedures and standards that will be used in a project.

Validation plan – Describes the approach, resources and schedule used for system validation.

Configuration management plan – Describes the configuration management procedures and structures to be used.

Maintenance plan – Predicts the maintenance requirements of the system, maintenance costs and effort required.

Staff development plan – Describes how the skills and experience of the project team members will be developed.

Pseudo-code for Project planning

```
Establish the project constraints
Make initial assessments of the project parameters
Define project milestones and deliverables
while project has not been completed or cancelled loop
        Draw up project schedule
        Initiate activities according to schedule
        Wait (for a while)
        Review project progress
        Revise estimates of project parameters
        Update the project schedule
        Renegotiate project constraints and deliverables
        if (problems arise) then
                Initiate technical review and possible revision
        end if
end loop
```

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Various sections in project plan

The project plan sets out the resources available to the project, the work breakdown and a schedule for carrying out the work. In some organizations, the project plan is just a single sheet which includes different types of plan.

However, a project plan should include the following sections:

1. Introduction – This briefly describes the objectives of the project and sets out constraints.
2. Project organizations – This describes the way in which the development team is organized.
3. Risk analysis – This describes possible project risks, their probabilities and risk reduction strategies.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Various sections in project plan

4. Hardware and software resource requirements – This describes the hardware and software required to carry out the development.

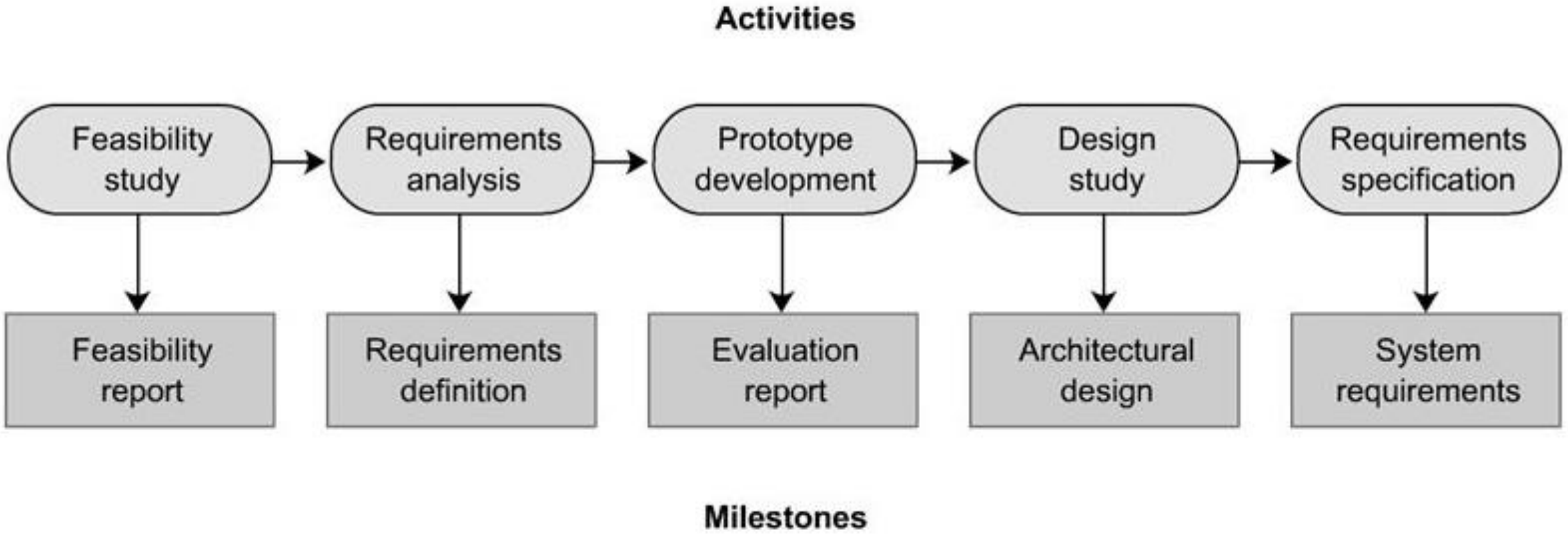
5. Work breakdown – This describes the breakdown of the project into activities and identifies the milestones and deliverables.

6. Project schedule – This shows the dependencies between activities, the estimated time required to reach milestone and allocation of people to activities.

7. Monitoring and reporting mechanisms – This defines the management reports that should be produced, when these should be produced and the project monitoring mechanisms used.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Milestones and deliverables



RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Software productivity

In standard economic terms, productivity is the ratio between the amount of goods or services produced and the labour or expense that goes into producing them.

