Noida Institute of Engineering and Technology, Greater Noida

Project Maintenance and Management Concepts

Unit: 5

Subject Name: Software Engineering
(ACSEH0603)

Course Details
B.Tech.(CSE R)
6th Sem

Categories Of Software Maintenance (CO5)

There are four types of software maintenance:

- **Corrective maintenance:** This refer to modifications initiated by defects in the software.
- **Adaptive maintenance:** It includes modifying the software to match changes in the ever-changing environment.
- **Perfective maintenance:** It means improving processing efficiency or performance or restructuring the software to improve changeability. This may include enhancement of existing system functionality, improvement in computational efficiency etc.
- **Preventive maintenance:** It is the process by which we prevent our system from being obsolete. It involves the concept of reengineering & reverse engineering in which an old system with an old technology is re-engineered using new technology. This maintenance prevents the system from dying out.

Maintenance Types

There are long term effects of corrective, adaptive and perfective changes. This leads to increase in the complexity of the software, which reflect deteriorating structure. The work is required to be done to maintain it or to reduce it, if possible. This work may be named as preventive maintenance.

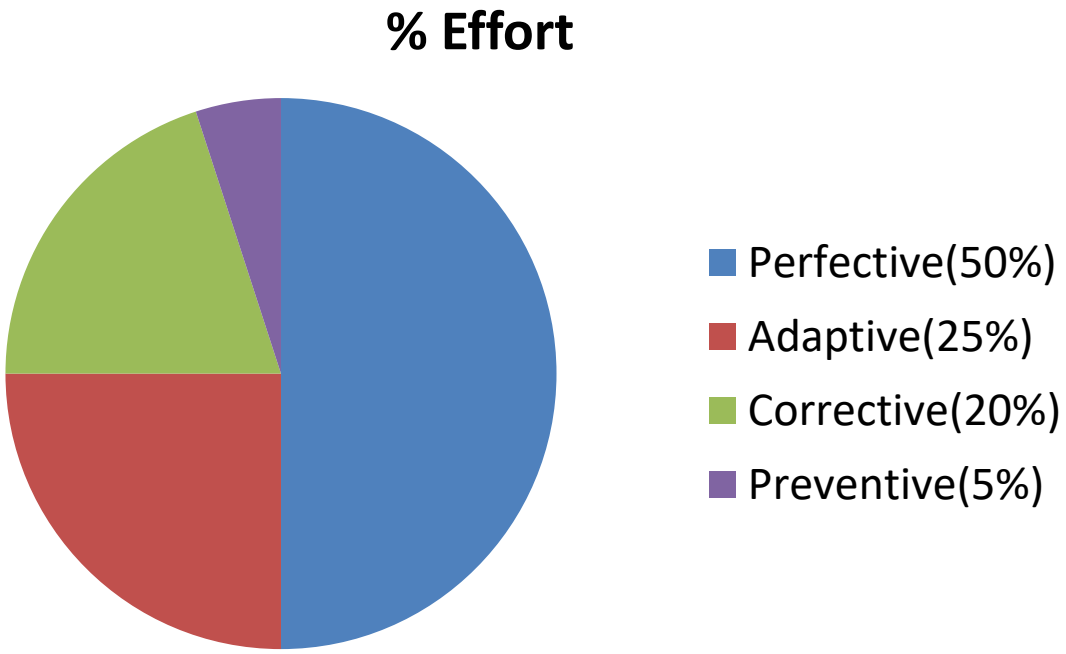


Fig: Distribution of maintenance effort

Problems during Software Maintenance

Problems During Maintenance

- The most important problem during maintenance is that before correcting or modifying a program, the programmer must first understand it. Then, the programmer must understand the impact of the intended change.
- Often the program is written by another person or group of persons working over the years in isolation from each other.
- Often the program is changed by person who did not understand it clearly, resulting in a deterioration of the program's original organization. Program listing, even those that are well organized, are not structured to support reading or comprehension.

Maintenance Process

Program Understanding

– The first phase consists of analyzing the program in order to understand.

Generating Particular Maintenance Proposal

– The second phase consists of generating a particular maintenance proposal to accomplish the implementation of the maintenance objective.

Ripple Effect

– The third phase consists of accounting for all of the ripple effect as a consequence of program modifications.

Maintenance Process

Modified Program Testing

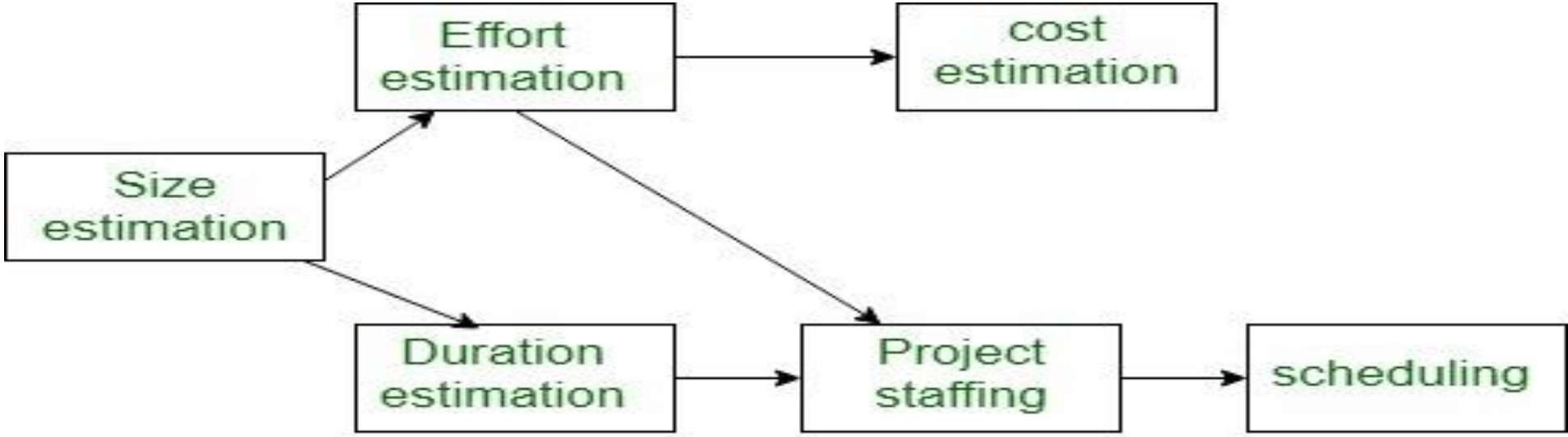
- The fourth phase consists of testing the modified program to ensure that the modified program has at least the same reliability level as before.

Maintainability

- Each of these four phases and their associated software quality attributes are critical to the maintenance process. All of these factors must be combined to form maintainability.

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 in which vital project coming up with activities is also undertaken. It may be simply discovered that size estimation is 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.



Sliding Window Planning:

Project designing needs utmost care and a spotlight since commitment to unrealistic time and resource estimates ends in schedule slippage. Schedule delays will cause client discontent and adversely affect team morale. It will even cause project failure. However, project designing could be a difficult activity. particularly for giant comes, it's pretty much troublesome to create correct plans. A region of this issue is thanks to the fact that the correct parameters, the scope of the project, project workers, etc. might be amended throughout the project.

So as to beat this drawback, generally project managers undertake project design little by little. Designing a project over a variety of stages protects managers from creating huge commitments too early. This method of staggered designing is thought of as window designing. Within the window technique, beginning with the associate initial setup, the project is planned additional accurately in sequential development stages. At the beginning of a project, project managers have incomplete information concerning the main points of the project. Their info base step by step improves because the project progresses through completely different phases.

Software Measurement and Metrics

Software Measurement: A measurement is a manifestation of the size, quantity, amount, or dimension of a particular attribute of a product or process. Software measurement is a titrate impute of a characteristic of a software product or the software process.

Software Measurement Principles

Formulation: The derivation of software measures and metrics appropriate for the representation of the software that is being considered.

Collection: The mechanism used to accumulate data required to derive the formulated metrics.

Analysis: The computation of metrics and the application of mathematical tools.

Interpretation: The evaluation of metrics results in insight into the quality of the representation.

Feedback: Recommendation derived from the interpretation of product metrics transmitted to the software team.

Need for Software Measurement

Software is measured to:

Create the quality of the current product or process.

Anticipate future qualities of the product or process.

Enhance the quality of a product or process.

Regulate the state of the project concerning budget and schedule.

Enable data-driven decision-making in project planning and control.

Identify bottlenecks and areas for improvement to drive process improvement activities.

Ensure that industry standards and regulations are followed.

Give software products and processes a quantitative basis for evaluation.

Enable the ongoing improvement of software development practices.

Classification of Software Measurement

There are 2 types of software measurement:

Direct Measurement: In direct measurement, the product, process, or thing is measured directly using a standard scale.

Indirect Measurement: In indirect measurement, the quantity or quality to be measured is measured using related parameters i.e. by use of reference.

Software Metrics

A metric is a measurement of the level at which any impute belongs to a system product or process.

Software metrics are a quantifiable or countable assessment of the attributes of a software product. There are 4 functions related to software metrics:

Planning

Organizing

Controlling

Improving

Characteristics of software Metrics

Quantitative: Metrics must possess a quantitative nature. It means metrics can be expressed in numerical values.

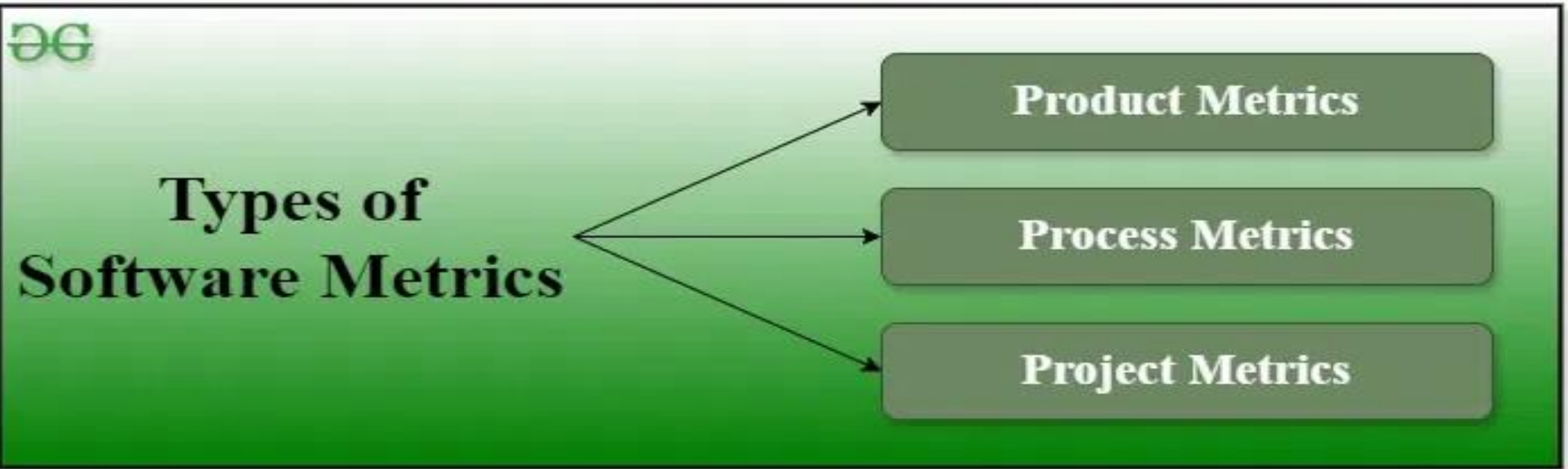
Understandable: Metric computation should be easily understood, and the method of computing metrics should be clearly defined.

Applicability: Metrics should be applicable in the initial phases of the development of the software.

Repeatable: When measured repeatedly, the metric values should be the same and consistent.

Economical: The computation of metrics should be economical.

Language Independent: Metrics should not depend on any programming language.



Characteristics of software Metrics

1. **Product Metrics:** Product metrics are used to evaluate the state of the product, tracing risks and undercover prospective problem areas. The ability of the team to control quality is evaluated. Examples include lines of code, cyclomatic complexity, code coverage, defect density, and code maintainability index.
2. **Process Metrics:** Process metrics pay particular attention to enhancing the long-term process of the team or organization. These metrics are used to optimize the development process and maintenance activities of software. Examples include effort variance, schedule variance, defect injection rate, and lead time.
3. **Project Metrics:** The project metrics describes the characteristic and execution of a project. Examples include effort estimation accuracy, schedule deviation, cost variance, and productivity. Usually measures-

   3. Number of software developer

   4. Staffing patterns over the life cycle of software

   5. Cost and schedule,Productivity

Characteristics of software Metrics

Advantages of Software Metrics

Reduction in cost or budget.

It helps to identify the particular area for improvising.

It helps to increase the product quality.

Managing the workloads and teams.

Reduction in overall time to produce the product,.

It helps to determine the complexity of the code and to test the code with resources.

It helps in providing effective planning, controlling and managing of the entire product.

Disadvantages of Software Metrics

It is expensive and difficult to implement the metrics in some cases.

Performance of the entire team or an individual from the team can't be determined. Only the performance of the product is determined.

Sometimes the quality of the product is not met with the expectation.

It leads to measure the unwanted data which is wastage of time.

Measuring the incorrect data leads to make wrong decision making.

Halstead's Software Metrics

Halstead's Software metrics are a set of measures proposed by Maurice Halstead to evaluate the complexity of a software program. These metrics are based on the number of distinct operators and operands in the program and are used to estimate the effort required to develop and maintain the program.

Field of Halstead Metrics

Program length (N):

This is the total number of operator and operand occurrences in the program.