In order to define software productivity, we must first establish a definition of software. At its most fundamental level, software is a computer program comprised of lines of code.

However, lines of code (LOC), is not the primary deliverable of a software project and customers often do not know how many lines of code are in the software they are buying.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Software productivity

A broader definition of software encompasses not only the computer program, but also the related procedures and documentation associated with the program. This often includes documentation of requirements, specifications, software design, and end-user procedures. The complete set of documentation provides a **more tangible** deliverable of a software project than does the program itself.

However, even though program code and documentation are the primary outputs of software production, they are not of direct interest to the software consumer. Software is bought based on what it can do, not on how it was coded or documented.

This means that the economic value of goods and services consumed is not measured in the same units as the natural units of production.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Software productivity

Subsequently, a different measure of software needs to be used in order to get a meaningful definition of software productivity.

This measure needs to reflect the utility value of the software, to wit the function that the software is intended to perform.

How is Software Productivity Measured?

There are various methods by which software productivity is measured, but whichever method is employed the goal should be uniform: **to give software managers and professionals a set of useful, tangible data points for sizing, estimating, managing, and controlling software projects with rigor and precision.**

How is Software Productivity Measured?

Software productivity measures are categorized in two ways:

1. **Size-related metrics:** These are related to the size of some output from an activity. The most commonly used size-related metric is lines of delivered source code. Other metrics that may be used are the number of delivered object code instructions or the number of pages of system documentation.

2. **Function-related metrics:** These are related to the overall functionality of the delivered software. Productivity is expressed in terms of the amount of useful functionality produced in some given time. Function points and application points are the best-known metrics of this type.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Size oriented metrics

Size oriented software metrics [Lines of code (LOC) or Source lines of code (SLOC)] are derived by normalizing quality and/or productivity measures by considering size of the software that has been produced.

Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
•	•	•	•	•	•		
•	•	•	•	•	•		
•	•	•	•	•	•		

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Size oriented metrics - Example

Consider this snippet of C code as an example:

```
for(i=0; i<100; i++) printf("hello");  /* How many lines of code is this? */
```

In this example we have:

- **1 Physical Line of Code (LOC)**
- **2 Logical Lines of Code (LLOC) (for statement and printf statement)**
- **1 comment line**

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Size oriented metrics - Example

Let us rearrange the snippet of C code now:

```
/* How many lines of code is this? */
{
for(i=0; i<100; i++)
}
printf("hello");
```

In this example we have:

- **5 Physical Line of Code (LOC)**
- **2 Logical Lines of Code (LLOC) (for statement and printf statement)**
- **1 comment line**

Problems in Size oriented metrics

1. LOC / SLOC is a poor productivity measure of individuals

2. LOC / SLOC is particularly ineffective at comparing programs written in different languages unless adjustment factors are applied to normalize languages.

Example is given.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Problems in Size oriented metrics

C	COBOL

`# include <stdio.h>`
 	
`int main() {`
`  printf("\nHello world\n");`
`}`
	`000100 IDENTIFICATION DIVISION.`
	`000200 PROGRAM-ID. HELLOWORLD.`
	`000300`
	`000400*`
	`000500 ENVIRONMENT DIVISION.`
	`000600 CONFIGURATION SECTION.`
	`000700 SOURCE-COMPUTER. RM-COBOL.`
	`000800 OBJECT-COMPUTER. RM-COBOL.`
	`000900`
	`001000 DATA DIVISION.`
	`001100 FILE SECTION.`
	`001200`
	`100000 PROCEDURE DIVISION.`
	`100100`
	`100200 MAIN-LOGIC SECTION.`
	`100300 BEGIN.`
	`100400     DISPLAY " " LINE 1 POSITION 1 ERASE EOS.`
	`100500     DISPLAY "Hello world!" LINE 15 POSITION 10.`
	`100600     STOP RUN.`
	`100700 MAIN-LOGIC-EXIT.`
	`100800     EXIT.`

Lines of code: 4 (excluding whitespace)	Lines of code: 17 (excluding whitespace)

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Function oriented metrics

Function oriented software metrics use a measure of the functionality delivered by the application as a normalization value. The most widely used function oriented metric is Function points.

Function points are used to

(1) estimate the cost or effort required to design, code and test the software.

(2) Predict the number of errors that will be encountered during testing

(3) Forecast the number of components and/or the projected score lines in the implemented system.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Function oriented metrics

Function points are derived using an empirical relationship based on information domain and assessment of software complexity.

Information domain can be calculated based on following manner.

Number of external inputs (EIs)

Number of external outputs (EOs)

Number of external inquiries (EQs)

Number of internal logical files (ILFs)

Number of external interface files (EIFs)

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Function oriented metrics

Once these data have been collected, the following table is made.

			Weighting factor				
Measurement parameter	**Count**		**Simple**	**Average**	**Complex**		
Number of user inputs		x	3	4	6	=	
Number of user outputs		x	4	5	7	=	
Number of user inquiries		x	3	4	6	=	
Number of files		x	7	10	15	=	
Number of external interfaces		x	5	7	10	=	
Count total							

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Function oriented metrics

To compute function points, the following relationship is used.

FP = Count total * [0.65+0.01*∑(F_i)]

Where as count total is the sum of all FP entries.

The F_i (i = 1 to 14) are value adjustment factors (VAF), based on responses to the following questions.

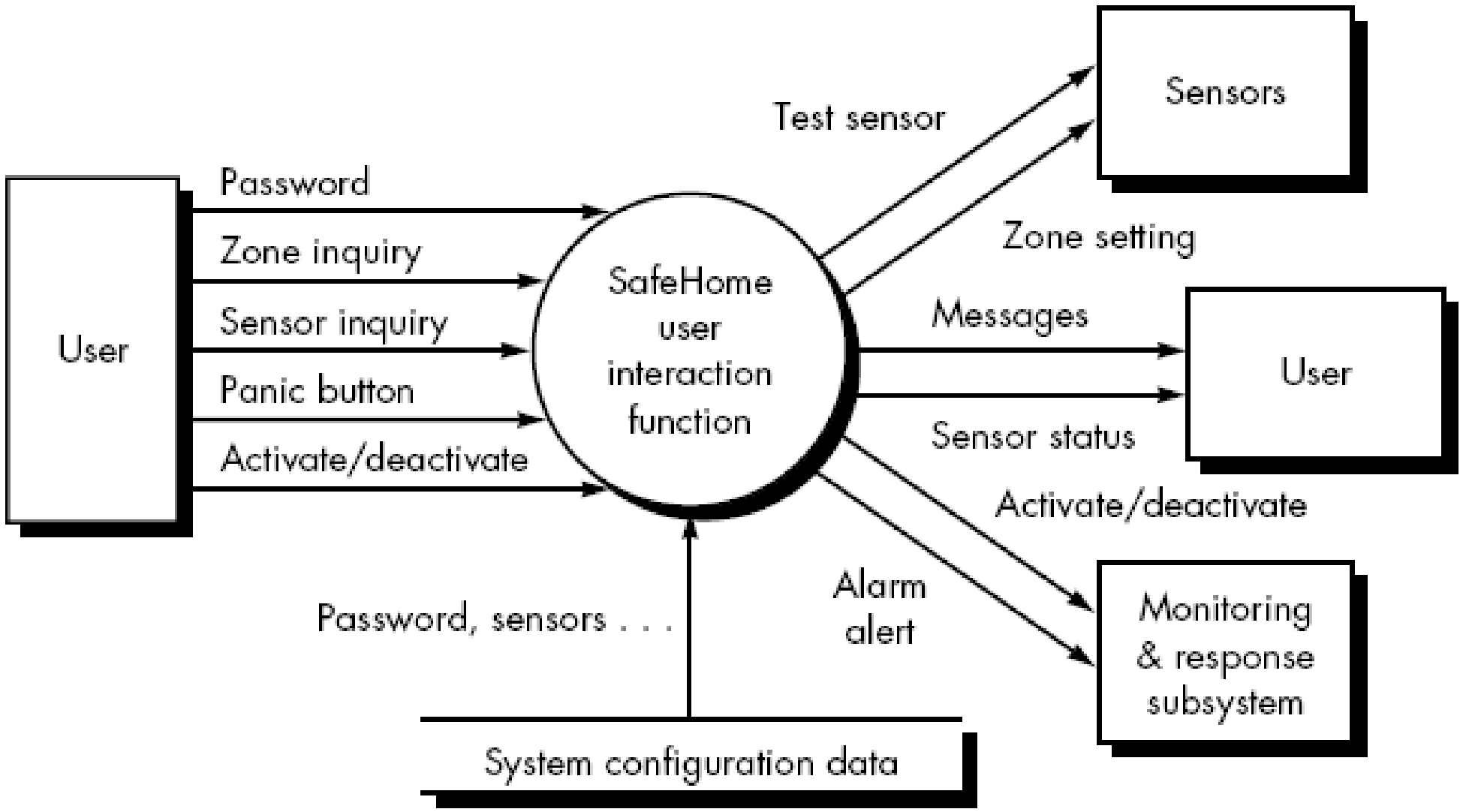
Rate each factor on a scale of 0 to 5.