Vocabulary size (n):

This is the total number of distinct operators and operands in the program.

Program volume (V):

This is the product of program length (N) and the logarithm of vocabulary size (n),

i.e., V = N*log2(n)

Program level (L):

This is the ratio of the number of operator occurrences to the number of operand occurrences in the program,

i.e., L = n1/n2

Halstead's Software Metrics

Program difficulty (D):

This is the ratio of the number of unique operators to the total number of operators in the program,

i.e., D = (n1/2) * (N2/n2)

Program effort (E):

This is the product of program volume (V) and program difficulty

Token Count

Overall, Halstead's software metrics can be a useful tool for software developers and project managers to estimate the effort required to develop and maintain software programs.

n1 = Number of distinct operators.

n2 = Number of distinct operands.

N1 = Total number of occurrences of operators.

N2 = Total number of occurrences of operands.

Halstead's Software Metrics

Halstead Metrics

Halstead metrics are:

Halstead Program Length:

The total number of operator occurrences and the total number of operand occurrences.

N = N1 + N2

And estimated program length is,

N^ = n1log2n1 + n2log2n2

NJ = log2(n1!) + log2(n2!)

NB = n1 * log2n2 + n2 * log2n1

NC = n1 * sqrt(n1) + n2 * sqrt(n2)

NS = (n * log2n) / 2

Halstead's Software Metrics

Halstead Vocabulary:

The total number of unique operators and unique operand occurrences.

$$n = n1 + n2$$

Program Volume:

$$V = Size * (\log_2 vocabulary) = N * \log_2(n)$$

$$error = Volume / 3000$$

Potential Minimum Volume:

$$V^* = (2 + n2^*) * \log_2(2 + n2^*)$$

Program Level:

$$L = V^* / V$$

$$L\hat{} = 2 * (n2) / (n1)(N2)$$

Program Difficulty:

$$D = (n1 / 2) * (N2 / n2)$$

$$D = 1 / L$$

Halstead's Software Metrics

Programming Effort: Measures the amount of mental activity needed to translate the existing algorithm into implementation in the specified program language.

E = V / L = D * V = Difficulty * Volume

Language Level:

L' = V / D / D

lambda = L * V* = L2 * V

Intelligence Content:

Determines the amount of intelligence presented (stated) in the program This parameter provides a measurement of program complexity, independently of the programming language in which it was implemented.

I = V / D

Programming Time:

Shows time (in minutes) needed to translate the existing algorithm into implementation in the specified program language.

T = E / (f * S)

Halstead's Software Metrics

List out the operators and operands and also calculate the values of software science measures like

```
int sort (int x[ ], int n)
{
    int i, j, save, im1;
    /*This function sorts array x in ascending order */
    If (n< 2) return 1;
    for (i=2; i< =n; i++)
    {
        im1=i-1;
        for (j=1; j< =im1; j++)
            if (x[i] < x[j])
            {
                Save = x[i];
                x[i] = x[j];
                x[j] = save;
            } }return 0;
```

Halstead's Software Metrics

int	4	sort	1
()	5	x	7
,	4	n	3
[]	7	i	8
if	2	j	7
<	2	save	3
;	11	im1	3
for	2	2	2
=	6	1	3
–	1	0	1
<=	2	–	–
++	2	–	–
return	2	–	–

Halstead's Software Metrics

Therefore,

N = 91

n = 24

V = 417.23 bits

N^ = 86.51

n2* = 3 (x: array holding integer to be sorted.
This is used both as input and output)

V* = 11.6

L = 0.027

D = 37.03

L^ = 0.038

T = 610 seconds

Halstead's Software Metrics

- *Halstead Length: N = N1+N2*
- *Halstead Calculated Length (Nx): Nx = N1 * log2(n1) + n2* log2(n2)*
- *Halstead Volume(V): V = N*log2(n)*
- *Halstead Effort(E): E=V*D*

Halstead's Software Metrics

HALSTEAD'S SOFTWARE METRICS THE POINTS FOR OPERATOR AND OPERANDS Comments are not considered. The identifier and function declarations are not considered. All the variables and constants are considered operands. Global variables used in different modules of the same program are counted as multiple occurrences of the same variable. Local variables with the same name in different functions are counted as unique operands. Functions calls are considered as operators. All looping statements e.g., do, while, while, for, all control statements are considered as operators. In control construct switch as well as all the case statements are considered as operators. The reserve words like return, default, continue, break, size of, etc., are considered as operators. All the brackets, commas, and terminators are considered as operators. GOTO is counted as an operator and the label is counted as an operand. The unary and binary occurrence of "+" and "-" are dealt separately. Similarly, "*" are dealt separately. In the array variables such as "array-name" and "index" are considered as operands and ANGLE BRACKETS is considered as operator. In the structure variables such as "struct-name, member-name", struct-name, member-name are taken as operands. Some names of member elements in different structure variables are counted as unique operands. All the hash directives are ignored.

Estimation Software Metrics

① Estimation model →

Calculation of cost to be procured during Software development so that we can decide total profile by setting the margin levels. major estimation can be divided into four types

→ Post estimation
→ Base estimation
→ Decomposition
→ Empirical model

Post estimation → In case of friendly party and known technology we do not estimate either the time or the cost, b/z we know that cost we support as in every situation.

e.g shared auto from NIET TO Parichowk. here we are not calculating the cost prior to the software development lets develop the s/w first afterwards we calculate the cost.

→ Base estimation → In base estimation we predict the cost and time of the entire project based on the experience which we ha gained from the previous projects.

Estimation Software Metrics

Though the project size is not very big we know the how long it will be going and what would be cost to be come by the end of project s/w development b/z we have handled so many projects like this. e.g if we want to go to Noida we know the fare of Auto for Noida from greater Noida for safer side we decided the fare prior final ride/ confirmation of ride. Tenda given to whole project. White wash of whole home with time and money decided priorly.

→ Decomposition Based Estimation →

It is used for large projects where decomposition of the problem into smaller problems is done usually it is done on two bases.

Direct Estimation → Size oriented (KLOC) metrices

Indirect Estimation → Function oriented (FP) metrices

e.g taxi cost per day bases, particular amount during acc to km per 9–5 after 5 that much amount extra charges for night fare

Estimation Software Metrics

Plus toll tax extra.

Size oriented metrics (KLOC) auto on metr basis. 30,000 KLOC code we charge acc to KLOC. lines of code. some time it is not possible to judge the cost of s/w

Indirect Estimation — Function point → Here in we know the difficulty level acc to that level we charge the s/w e.g we cannot have the same fare price for maruti 800 / Alto or BMW / Audi.

Direct Estimation →

Que Consider a banking application which requires 4 modules as follows -

Database = 5400 LOC
User Interface = 8500 LOC
Server = 7300 LOC
Client = 5500 LOC

Total = 5400 + 8500 + 7300 + 5500
= 26,700

Find the total expenditure of company if productivity of the person is 840 LOC per month

Estimation Software Metrics

Consider the salary of each developer is 1500 dollar per month.

1 developer writes 870 Loc in one month

870 ——— 1 person month

1 ——— $\frac{1}{870}$

26,700 $\frac{1}{870} \times 26{,}700 = 30.689$ PM

= Effort

Cost $30.689 \times 1500 =$ $46,034

Duration = $\frac{\text{Effort}}{\text{Team size}}$

$= \frac{30.689}{4} = \boxed{7.67}$ approx 8 months

or if we want to complete our work into 3 months then

Team size = $\frac{\text{Effort}}{\text{Duration}} = \frac{30.689}{3}$

$= \boxed{10.227}$

upper bound like we require minimum 11 persons to complete it into 3 months.

Estimation Software Metrics

Effort unit is person month. ⑤

Effort = Size / productivity [340 LoC] productivity

productivity = Size / Effort

Size = Effort x Productivity

Cost = Effort x Pay

Duration = Effort / Team Size

Team size = Effort / Duration

Effort = Duration x Team Size

Que → Consider a application which contain 3 modules m1, m2, m3 having following size

m1 — 14.4 KLoC

m2 — 21.5 KLoC

m3 — 8.4 KLoC

If the cost of writing 1 KLoC is ₹ 50 dollars Find the cost of application, if the productivity of the developer is 4 KLoC per month. Find the effort if the project is required to

Estimation Software Metrics

be completed in month then predict the team size.

$$\text{Effort} = \frac{\text{Size}}{\text{Productivity}} = \frac{44.3 \text{ KLOC}}{4 \text{ KLOC}}$$

$$= 11.078 \text{ Person month}$$

If one person write then 11 months required to complete the project or 12 persons writes the code only one month is sufficient to complete the code.

Indirect measurement

measurement	simple	Average	complex
i/p	3	4	6
o/p	4	5	7
Inquiry	3	4	6
Files	7	10	15
External interface	5	7	10

Given parameters

i/p - 29
o/p - 27
Inquiry - 16
Files - 5

29 x 3 = 87
27 x 4 = 108
16 x 3 = 48
5 x 7 = 35
5 x 5 = 25

Estimation Software Metrics

Now develop tells his productivity in terms of function points. Let us assume fp is 20 (1)

$$\text{Effort} = \frac{\text{Size}}{\text{Productivity}} = \frac{303}{20} = 15.15 \text{ PM}$$

If one person works then 15 months required to complete the s/w or 16 persons can complete it in one month only.

In Adjusteable case

FP = total_count * EAF (error adjusteable factor)

$$EAF = 0.65 + 0.01 * \sum_{i=1}^{i=14} f_i$$

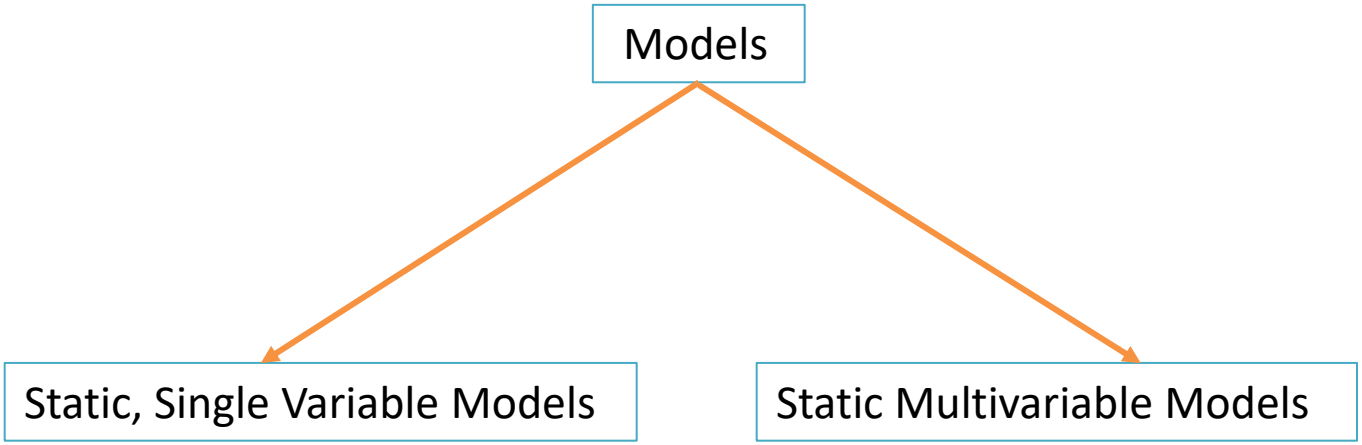
Toteel ount = 303

EAF = 1.17

FP = 303 x 1.17 = 354.51 fp.

Adjusteal count

Cost Estimation (CO5)



Static, Single Variable Models

Effort (E in Person-months), documentation (DOC, in number of pages) and duration (D, in months) are calculated from the number of lines of code (L, in thousands of lines) used as a predictor. Methods using this model use an equation to estimate the desired values such as cost, time, effort, etc. They all depend on the same variable used as predictor (say, size)

$$c = \mathrm{a}\mathrm{L}^{b}$$

C is the cost, L is the size and a,b are constants

$$Effort(E) = 1.4L^{0.93}$$

$$DOC = 30.4L^{0.90}$$

$$Duration(D) = 4.6L^{0.26}$$

Static, Multivariable Models

These models are often based on equation (i), they actually depend on several variables representing various aspects of the software development environment, for example method used, user participation, customer oriented changes, memory constraints, etc.