Number of factors considered (F_i)

1. Does the system require reliable backup and recovery ?
2. Is data communication required ?
3. Are there distributed processing functions ?
4. Is performance critical ?
5. Will the system run in an existing heavily utilized operational environment ?
6. Does the system require on line data entry ?
7. Does the on line data entry require the input transaction to be built over multiple screens or operations ?
8. Are the master files updated on line ?
9. Is the inputs, outputs, files, or inquiries complex ?
10. Is the internal processing complex ?
11. Is the code designed to be reusable ?
12. Are conversion and installation included in the design ?
13. Is the system designed for multiple installations in different organizations ?
14. Is the application designed to facilitate change and ease of use by the user ?

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Function oriented metrics - Example



RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Function oriented metrics - Example

External inputs	: Password, panic button, activate/deactivate.	(3)
External outputs	: messages, sensor status	(2)
External inquires	: zone inquiry, sensor inquiry.	(2)
Internal logical files	: System configuration file	(1)
External interface files	: test sensor, zone setting, activate/deactivate, alarm alert	(4)

Assume weight factor is : Simple.

Then the function point table is,

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Function oriented metrics - Example

Information Domain Value	Count		Weighting factor Simple	Average	Complex		
External Inputs (EIs)	3	×	3	4	6	=	9
External Outputs (EOs)	2	×	4	5	7	=	8
External Inquiries (EQs)	2	×	3	4	6	=	6
Internal Logical Files (ILFs)	1	×	7	10	15	=	7
External Interface Files (EIFs)	4	×	5	7	10	=	20
Count total	→						50

Assume that (*Fi*) is 46 (a moderately complex product). Therefore, FP= 50 * [0.65 + (0.01 * 46)] = 56

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Software cost estimation

There is no simple way to make an accurate estimate of the effort required to develop a system. You may have to make initial estimates on the basis of high level user requirement.

Problems in cost estimation at an early stage:

1. The software may have to run on unfamiliar computers or demand a new technology.
2. The people involved in the project and their skills will probably not to be known.

Why software cost estimation is needed?

- Interesting fact is, 55% of projects over budget.
- 53% of projects cost 189% more than initial estimates.

NB:- It is not a recent report.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Software cost estimation

Cost estimation can be defined as the approximate judgement of the costs for a project. Cost estimation will never be an exact science because there are too many variables involved in the calculation for a cost estimate, such as human, technical, environmental, and political.

Cost estimation is usually measured in terms of effort. The most common metric used is person months/man months (PM/MM). The effort is the amount of time for one person to work for a certain period of time (Usually in months or years).

It is important that the specific characteristics of the development environment are taking into account when comparing the effort of two or more projects because no two development environments are the same. A clear example of differences in development environments are the amount of time people work in different countries; the typical workweek in North America is 40 hours per week, while in Europe the typical workweek is 35 hours per week

In general, a Software cost estimation is

- An approximate judgment of the costs for a project
  - Many variables
- Often measured in terms of effort (i.e., person months/years)
- Different development environments will determine which variables are included in the cost value
  - **Management costs**
  - **Development costs**
  - **Training costs**
  - **Quality assurance costs**

Cost estimation during software life cycle

- Cost estimation should be done throughout the software life cycle to allow for refinement.
- Need effective monitoring and control of the software costs to verify and improve accuracy of estimates
  - At appropriate level of detail
  - Gathering data should not be difficult
- Success of a cost estimate method is not necessarily the accuracy of the initial estimates, but rather the rate at which estimates converge to the actual cost.

Who is the estimator?

- Someone responsible for the implementation
  - Can compare previous projects in organization to current project
  - Usually experienced
- Someone from outside the organization
  - Can provide unbiased estimate
  - Tend to use empirical methods of estimation
    - Difficulties:
      - Lack of confidence.
      - Lack of historical data to calibrate the model.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

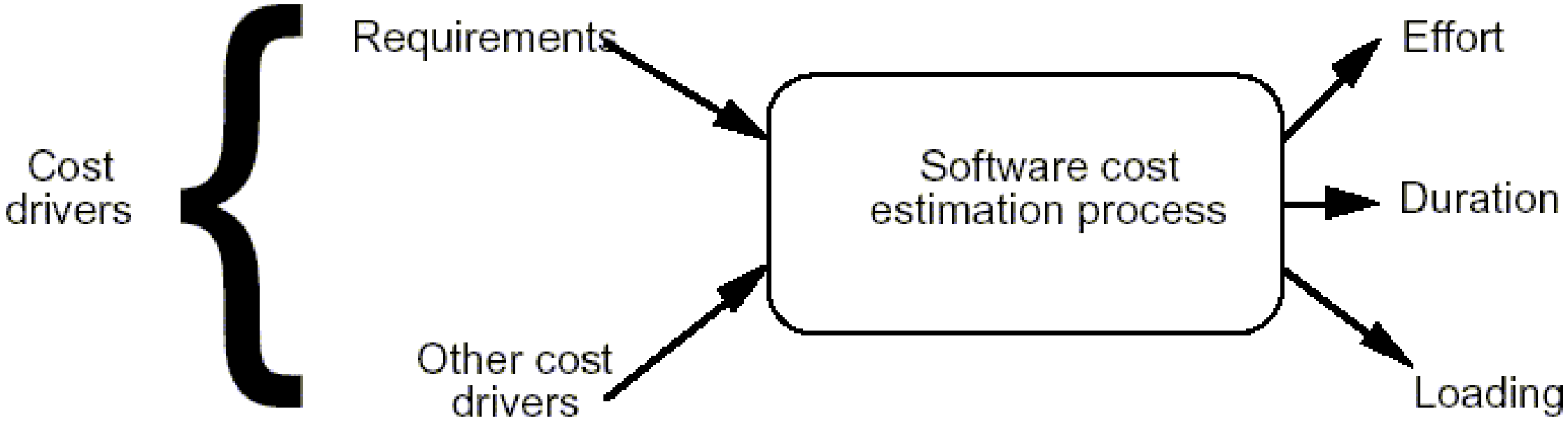
General steps and remarks

- Establish Plan
  - What data should we gather
  - Why are we gathering this data
  - What do we hope to accomplish
- Do cost estimation for initial requirements
  - Decomposition
- Use several methods
  - There is no perfect technique
- If get wide variances in methods, then should re-evaluate the information used to make estimates
- Do re-estimates during life cycle
- Make any required changes to development
- Do a final assessment of cost estimation at the end of the project

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Classical view of Cost estimation process

Definition: Software cost estimation process is a set of techniques and procedures that is used to derive the software cost estimate. There is usually a set of inputs to the process and then the process uses these inputs to generate or calculate a set of outputs.



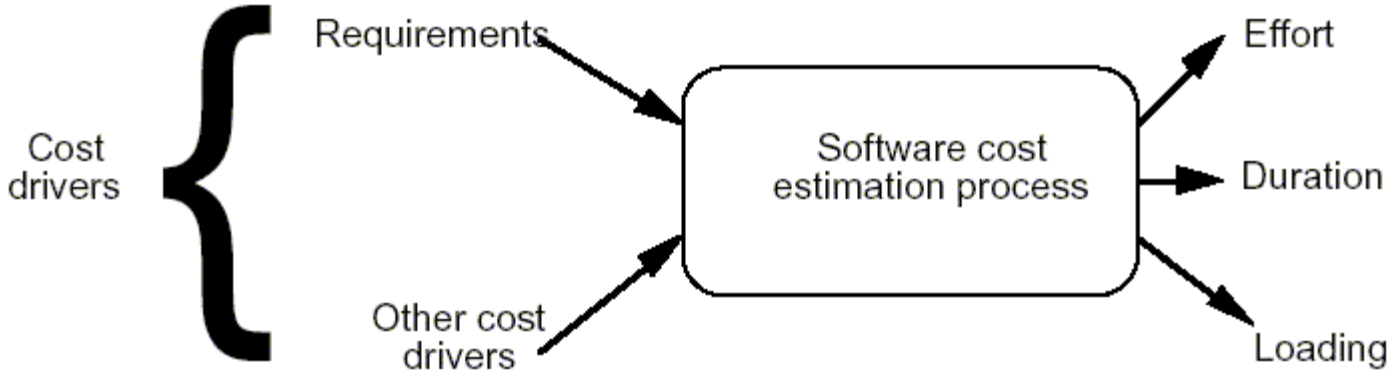
RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Classical view - Cost estimation process

In a classical view of the estimation process, it will generate three outputs - efforts, duration and loading. The following is a brief description of the outputs:

- **Manpower loading** - number of personnel (which also includes management personnel) that are allocated to the project as a function of time.
- **Project duration** - time that is needed to complete the project.
- **Effort** - amount of effort required to complete the project and is usually measured in units as man-months (MM) or person-months (PM).