$$E = 5.2\ L^{0.91}$$

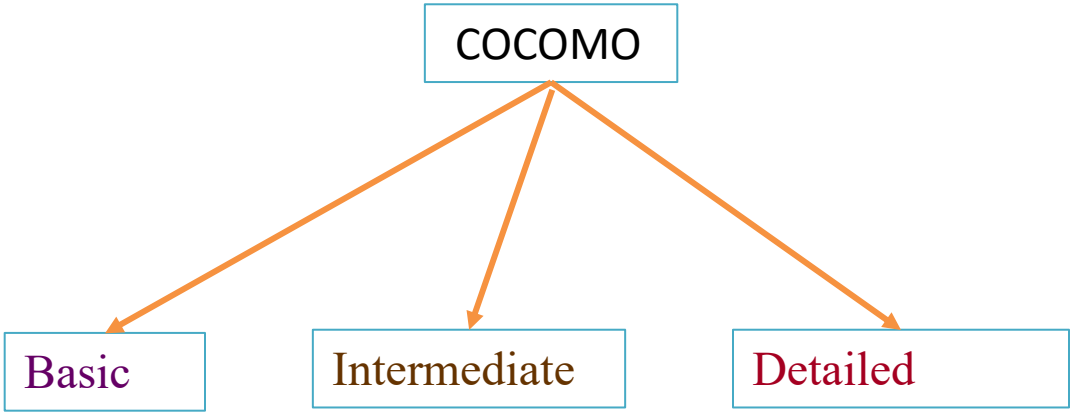
$$D = 4.1\ L^{0.36}$$

The productivity index uses 29 variables which are found to be highly correlated to productivity as follows:

$$I = \sum_{i=1}^{29} W_i X_i$$

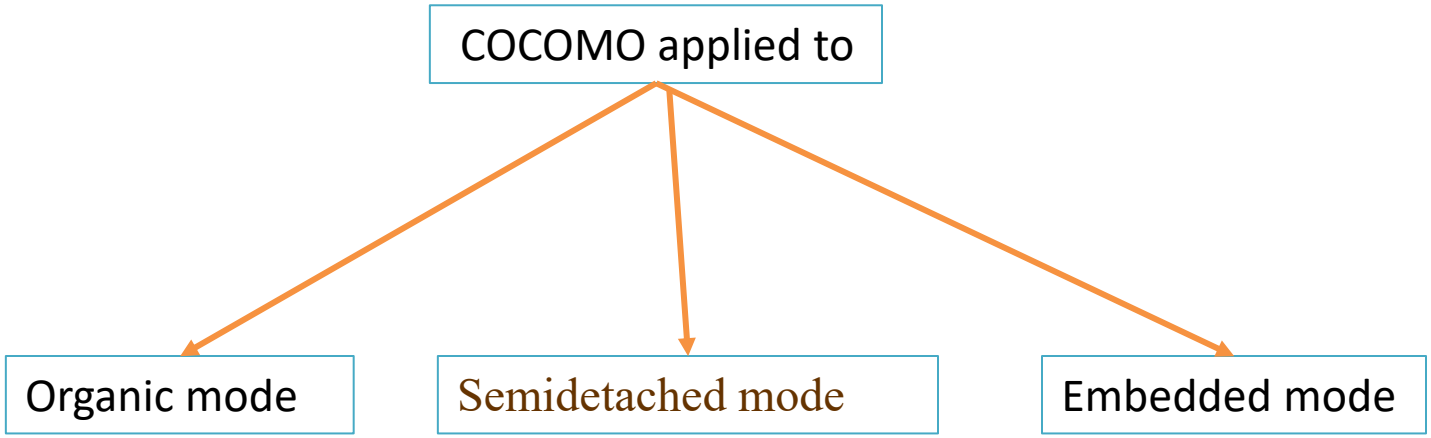
COCOMO (CO5)

Constructive Cost model



COCOMO (CO5)

Constructive Cost model



Comparison of three COCOMO modes

Mode	*Project size*	*Nature of Project*	*Innovation*	*Deadline of the project*	*Development Environment*
Organic	Typically 2-50 KLOC	Small size project, experienced developers in the familiar environment. For example, pay roll, inventory projects etc.	Little	Not tight	Familiar & In house
Semi detached	Typically 50-300 KLOC	Medium size project, Medium size team, Average previous experience on similar project. For example: Utility systems like compilers, database systems, editors etc.	Medium	Medium	Medium
Embedded	Typically over 300 KLOC	Large project, Real time systems, Complex interfaces, Very little previous experience. For example: ATMs, Air Traffic Control etc.	Significant	Tight	Complex Hardware/ customer Interfaces required

Basic Model

Basic COCOMO model takes the form

$$E = a_b (KLOC)^{b_b}$$

$$D = c\ (E)$$

where E is effort applied in Person-Months, and D is the development time in months. The coefficients a_b, b_b, c_b and d_b are given in table.

Basic COCOMO coefficients

Software Project	a_b	b_b	c_b	d_b
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

Basic COCOMO coefficients

When effort and development time are known, the average staff size to complete the project may be calculated as:

$$\text{Average staff size } (SS) = \frac{E}{D} \; Persons$$

When project size is known, the productivity level may be calculated as:

$$\text{Productivity } (P) = \frac{KLOC}{E} \; KLOC / PM$$

Intermediate Model

Cost drivers

Product Attributes

- Required s/w reliability
- Size of application database
- Complexity of the product

Hardware Attributes

- Run time performance constraints
- Memory constraints
- Virtual machine volatility
- Turnaround time

Intermediate Model

Personal Attributes

- Analyst capability
- Programmer capability
- Application experience
- Virtual m/c experience
- Programming language experience

Project Attributes

- Modern programming practices
- Use of software tools
- Required development Schedule

Multipliers of different cost drivers

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	--

Multipliers of different cost drivers

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	--

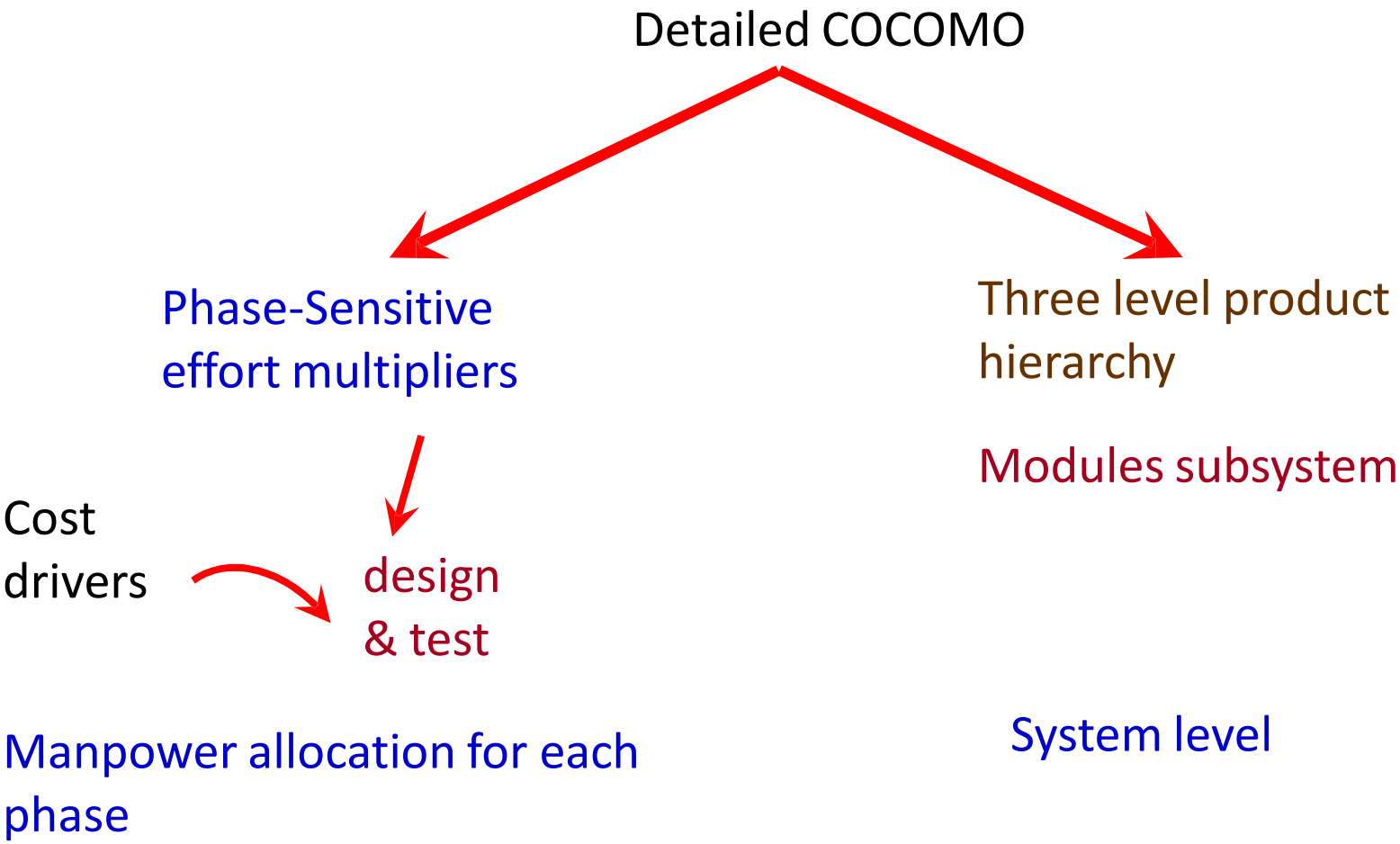
Intermediate COCOMO equations

$$E = a_i (KLOC)^{b_i} * EAF$$

$$D = c (E)^{d_i}$$

Project	a_i	b_i	c_i	d_i
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Detailed COCOMO Model



Detailed COCOMO Model

Development Phase

Plan / Requirements

EFFORT DEVELOPMENT TIME	:	6% to 8%
% depend on mode & size	:	10% to 40%

Detailed COCOMO Model

Design

Effort	:	16% to 18%
Time	:	19% to 38%

Programming

Effort	:	48% to 68%
Time	:	24% to 64%

Integration & Test

Effort	:	16% to 34%
Time	:	18% to 34%

Detailed COCOMO Model

Principle of the effort estimate

Size equivalent

As the software might be partly developed from software already existing (that is, re-usable code), a full development is not always required. In such cases, the parts of design document (DD%), code (C%) and integration (I%) to be modified are estimated. Then, an adjustment factor, A, is calculated by means of the following equation.

$$A = 0.4\ DD + 0.3\ C + 0.3\ I$$

The size equivalent is obtained by

$$S\ (equivalent) = (S \times A) / 100$$

$$E_p = \mu_p E$$

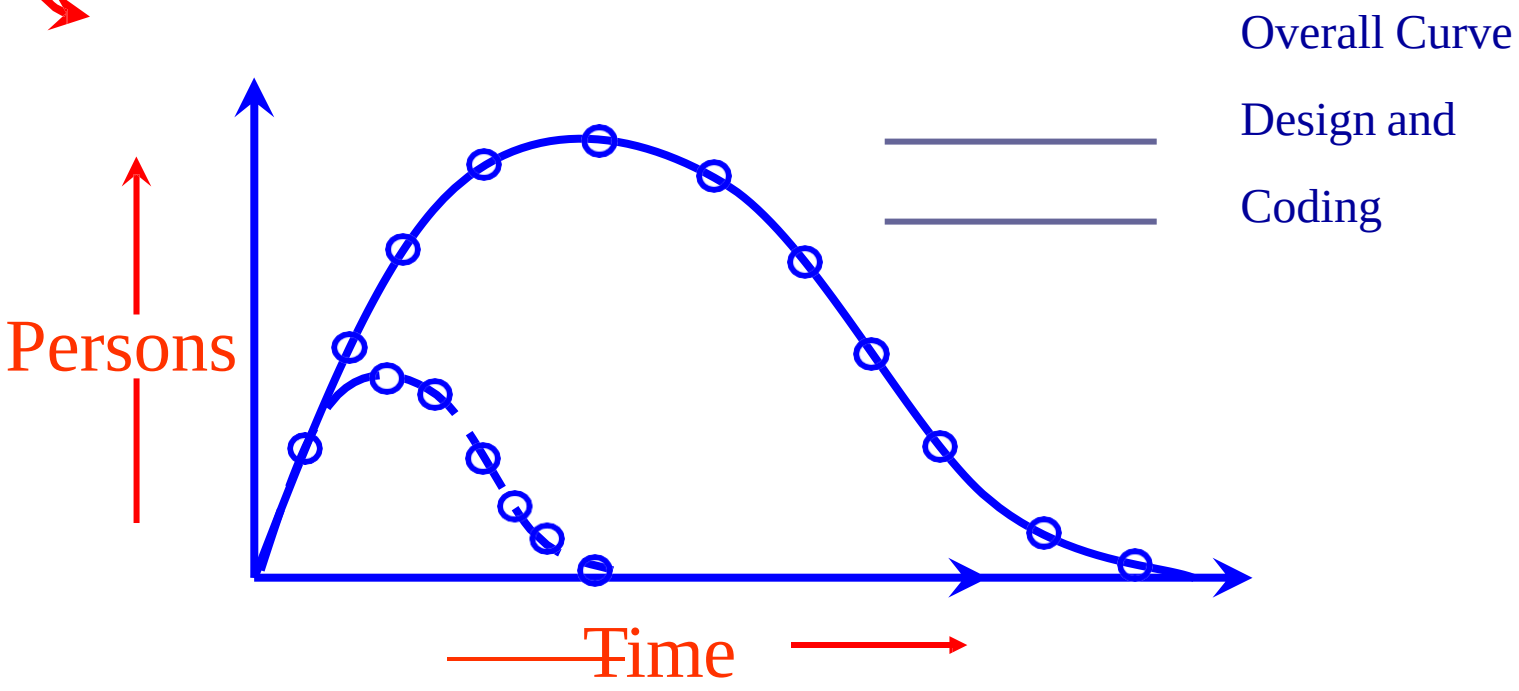
$$D_p = \tau_p D$$

Resource Allocation Model (CO5)

Norden of IBM

Rayleigh curve

Model for a range of hardware development projects.



The Rayleigh manpower loading curve

Resource Allocation Model

The Norden / Rayleigh Curve

The curve is modeled by differential equation

$$m(t) = \frac{dy}{dt} = 2kate^{-at^2}$$

dt = manpower utilization rate per unit time

a = parameter that affects the shape of the curve

K = area under curve in the interval $[0, \infty]$

t = elapsed time

Resource Allocation Model

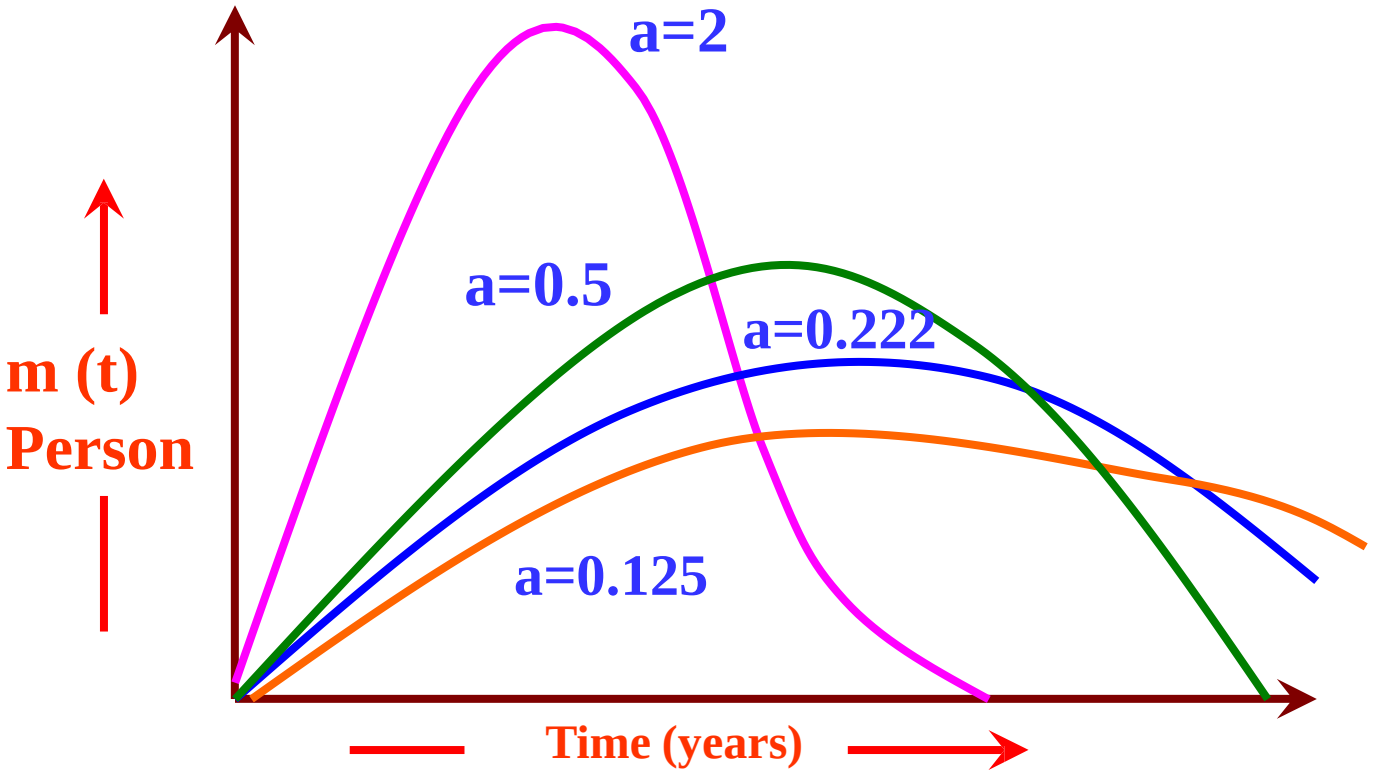


Fig: Influence of parameter ‘a’ on the manpower distribution

COCOMO Model

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e., the number of Lines of Code. The Cocomo Model is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models.

1.Organic

A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.

2. **Semi-detached**

A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environments lie in between organic and embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered Semi-Detached types.

3. Embedded

A software project requiring the highest level of complexity, creativity, and experience requirement falls under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

Detailed Structure of COCOMO Model

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed Cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

Planning and requirements

System design

Detailed design

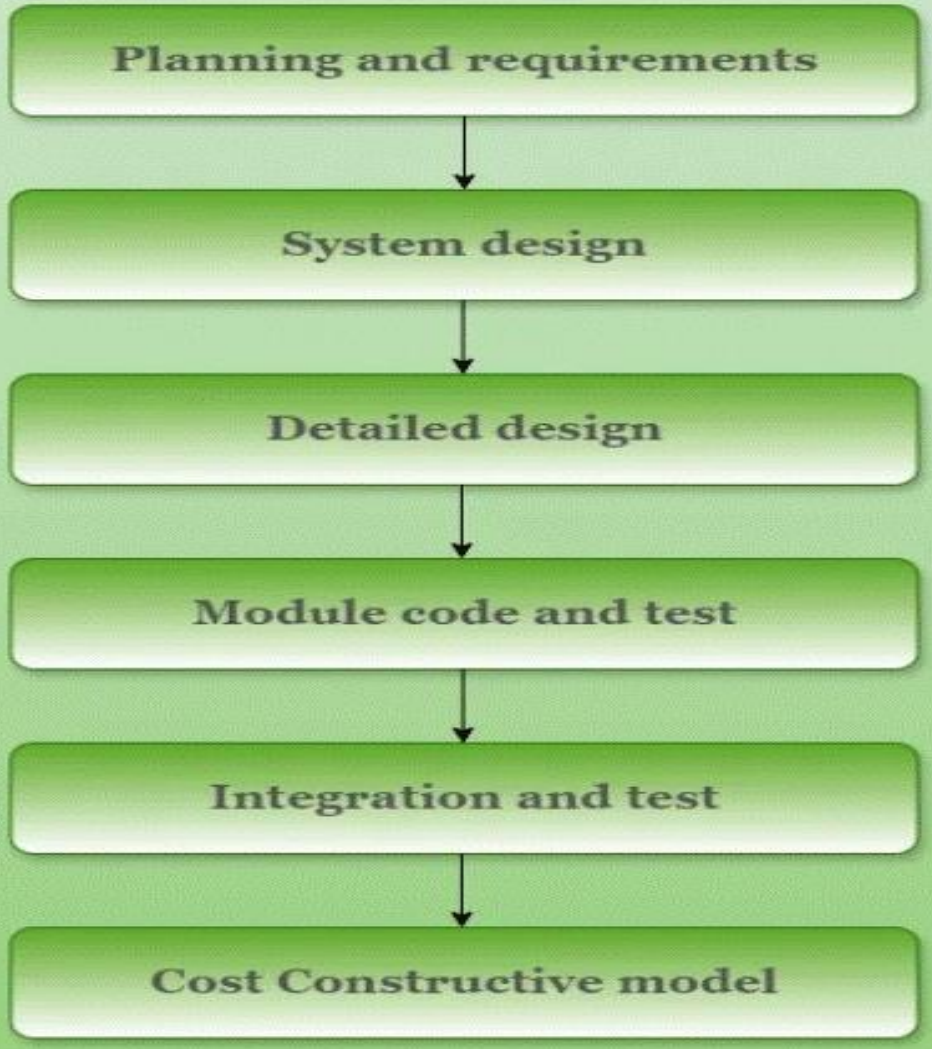
Module code and test

Integration and test

Cost Constructive model

Detailed Structure of COCOMO Model

Six phases of COCOMO Model



Detailed Structure of COCOMO Model

- **Importance of the COCOMO Model**

1. **Cost Estimation:** To help with resource planning and project budgeting, COCOMO offers a methodical approach to software development cost estimation.
2. **Resource Management:** By taking team experience, project size, and complexity into account, the model helps with efficient resource allocation.
3. **Project Planning**: COCOMO assists in developing practical project plans that include attainable objectives, due dates, and benchmarks.
4. **Risk management**: Early in the development process, COCOMO assists in identifying and mitigating potential hazards by including risk elements.
5. **Support for Decisions**: During project planning, the model provides a quantitative foundation for choices about scope, priorities, and resource allocation.
6. **Benchmarking**: To compare and assess various software development projects to industry standards, COCOMO offers a benchmark.
7. **Resource Optimization:** The model helps to maximize the use of resources, which raises productivity and lowers costs.

Detailed Structure of COCOMO Model

- Types of COCOMO Model
- 1. Basic Model
- E = a(KLOC)^b
- Time = c(Effort)^d,Person required = Effort/ time

Software Projects	a	b	c	d
Organic/ Application	2.4	1.05	2.5	0.38
Semi-Detached/ Utility	3.0	1.12	2.5	0.35
Embedded/System	3.6	1.20	2.5	0.32

NIET Greater Noida
Estd 2001 GET FUTURE READY
AN AUTONOMOUS INSTITUTE

Detailed Structure of COCOMO Model

In order to classify a product into any of the three proposed classes, we take into consideration the characteristics of the product as well as those of the development team.

→ Data processing and scientific Program are considered to be applicable Programs. compilers linkers etc are utility Programs. operating systems, time real system Programs are be system Programs.

Relative level of the Product complexicity for three categories of product are in the ratio 1:3:9 for application, utility system Programs relatively.

Organic — Relatively small group works in a familia enviorment to develop well understood application Program, little innova tion is required, constraints and deadlines are few and development enviornment is steeble.

Detailed Structure of COCOMO Model

(2) Semidetached — Project team consists of a mixture of experienced and inexperienced staff. It is a medium size, some innovation is required. Constraints and deadlines are moderate and development enviornment is somewhat fixed.

Embedded :— S/w is strongly coupled to complex h/w such as air traffic control, ATM's weapon systems. The Project team is large a great deal of innovation is required. Constraints and deadlines are tight and the development environment consists of many complex interfaces, including those with h/w and with customer

Basic COCOMO

$$D_E = a_b (KLOC)^{bb}$$

$$D_D = C_b (D_E)^{db}$$

$$\text{Team Size} = D_E / D_D$$

Detailed Structure of COCOMO Model

Sl.no Project	a_b	b_b	c_b	d_b
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

KLOC

Que Sl.no size 20 KLOC

	Organic	Semidetached	Embedded
Effort	55.75 PM	85.45 PM	131 PM
Duration	11.52 m	11.8 m	11.89 m
Team size	5	7	11

Effort

$$DE = a_b (KLOC)^{b_b}$$

$$= 2.4 (20)^{1.05} = 55.75 \text{ PM}$$

Duration $D_D = C_b (DE)^{d_b}$

$$= 2.5 (55.75)^{0.38}$$

$$= 11.52 \text{ CM}$$

Team size $= DE / D_D = 55.75 / 11.52$

Detailed Structure of COCOMO Model

Intermediate COCOMO —

An extension of Basic Cocomo model that defines and refines effort and duration estimates by considering additional project factors - such as product attributes, hardware constraints, team attributes.

Effort Adjustment factor accounts for the influence of cost drivers. At this stage team size and duration can be calculated easily.

mode	a	b
organic	3.2	1.05
semidetached	3.0	1.12
Embedded	2.8	1.20

20 KLOC | organic | semidetached | Embedded

EAF = 1.25

Effort | 92.9 PM | 107.4 PM | 127.4 PM

$DE = a(KLOC)^b \times EAF$ (Effort Adjustment factor)

Development Effort

NIET Greater Noida
GET FUTURE READY
Estd 2001
AN AUTONOMOUS INSTITUTE

Detailed Structure of COCOMO Model

(5) Detailed cocomo :- most comprehensive and accurate cocomo model that divides a sw project into multiple components & modules and accounts for interactions b/w cost drivers and project phases.

Estimates effort and duration for each component using the intermediate cocomo and then sums up.

Considers software reuse, hardware constraints and team in estimation.

Detailed Structure of COCOMO Model

```
import java.util.Arrays;


public class BasicCOCOMO
{
   private static final double[][] TABLE =
   {
      {2.4, 1.05, 2.5, 0.38},
      {3.0, 1.12, 2.5, 0.35},
      {3.6, 1.20, 2.5, 0.32}
   };
   private static final String[] MODE =
   {
      "Organic", "Semi-Detached", "Embedded"
   };
```

Detailed Structure of COCOMO Model

```
public static void calculate(int size)
    {
        int model = 0;
        // Check the mode according to size
        if (size >= 2 && size <= 50)
        {
            model = 0;
        } else if (size > 50 && size <= 300)
        {
            model = 1;
        } else if (size > 300)
        {
            model = 2;
        }
```

Detailed Structure of COCOMO Model

```
{
    model = 2;
}

System.out.println("The mode is " + MODE[model]);

// Calculate Effort
double effort = TABLE[model][0] * Math.pow(size,
                                    TABLE[model][1]);
```

Detailed Structure of COCOMO Model

```
System.out.println("The mode is " + MODE[model]); // Calculate Effort
      double effort = TABLE[model][0] * Math.pow(size,
TABLE[model][1]);
// Calculate Time
      double time = TABLE[model][2] * Math.pow(effort,
TABLE[model][3]);
      // Calculate Persons Required      double staff = effort / time;
      // Output the values calculated
      System.out.println("Effort = " + Math.round(effort) +" Person-
Month");
      System.out.println("Development Time = " + Math.round(time) +
                  " Months");
      System.out.println("Average Staff Required = " + Math.round(staff)
+  " Persons"); }
```

Detailed Structure of COCOMO Model

```
public static void main(String[] args)
    {
        int size = 4;
        calculate(size);
    }
}
```

Output

The mode is Organic

Effort = 10.289 Person-Month

Development Time = 6.06237 Months

Average Staff Required = 2 Persons

Detailed Structure of COCOMO Model

Intermediate Model

The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software systems. However, in reality, no system's effort and schedule can be solely calculated based on Lines of Code. For that, various other factors such as reliability, experience, and Capability. These factors are known as Cost Drivers and the Intermediate Model utilizes 15 such drivers for cost estimation.