.

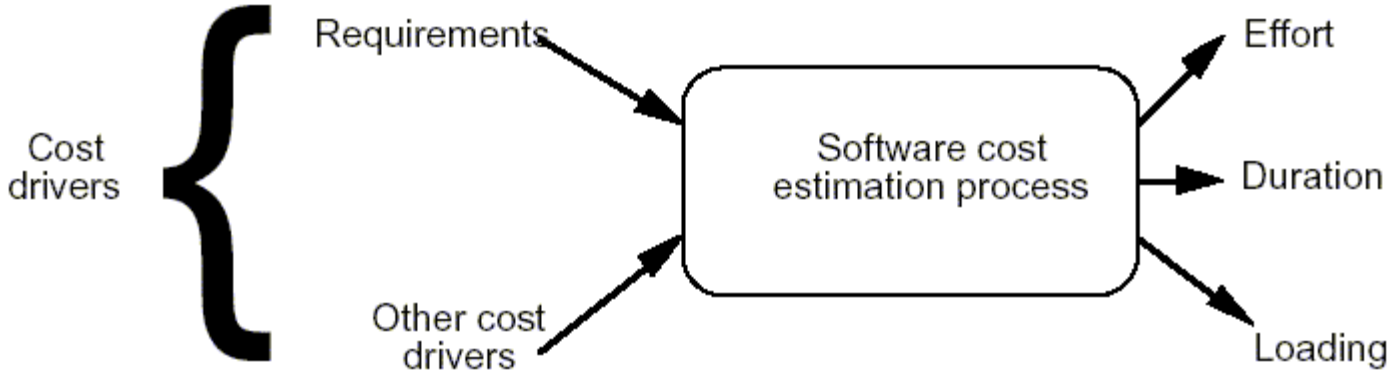


RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Classical view - Cost estimation process

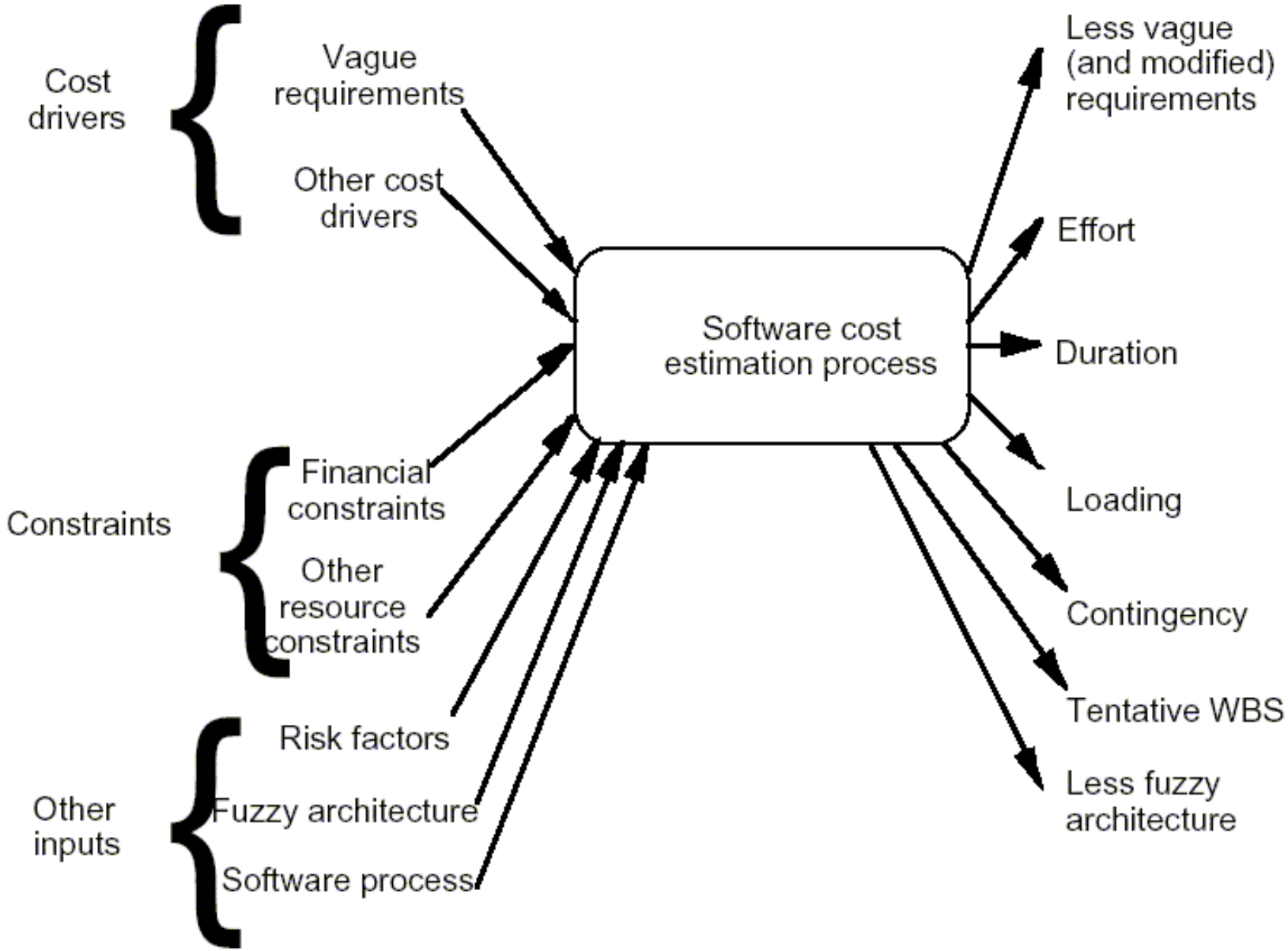
The outputs (loading, duration and effort) are usually computed as fixed number with or without tolerance in the classical view.