Classification of Cost Drivers and their Attributes:

Product attributes:

Required software reliability extent

Size of the application database

The complexity of the product

Run-time performance constraints

Detailed Structure of COCOMO Model

Memory constraints

The volatility of the virtual machine environment

Required turnabout time

Analyst capability

Software engineering capability

Application experience

Virtual machine experience

Programming language experience

Use of software tools

Application of software engineering methods

Required development schedule

CASE Studies and Examples

NASA Space Shuttle Software Development: NASA estimated the time and money needed to build the software for the Space Shuttle program using the COCOMO model. NASA was able to make well-informed decisions on resource allocation and project scheduling by taking into account variables including project size, complexity, and team experience.

Big Business Software Development: The COCOMO model has been widely used by big businesses to project the time and money needed to construct intricate business software systems. These organizations were able to better plan and allocate resources for their software projects by using COCOMO's estimation methodology.

Commercial Software goods: The COCOMO methodology has proven advantageous for software firms that create commercial goods as well. These businesses were able to decide on pricing, time-to-market, and resource allocation by precisely calculating the time and expense of building new software products or features.

Academic Research Initiatives: To estimate the time and expense required to create software prototypes or carry out experimental studies, academic research initiatives have employed COCOMO. Researchers were able to better plan their projects and allocate resources by using COCOMO's estimate approaches.

CASE Studies and Examples

Advantages of the COCOMO Model

Systematic cost estimation: Provides a systematic way to estimate the cost and effort of a software project.

Helps to estimate cost and effort: This can be used to estimate the cost and effort of a software project at different stages of the development process.

Helps in high-impact factors: Helps in identifying the factors that have the greatest impact on the cost and effort of a software project.

Helps to evaluate the feasibility of a project: This can be used to evaluate the feasibility of a software project by estimating the cost and effort required to complete it.

CASE Studies and Examples

Disadvantages of the COCOMO Model

Assumes project size as the main factor: Assumes that the size of the software is the main factor that determines the cost and effort of a software project, which may not always be the case.

Does not count development team-specific characteristics: Does not take into account the specific characteristics of the development team, which can have a significant impact on the cost and effort of a software project.

Not enough precise cost and effort estimate: This does not provide a precise estimate of the cost and effort of a software project, as it is based on assumptions and averages.

Software Configuration Management (CO5)

Configuration Management

- SCM is used to control the process of software development and software maintenance.

- The configuration management is different in development and maintenance phases of life cycle due to different environments.

Software Configuration Management Activities

The activities are divided into four broad categories.

- ✓ The identification of the components and changes
- ✓ The control of the way by which the changes are made
- ✓ Auditing the changes
- ✓ Status accounting recording and documenting all the activities that have take place

Software Configuration Management Activities

The following documents are required for these activities

- Project plan
- Software requirements specification document
- Software design description document
- Source code listing
- Test plans / procedures / test cases
- User manuals

Software Configuration Management Activities

Principal Activities of SCM:

1. Configuration identification: which part s of a system should keep track of.

- Configuration control: ensure change to a system happen smoothly.

- Necessity of SCM
- Inconsistency problem when object is replicated.
- Problems associated with concurrent accesses.
- Providing a stable development environment.
- System accounting and maintaining status.

Version Control (CO5)

1

A version control tool is the first stage towards being able to manage multiple versions. Once it is in place, a detailed record of every version of the software must be kept. This comprises the

- Name of each source code component, including the variations and revisions
- The versions of the various compilers and linkers used
- The name of the software staff who constructed the component
- The date and the time at which it was constructed

2 Change Control

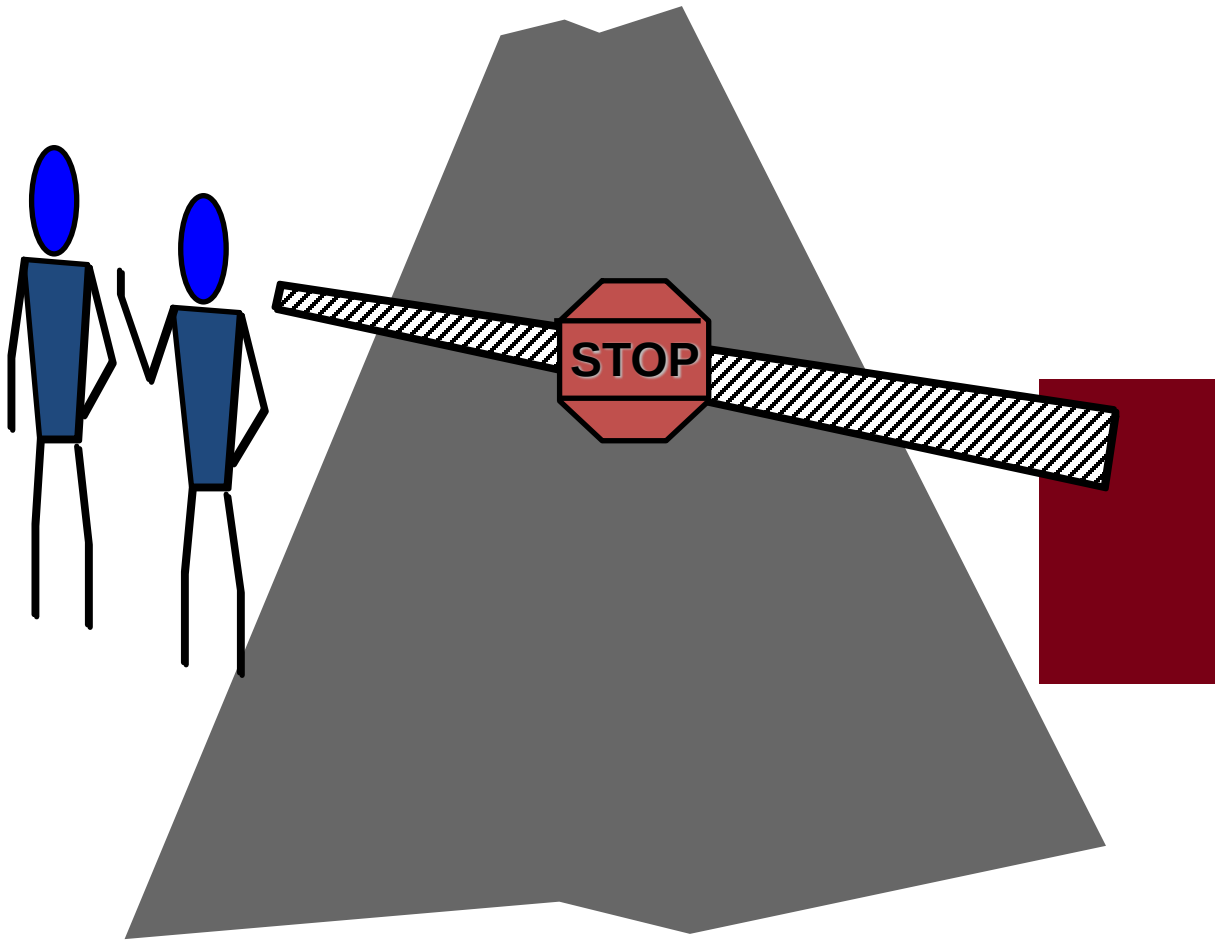
Change Control Process (CO5)

Change control process comes into effect when the software and associated documentation are delivered to configuration management change request form, which should record the recommendations regarding the change.

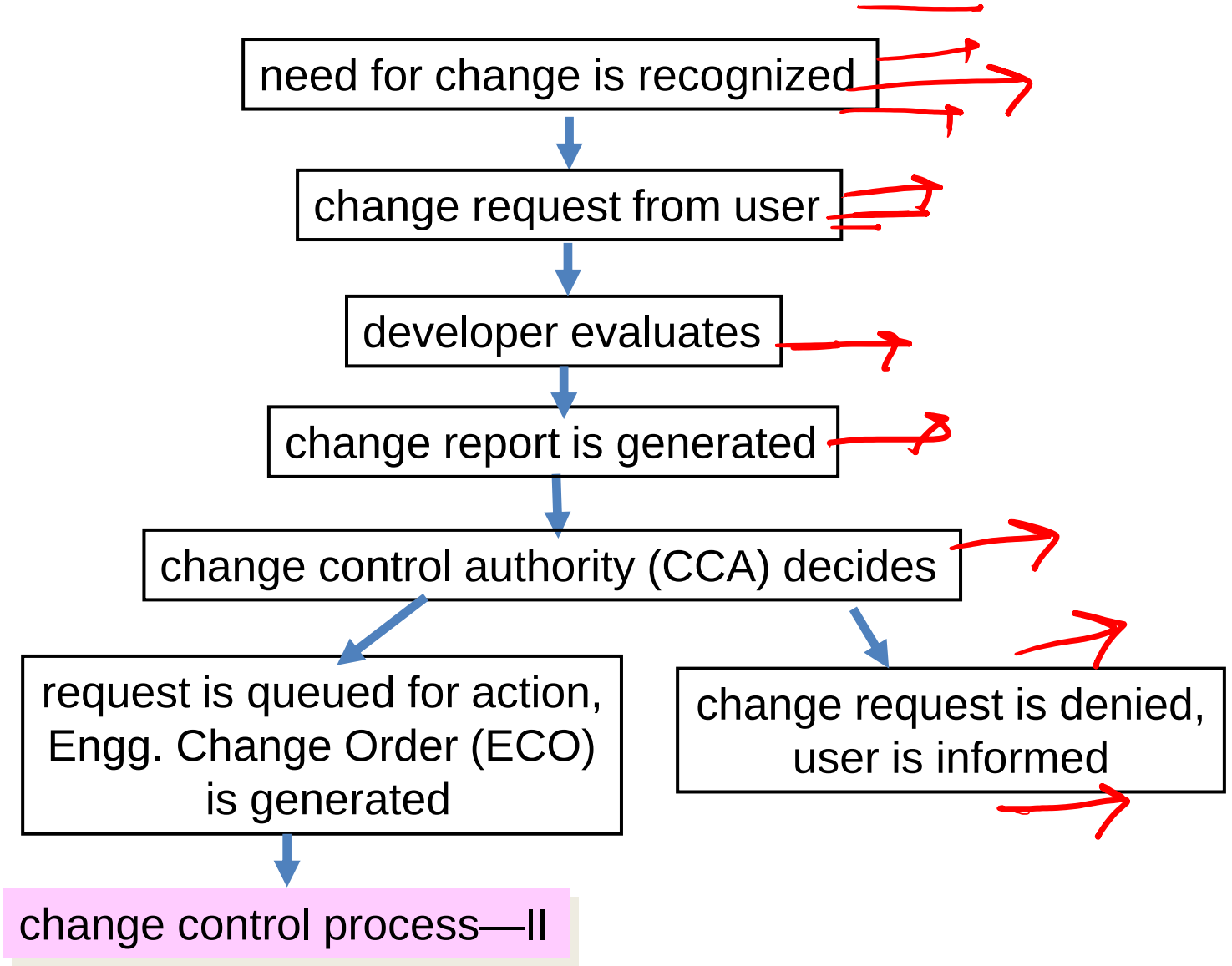
Change control

- Change control process
  - Access control and synchronization control
- Change control
  - Before an SCI becomes a baseline
  - Once an SCI becomes a baseline
- Role of Change Control Authority (CCA) – Take a global view (big picture)
  - How will the change impact h/w?
  - How will the change impact performance?
  - How will the change modify the customer's perception of the product?
  - How will the change affect the product quality and reliability?