But in reality, the cost estimation process is more complex than what is shown in figure. Many of the data that are inputs to the process are modified or refined during the software cost estimation process.



RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Real/Actual view - Cost estimation process



RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Cost estimation accuracy

The cost estimation accuracy helps to determine how well or how accurate our estimation is when using a particular model or technique. We can assess the performance of the software estimation technique by 3 types of error:

- Absolute Error ($E_{pred} - E_{act}$)
- Percentage or Relative Error ($E_{pred} - E_{act}$) / E_{act}
- Mean Magnitude of Relative Error $\frac{1}{n} \cdot \sum_{i=1}^{i=n} \left(\frac{(|E_{pred} - E_{act}|)}{E_{act}} \right)_i$

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Cost estimation methods

1. Algorithmic (Parametric) model
2. Expert Judgment (Expertise Based)
3. Top – Down
4. Bottom – Up
5. Estimation by Analogy
6. Price to Win Estimation

Algorithmic (Parametric) model

- Use of mathematical equations to perform software estimation.
- Equations are based on theory or historical data.
- Use input such as SLOC, number of functions to perform and other cost drivers
- Accuracy of model can be improved by calibrating the model to the specific environment

Algorithmic (Parametric) model (cont....)

- Examples:
  - COCOMO (COnstructive COst MOdel)
    - Developed by Boehm in 1981
    - Became one of the most popular and most transparent cost model
    - Mathematical model based on the data from 63 historical software project
  - COCOMO II
    - Published in 1995
    - To address issue on non-sequential and rapid development process models, reengineering, reuse driven approaches, object oriented approach etc.
    - Has three sub-models – application composition, early design and post-architecture.

Algorithmic (Parametric) model (cont....)

- Advantages
  - Generate repeatable estimations
  - Easy to modify input data
  - Easy to refine and customize formulas
  - Objectively calibrated to experience

- Disadvantages
  - Unable to deal with exceptional conditions
  - Some experience and factors can not be quantified
  - Sometimes algorithms may be proprietary (owned by someone)

Expert judgement

- Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are discussed and compared. The estimation process iterates until an agreed estimate is reached.

- Examples
  - Delphi
    - Developed by Rand Corporation in 1940 where participants are involved in two assessment rounds.
  - Work Breakdown Structure (WBS)
    - A way of organizing project element into a hierarchy that simplifies the task of budget estimation and control.

Expert judgement

- Advantages
  - Useful in the absence of quantified, empirical data.
  - Can factor in differences between past project experiences and requirements of the proposed project
  - Can factor in impacts caused by new technologies, applications and languages.

- Disadvantages
  - Estimate is only as good expert's opinion
  - Hard to document the factors used by the experts.

Top - down

- Also called Macro Model
- Derived from the global properties of the product and then partitioned into various low level components.
- Example – Putnam model. (discussion comes later)

Top - down

- Advantages
  - Requires minimal project detail
  - Usually faster and easier to implement
  - Focus on system level activities

- Disadvantages
  - Tend to overlook low level components
  - No detailed basis.

Bottom - up

- Cost of each software components is estimated and then combine the results to arrive the total cost for the project
- The goal is to construct the estimate of the system from the knowledge accumulated about the small software components and their interactions
- An example – COCOMO's detailed model.