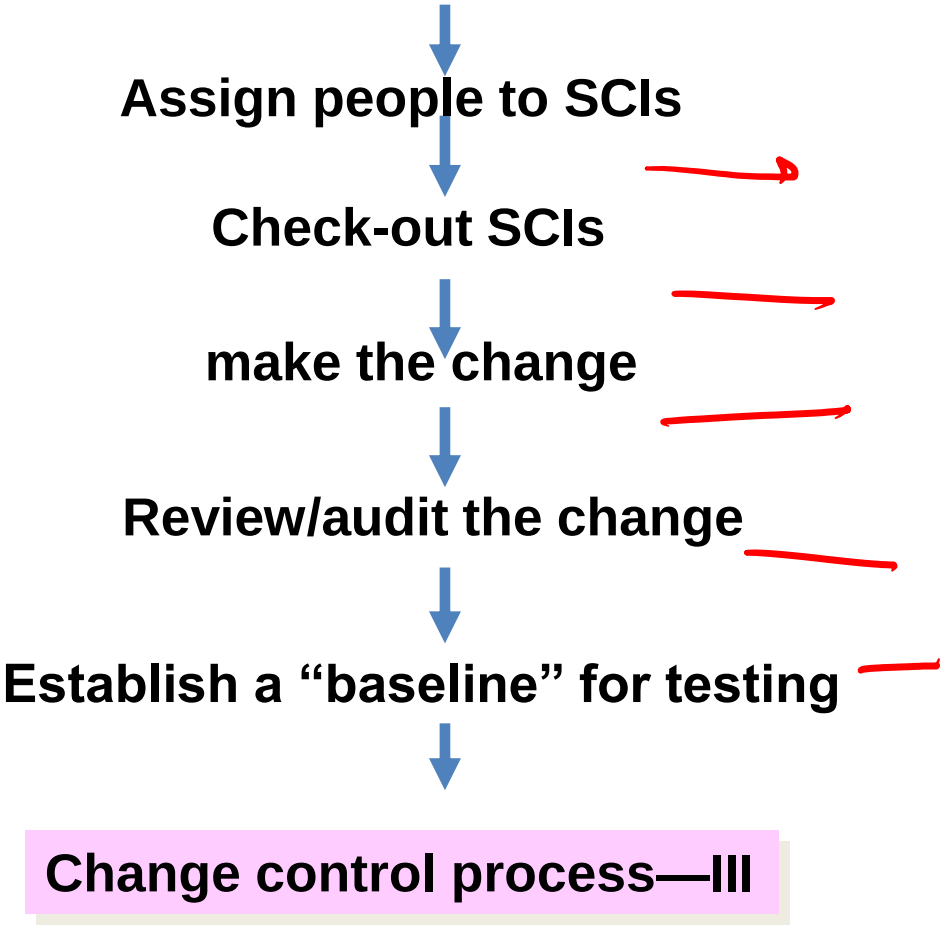
Change control



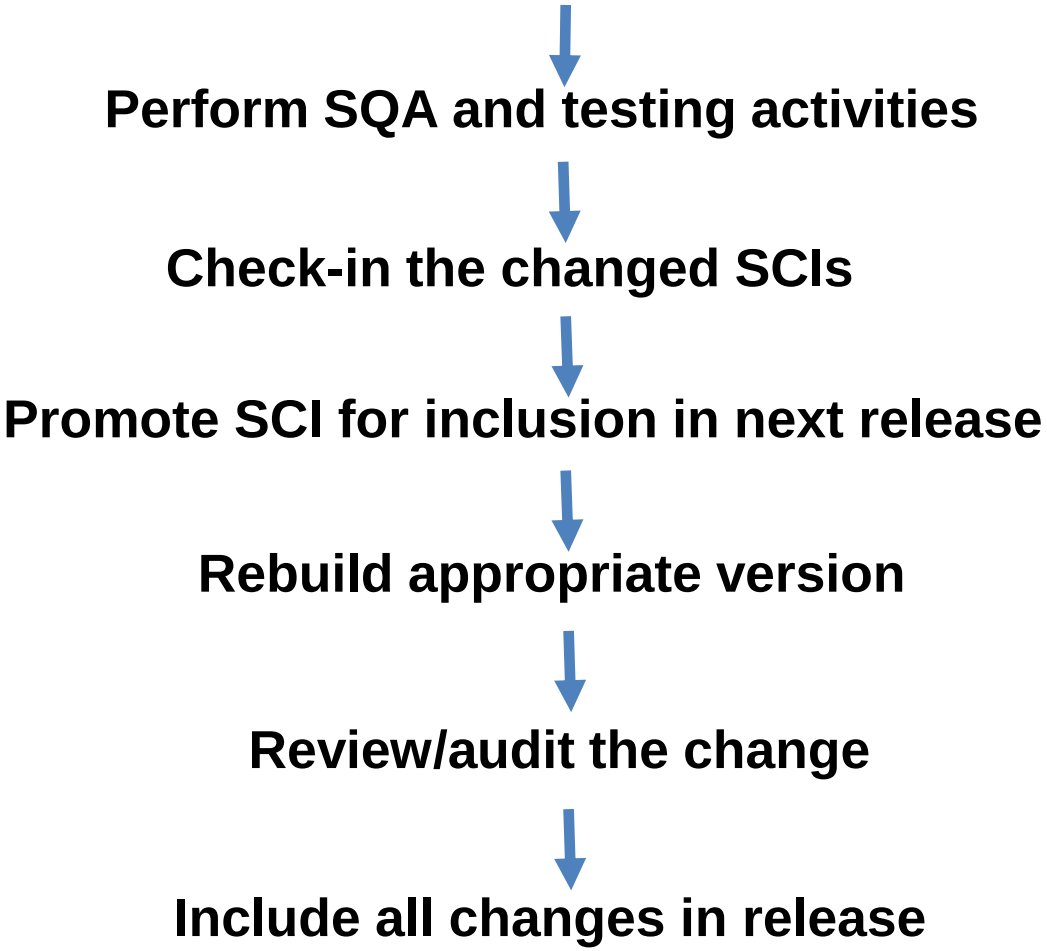
Change Control Process—I



Change Control Process—II



Change Control Process—III



Configuration audit

- How can we ensure that the change has been properly implemented?
  - Formal technical reviews
    - Focuses on the technical correctness of the configuration object that has been modified
  - Software configuration audit
    - Complements FTR by assessing a configuration object for characteristics that are generally not considered during review

System configuration management –

Whenever software is built, there is always scope for improvement and those improvements bring picture changes.

Changes may be required to modify or update any existing solution or to create a new solution for a problem.

Requirements keep on changing daily so we need to keep on upgrading our systems based on the current requirements and needs to meet desired outputs.

Changes should be analyzed before they are made to the existing system, recorded before they are implemented, reported to have details of before and after, and controlled in a manner that will improve quality and reduce error.

System configuration management –

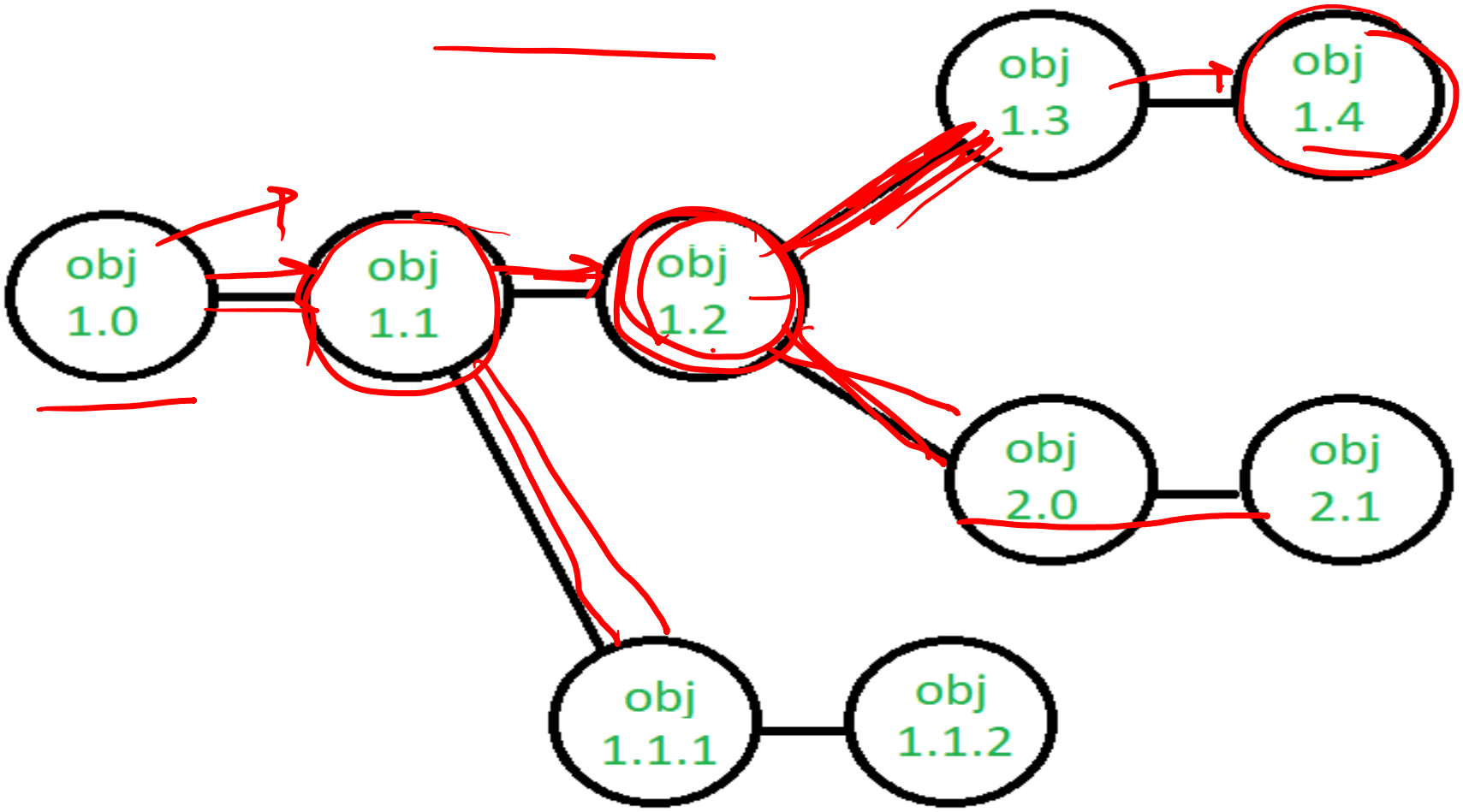
System Configuration Management (SCM) is an arrangement of exercises that controls change by recognizing the items for change, setting up connections between those things, making/characterizing instruments for overseeing diverse variants, controlling the changes being executed in the current framework, inspecting and revealing/reporting on the changes made.

Processes involved in SCM – Configuration management provides a disciplined environment for smooth control of work products. It involves the following activities:

1.Identification and Establishment – Identifying the configuration items from products that compose baselines at given points in time (a baseline is a set of mutually consistent Configuration Items, which has been formally reviewed and agreed upon, and serves as the basis of further development). Establishing relationships among items, creating a mechanism to manage multiple levels of control and procedure for the change management system.

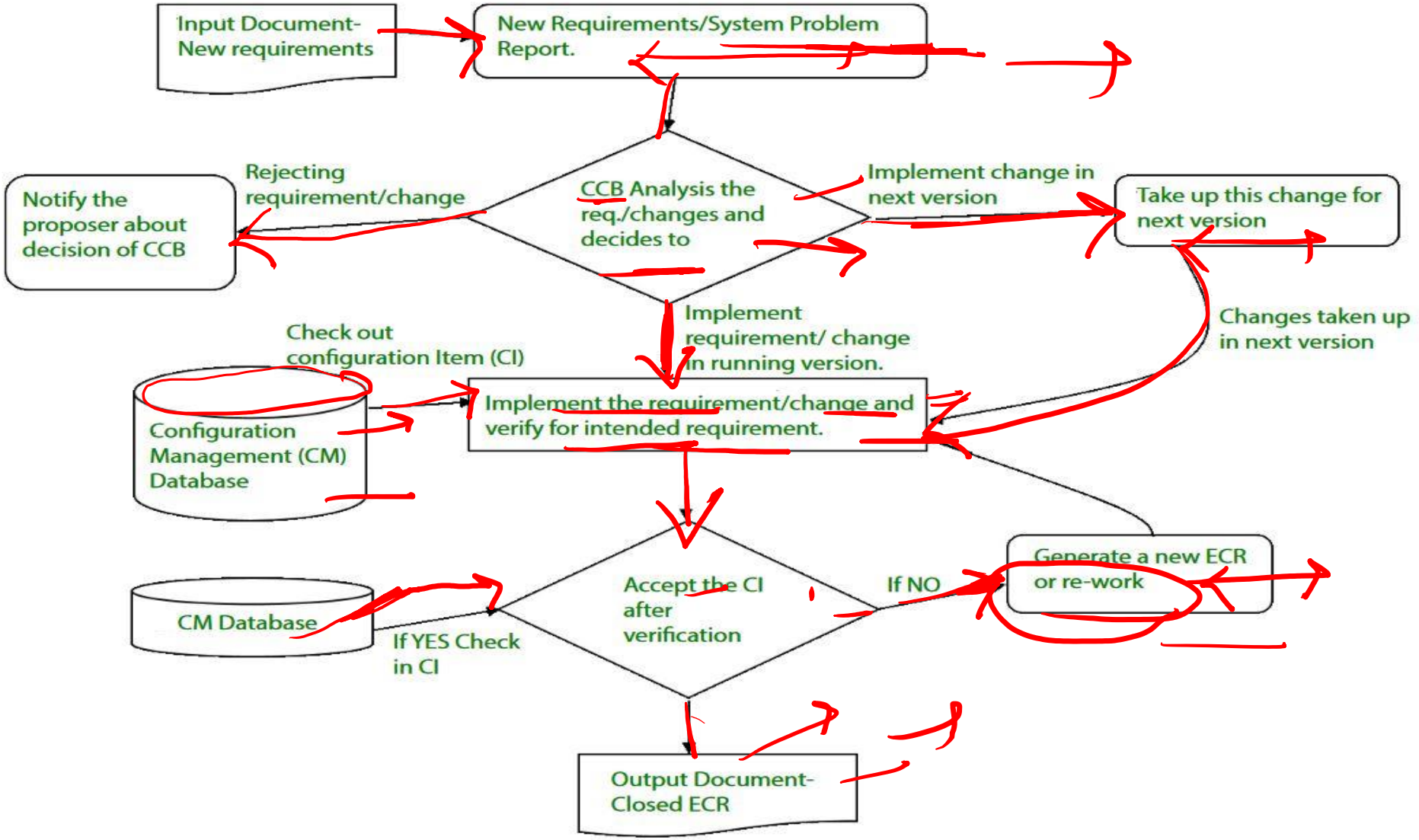
System configuration management –

Version control – Creating versions/specifications of the existing product to build new products with the help of the SCM system. A description of the version is given below:



System configuration management –

Change control – Controlling changes to Configuration items (CI). The change control process is explained



System configuration management –

A change request (CR) is submitted and evaluated to assess technical merit, potential side effects, the overall impact on other configuration objects and system functions, and the projected cost of the change. The results of the evaluation are presented as a change report, which is used by a change control board (CCB) —a person or group who makes a final decision on the status and priority of the change.

An engineering change Request (ECR) is generated for each approved change. Also, CCB notifies the developer in case the change is rejected with proper reason. The ECR describes the change to be made, the constraints that must be respected, and the criteria for review and audit. The object to be changed is “checked out” of the project database, the change is made, and then the object is tested again.

System configuration management –

4.Configuration auditing – A software configuration audit complements the formal technical review of the process and product. It focuses on the technical correctness of the configuration object that has been modified. The audit confirms the completeness, correctness, and consistency of items in the SCM system and tracks action items from the audit to closure.