Bottom - up

- Advantages
  - More stable
  - More detailed
  - Allow each software group to hand an estimate
- Disadvantages
  - May overlook system level costs
  - More time consuming.

Estimation by Analogy

- Comparing the proposed project to previously completed similar project in the same application domain
- Actual data from the completed projects are extrapolated
- Can be used either at system or component level.

Estimation by Analogy

- Advantages
  - Based on actual project data
- Disadvantages
  - Impossible if no comparable project had been tackled in the past.
  - How well does the previous project represent this one.

Price to Win

- The software cost is estimated to be whatever the customer has available to spend on the project. It depends on the customer's budget and not on the software functionality.
- Advantages
  - Often rewarded with the contract.
- Disadvantages
  - Time and money run out before the job is done

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

COCOMO-1 or COCOMO 81

- The COnstructive COst MOdel was developed by Barry W. Boehm in the late 1970s and published in Boehm's 1981 book *Software Engineering Economics* as a model for estimating effort, cost, and schedule for software projects.
- It drew on a study of 63 projects at TRW Aerospace where Boehm was Director of Software Research and Technology.
- The study examined projects ranging in size from 2,000 to 1,00,000 lines of code, and programming languages ranging from assembly to PL/I.
- These projects were based on the waterfall model of software development which was the prevalent software development process in 1981.

COCOMO-1 or COCOMO 81

- COCOMO has three different models (each one increasing with detail and accuracy):
  - **Basic**, applied early in a project
  - **Intermediate**, applied after requirements are specified.
  - **Advanced**, applied after design is complete

- COCOMO has three different modes:
  - **Organic** – "relatively small software teams develop software in a highly familiar, in-house environment"
  - **Embedded** – operate within tight constraints, product is strongly tied to "complex of hardware, software, regulations, and operational procedures"
  - **Semi-detached** – intermediate stage somewhere between organic and embedded.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

COCOMO 81

- COCOMO uses two equations to calculate effort in man months (MM) and the number on months estimated for project (TDEV)
- MM is based on the number of thousand lines of delivered instructions/source (KLoC)
- **$MM = a(KLoC)^b * EAF$**
- **$TDEV = c(MM)^d$**
- EAF is the **Effort Adjustment Factor** derived from the Cost Drivers, EAF for the basic model is 1
- The values for a, b, c, and d differ depending on which mode you are using

Mode	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

COCOMO Intermediate

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

Intermediate COCOMO

Each of the 15 attributes receives a rating on a six-point scale that ranges from "very low" to "extra high" (in importance or value).

An effort multiplier from the table below applies to the rating. The product of all effort multipliers results in an *effort adjustment factor (EAF)*.

Typical values for EAF range from 0.9 to 1.4.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Intermediate COCOMO

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

Intermediate COCOMO

The Intermediate COCOMO formula now takes the form:

$E=a_i(KLoC)(b_i)(EAF)$

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

Software project	a_i	b_i
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

COCOMO II

- Main objectives of COCOMO II:
  - To develop a software cost and schedule estimation model tuned to the life cycle practices of the 1990's and 2000's
  - To develop software cost database and tool support capabilities for continuous model improvement.

Has three different models:

- The Application Composition Model
  - Used during early stages of software engineering, when prototyping user interfaces, consideration of software and system interaction.
- The Early Design Model
  - This model can get rough estimates before the entire architecture has been decided
- The Post-Architecture Model
  - Most detailed model, used after overall architecture has been decided on.

COCOMO II

COCOMO II uses object points – Calculated from (1) Screens, (2) Reports and (3) Components likely to be required to build the application.

Each object point is classified into one of three complexity levels: simple, medium or difficult.

Once complexity level is identified, the screens, reports and components are weighed according to the following table.

COCOMO II

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

COCOMO II

The object point count is calculated by multiplying the original number of object instances by the complexity weight factor.

When a reusable component is being used, the object count is adjusted by taking the following formula:

NOP = (object points)*[(100-%reuse)/100]

Where NOP is defined as new object points.

RSET
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

COCOMO II

To derive an effort based on the computed NOP value, a “productivity rate” must be achieved. The following figure presents the productivity rate (PROD) for different levels of developer experience and development environment maturity.

Developer’s experience/capability	Very low	Low	Nominal	High	Very High
Environment maturity/capability	Very low	Low	Nominal	High	Very High
PROD	4	7	13	25	50

PROD = NOP / person-month

So, person-month = NOP/PROD