5.Reporting – Providing accurate status and current configuration data to developers, testers, end users, customers, and stakeholders through admin guides, user guides, FAQs, Release notes, Memos, Installation Guide, Configuration guides, etc.

Importance of Software Configuration Management

1.Effective Bug Tracking: Linking code modifications to issues that have been reported, makes bug tracking more effective.

2.Continuous Deployment and Integration: SCM combines with continuous processes to automate deployment and testing, resulting in more dependable and timely software delivery.

3.Risk management: SCM lowers the chance of introducing critical flaws by assisting in the early detection and correction of problems.

4.Support for Big Projects: Source Code Control (SCM) offers an orderly method to handle code modifications for big projects, fostering a well-organized development process.

5.Reproducibility: By recording precise versions of code, libraries, and dependencies, source code versioning (SCM) makes builds repeatable.

6.Parallel Development: SCM facilitates parallel development by enabling several developers to collaborate on various branches at once.

Key objectives of SCM

1.Control the evolution of software systems: SCM helps to ensure that changes to a software system are properly planned, tested, and integrated into the final product.

2.Enable collaboration and coordination: SCM helps teams to collaborate and coordinate their work, ensuring that changes are properly integrated and that everyone is working from the same version of the software system.

3.Provide version control: SCM provides version control for software systems, enabling teams to manage and track different versions of the system and to revert to earlier versions if necessary.

4.Facilitate replication and distribution: SCM helps to ensure that software systems can be easily replicated and distributed to other environments, such as test, production, and customer sites.

The main advantages of SCM

1.Improved productivity and efficiency by reducing the time and effort required to manage software changes.

2.Reduced risk of errors and defects by ensuring that all changes were properly tested and validated.

3.Increased collaboration and communication among team members by providing a central repository for software artifacts.

4.Improved quality and stability of software systems by ensuring that all changes are properly controlled and managed.

The main disadvantages of SCM

1.Increased complexity and overhead, particularly in large software systems.

2.Difficulty in managing dependencies and ensuring that all changes are properly integrated.

3.Potential for conflicts and delays, particularly in large development teams with multiple contributors.

Reverse Engineering (CO5)

Reverse engineering is the process followed in order to find difficult, unknown and hidden information about a software system.

- Mapping between concrete and abstract levels
- Rediscovering high level structures
- Finding missing links between program syntax and semantics
- To extract reusable component

Reverse Engineering

Scope and Tasks

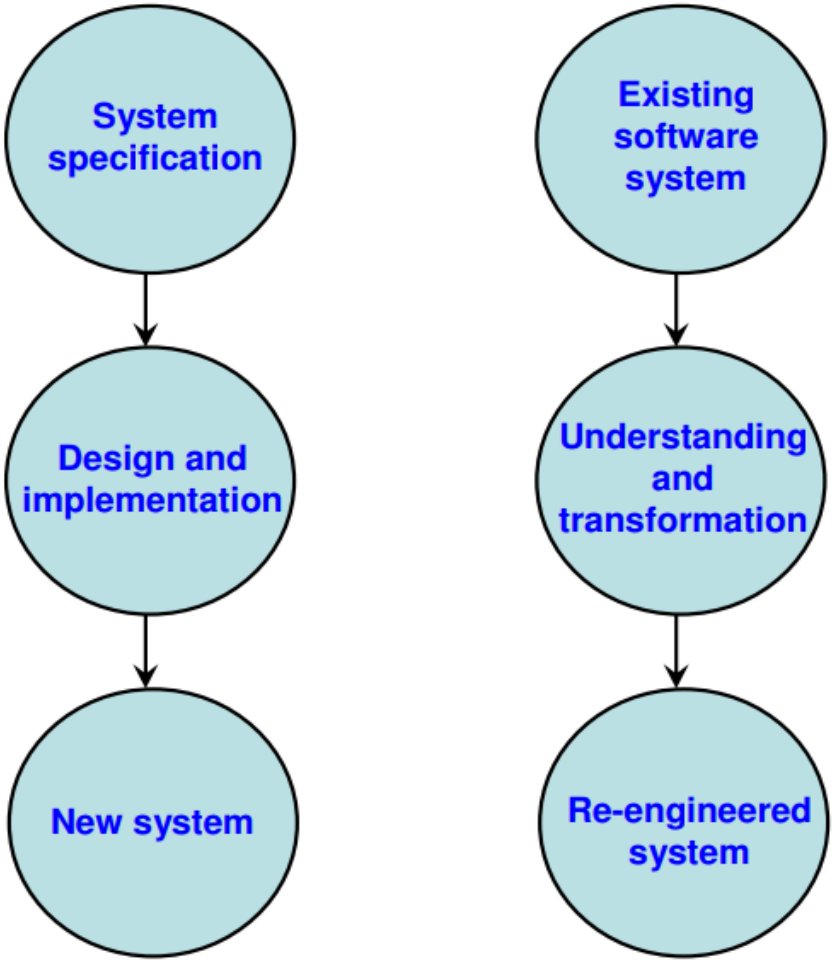
The areas where reverse engineering is applicable include (but not limited to):

1. Program comprehension
2. Re-documentation and/ or document generation
3. Recovery of design approach and design details at any level of abstraction
4. Identifying reusable components
5. Identifying components that need restructuring
6. Recovering business rules, and
7. Understanding high level system description

Re-Engineering

- Software re-engineering is concerned with taking existing legacy systems and re-implementing them to make them more maintainable.
- The critical distinction between re-engineering and new software development is the starting point for the development as shown in Fig. in next slide

Re-Engineering



It is the combination of two consecutive process

1. Forward Engineering
2. Reverse Engineering

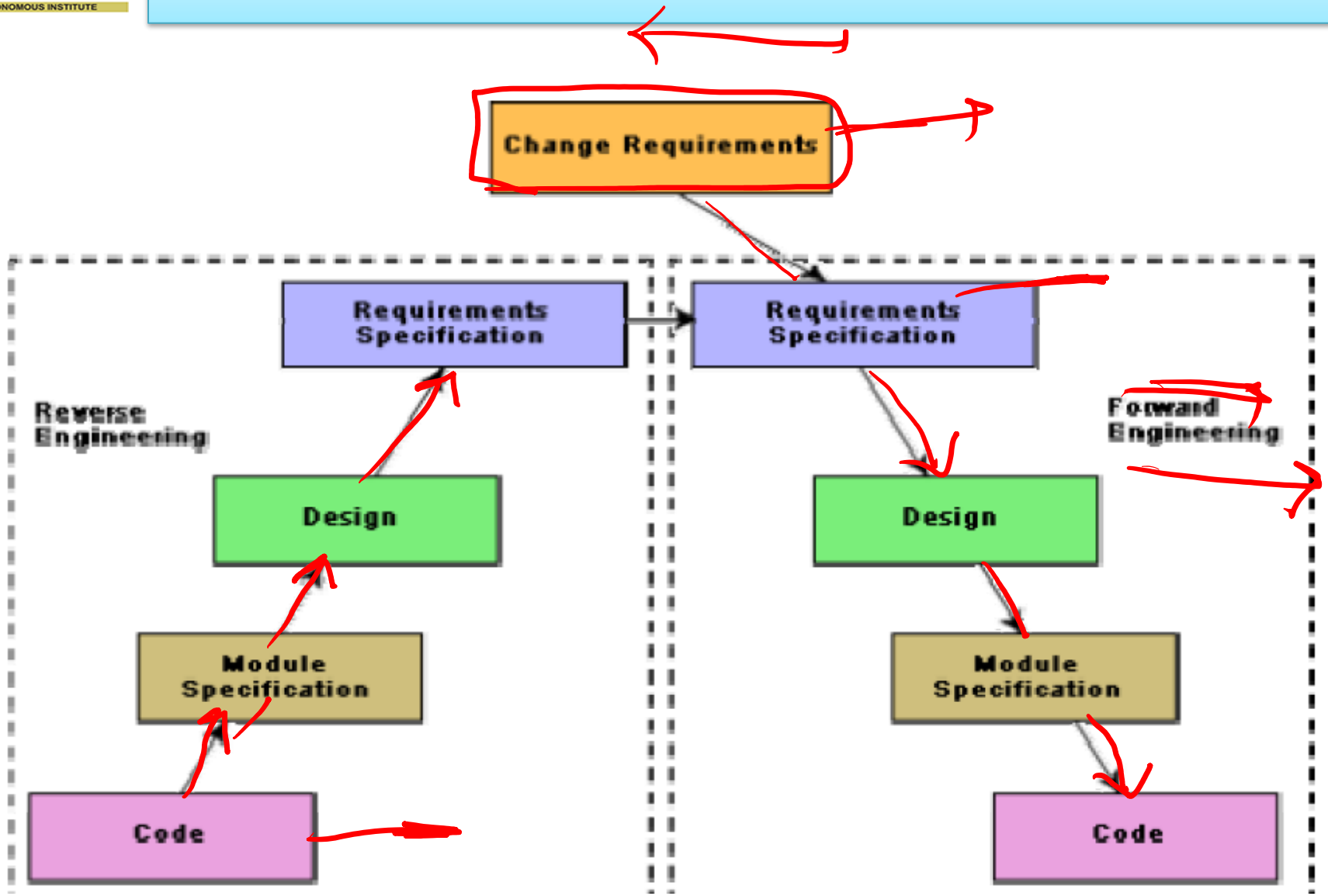
Objective of Re-Engineering

- Prepare functional enhancement.
- Improve maintainability.
- Enhance skills of software developers to incorporate newer technologies.
- Improve reliability.
- Apply integration.

Steps in Re- Engineering

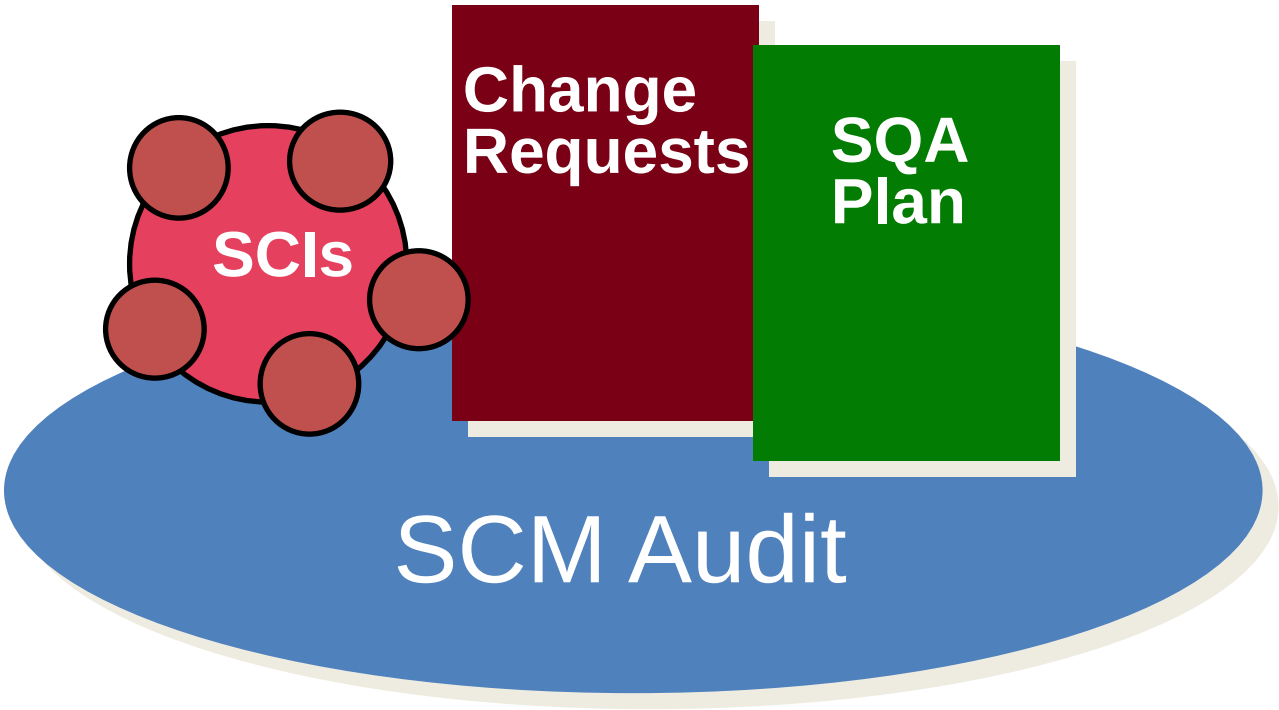
1. Goal setting.
2. Critical analysis of existing scenario such as process, task, design, methods etc.
3. Identifying the problems and solving them by new innovative thinking.

Re-Engineering



Configuration audit

Auditing



Re-engineering

- Software Re-engineering is a process of software development that is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of a system to reconstitute it in a new form. This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing, etc.

- software re-engineering, is the process of analyzing, designing, and modifying existing software systems to improve their quality, performance, and maintainability.

Re-engineering

- It includes updating the software to work with new hardware or software platforms, adding new features, or improving the software's overall design and architecture.

- Software re-engineering, also known as software restructuring or software renovation, refers to the process of improving or upgrading existing software systems to improve their quality, maintainability, or functionality.

- It involves reusing the existing software artifacts, such as code, design, and documentation, and transforming them to meet new or updated requirements.

Objective of Re-engineering

- To describe a cost-effective option for system evolution.
- To describe the activities involved in the software maintenance process.
- To distinguish between software and data re-engineering and to explain the problems of data re-engineering.

process of software re-engineering

- **Planning:** The first step is to plan the re-engineering process, which involves identifying the reasons for re-engineering, defining the scope, and establishing the goals and objectives of the process.

- **Analysis:** The next step is to analyze the existing system, including the code, documentation, and other artifacts. This involves identifying the system's strengths and weaknesses, as well as any issues that need to be addressed.

- **Design:** Based on the analysis, the next step is to design the new or updated software system. This involves identifying the changes that need to be made and developing a plan to implement them.

process of software re-engineering

- **Implementation:** The next step is to implement the changes by modifying the existing code, adding new features, and updating the documentation and other artifacts.

- **Testing:** Once the changes have been implemented, the software system needs to be tested to ensure that it meets the new requirements and specifications.

- **Deployment:** The final step is to deploy the re-engineered software system and make it available to end-users.

Process of Software re-engineering

- **To improve the software's performance and scalability:** By analyzing the existing code and identifying bottlenecks, re-engineering can be used to improve the software's performance and scalability.
- **To add new features:** Re-engineering can be used to add new features or functionality to existing software.
- **To support new platforms:** Re-engineering can be used to update existing software to work with new hardware or software platforms.
- **To improve maintainability:** Re-engineering can be used to improve the software's overall design and architecture, making it easier to maintain and update over time.
- **To meet new regulations and compliance:** Re-engineering can be done to ensure that the software is compliant with new regulations and standards.

Process of Software re-engineering

- **Improving software quality:** Re-engineering can help improve the quality of software by eliminating defects, improving performance, and enhancing reliability and maintainability.

- **Updating technology:** Re-engineering can help modernize the software system by updating the technology used to develop, test, and deploy the system.

- **Enhancing functionality:** Re-engineering can help enhance the functionality of the software system by adding new features or improving existing ones.

- **Resolving issues:** Re-engineering can help resolve issues related to scalability, security, or compatibility with other systems.

Steps of Software re-engineering

- **Assessment:** The existing software system is analyzed to identify its strengths and weaknesses, and to determine the extent of the required re-engineering effort.
- **Planning:** A plan is developed to guide the re-engineering effort, including the identification of goals, objectives, and performance metrics.
- **Requirements analysis:** The requirements of the new system are defined based on user needs and business requirements.
- **Architecture and design:** The architecture and design of the new system are developed, taking into account the requirements of the new system and the limitations of the existing system.

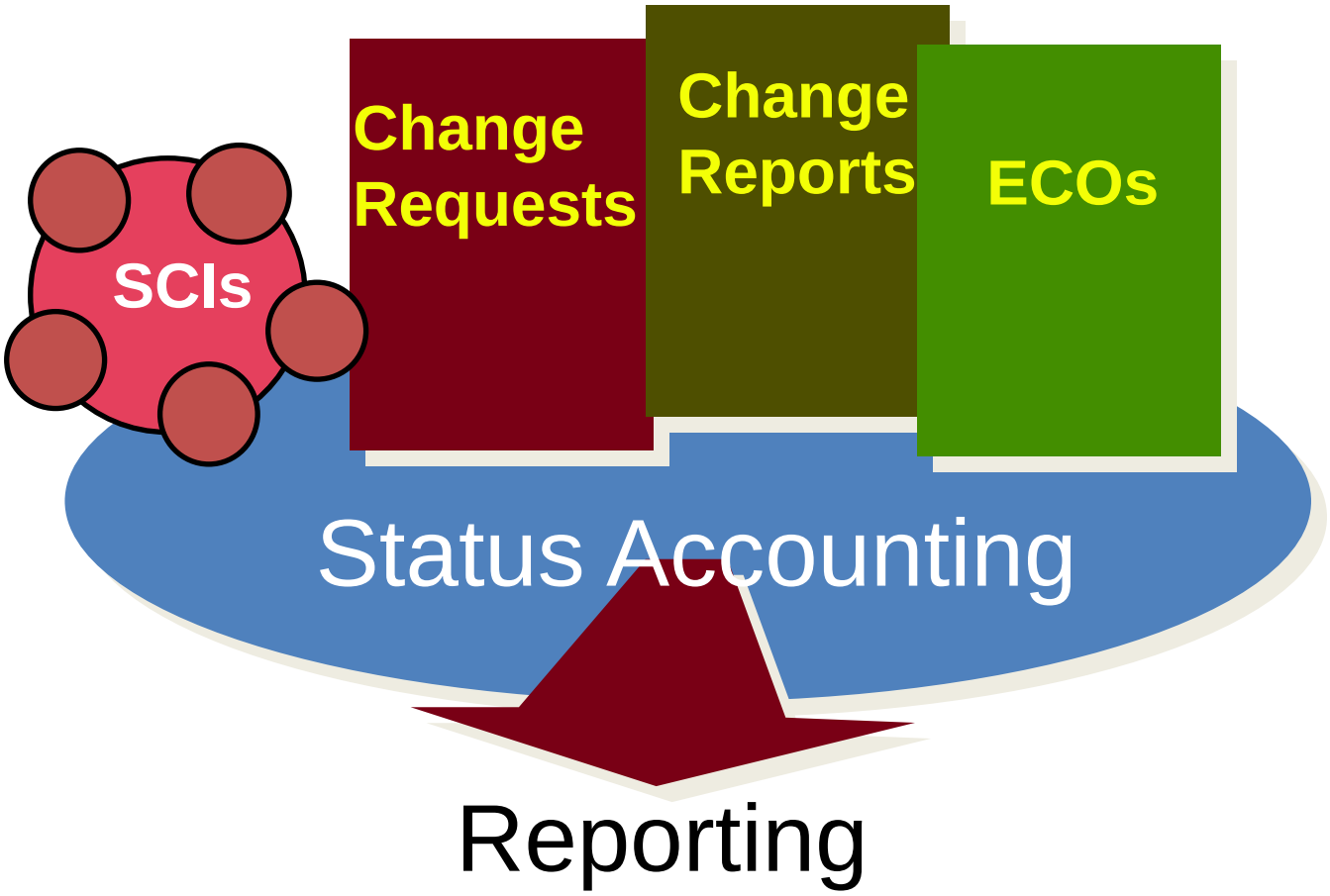
Steps of Software re-engineering

- **Implementation:** The new system is developed and implemented, either by modifying the existing system or by developing a new system from scratch.

- **Testing:** The new system is tested to ensure that it meets the requirements of the new system and is free from errors and defects.

- **Maintenance:** The new system is maintained over time to ensure that it remains reliable, performant, and secure.

Status Accounting



CASE Tool (CO5)

CASE (Computer Aided Software Engineering)tool
a CASE tool used to automate some activity associated with software development.

CASE tools assist

- Phase activities such as specification, structured analysis, design, coding, testing, etc.;
- Non-phase activities such as project management and configuration management.

Reasons for using CASE tools

The primary reasons for using a CASE tool are:

- To increase productivity
- To help produce better quality software at lower cost

CASE Tool (CO5)

Computer-aided software engineering (CASE) is the implementation of computer-facilitated tools and methods in software development. CASE is used to ensure high-quality and defect-free software. CASE ensures a check-pointed and disciplined approach and helps designers, developers, testers, managers, and others to see the project milestones during development.

CASE can also help as a warehouse for documents related to projects, like business plans, requirements, and design specifications.

CASE illustrates a wide set of labor-saving tools that are used in software development. It generates a framework for organizing projects and to be helpful in enhancing productivity.

CASE Tool (CO5)

The essential idea of CASE tools is that in-built programs can help to analyze developing systems in order to enhance quality and provide better outcomes. Throughout the 1990, CASE tool became part of the software lexicon, and big companies like IBM were using these kinds of tools to help create software.

Types of CASE Tools:

1.Diagramming Tools: It helps in diagrammatic and graphical representations of the data and system processes. It represents system elements, control flow and data flow among different software components and system structures in a pictorial form. For example, Flow Chart Maker tool for making state-of-the-art flowcharts.

2.Computer Display and Report Generators: These help in understanding the data requirements and the relationships involved.

3.Analysis Tools: It focuses on inconsistent, incorrect specifications involved in the diagram and data flow. It helps in collecting requirements, automatically check for any irregularity, imprecision in the diagrams, data redundancies, or erroneous omissions. For example:

3. (i) Accept 360, Accompa, Case Complete for requirement analysis.
4. (ii) Visible Analyst for total analysis.
5.

4.Central Repository: It provides a single point of storage for data diagrams, reports, and documents related to project management.

Types of CASE Tools:

4.Central Repository: It provides a single point of storage for data diagrams, reports, and documents related to project management.

5.Documentation Generators: It helps in generating user and technical documentation as per standards. It creates documents for technical users and end users.
For example, Doxygen, DrExplain, Adobe RoboHelp for documentation.

6.Code Generators: It aids in the auto-generation of code, including definitions, with the help of designs, documents, and diagrams.
7.Tools for Requirement Management: It makes gathering, evaluating, and managing software needs easier.

8.Tools for Analysis and Design: It offers instruments for modelling system architecture and behaviour, which helps throughout the analysis and design stages of software development.

9.Tools for Database Management: It facilitates database construction, design, and administration.
10.Tools for Documentation: It makes the process of creating, organizing, and maintaining project documentation easier.

Advantages of the CASE approach:

•**Improved Documentation:** Comprehensive documentation creation and maintenance is made easier by CASE tools. Since automatically generated documentation is usually more accurate and up to date, there are fewer opportunities for errors and misunderstandings brought on by out-of-current material.

•**Reusing Components:** Reusable component creation and maintenance are frequently facilitated by CASE tools. This encourages a development approach that is modular and component-based, enabling teams to shorten development times and reuse tested solutions.

•**Quicker Cycles of Development:** Development cycles take less time when certain jobs, such testing and code generation, are automated. This may result in software solutions being delivered more quickly, meeting deadlines and keeping up with changing business requirements.

Advantages of the CASE approach:

- **Improved Results**: Code generation, documentation, and testing are just a few of the time-consuming, repetitive operations that CASE tools perform. Due to this automation, engineers are able to concentrate on more intricate and imaginative facets of software development, which boosts output.

- **Achieving uniformity and standardization:** Coding conventions, documentation formats and design patterns are just a few of the areas of software development where CASE tools enforce uniformity and standards. This guarantees consistent and maintainable software development.

Disadvantages of the CASE approach:

•**Cost:** Using a case tool is very costly. Most firms engaged in software development on a small scale do not invest in CASE tools because they think that the benefit of CASE is justifiable only in the development of large systems.

•**Learning Curve:** In most cases, programmers' productivity may fall in the initial phase of implementation, because users need time to learn the technology. Many consultants offer training and on-site services that can be important to accelerate the learning curve and to the development and use of the CASE tools.

•**Tool Mix:** It is important to build an appropriate selection tool mix to urge cost advantage CASE integration and data integration across all platforms is extremely important.

CASE Tool (CO5)

CASE (Computer Aided Software Engineering)tool
a CASE tool used to automate some activity associated with software development.

CASE tools assist

- Phase activities such as specification, structured analysis, design, coding, testing, etc.;
- Non-phase activities such as project management and configuration management.

Reasons for using CASE tools

The primary reasons for using a CASE tool are:

- To increase productivity
- To help produce better quality software at lower cost

List of CASE Tool

Application.	CASE tool	Purpose of tool
Planning	Excel spreadsheet, ms project, pert network, estimation tools	Functional app.: planning, scheduling, control
Editing	Dig. Editor, text editor, word processor	Speed and efficiency
Testing	Test data generator, file comparator	Speed and efficiency
Prototyping	High level modeling language, UI generator	Confirmation and certification of SRS and SDD

List of CASE Tool

Application.	CASE tool	Purpose of tool
Documentation	Report generator, PPT presentation	Faster structural documentation
Programming and language processing integration	Program generator, code generator, compiler, interpreter	High quality with error free programming
Re engineering tool	Cross reference system, program re engineering system	Reverse engineering to fined structure, design and design information
Program analysis tool	Cross reference system, static and dynamic analyzers	Analyzes risks, functions, features.

Benefits of CASE Tool

- Improved productivity.
- Better documentation.
- Reduced lifetime maintenance.
- Improved accuracy
- Opportunity to non - programmers

Risk analysis and management (CO5)

What is risk ?

Tomorrow's problems are today's risks.

"Risk is a problem that may cause some loss or threaten the success of the project, but which has not happened yet".

Risk analysis and management

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

Current problems &

Typical Software Risk

Capers Jones has identified the top five risk factors that threaten projects in different applications

1. Dependencies on outside agencies or factors.
    - Availability of trained, experienced persons
    - Inter group dependencies
    - Customer-Furnished items or information
    - Internal & external subcontractor relationships

Risk analysis and management

2. Requirement issues

Uncertain requirements

Wrong product

or

Right product badly

Either situation results in unpleasant surprises and unhappy customers.

Risk analysis and management

2. Requirement issues

Uncertain requirements

Wrong product

or

Right product badly

Either situation results in unpleasant surprises and unhappy customers.

Risk analysis and management

3. Management Issues

Project managers usually write the risk management plans, and most people do not wish to air their weaknesses in public.

- Inadequate planning
- Inadequate visibility into actual project status
- Unclear project ownership and decision making
- Staff personality conflicts
- Unrealistic expectation
- Poor communication

Risk analysis and management

4. Lack of knowledge

- Inadequate training
- Poor understanding of methods, tools, and techniques
- Inadequate application domain experience

- New Technologies
- Ineffective, poorly documented or neglected processes

Risk analysis and management

5. Other risk categories

- Unavailability of adequate testing facilities
- Turnover of essential personnel
- Unachievable performance requirements
- Technical approaches that may not work

Risk analysis and management

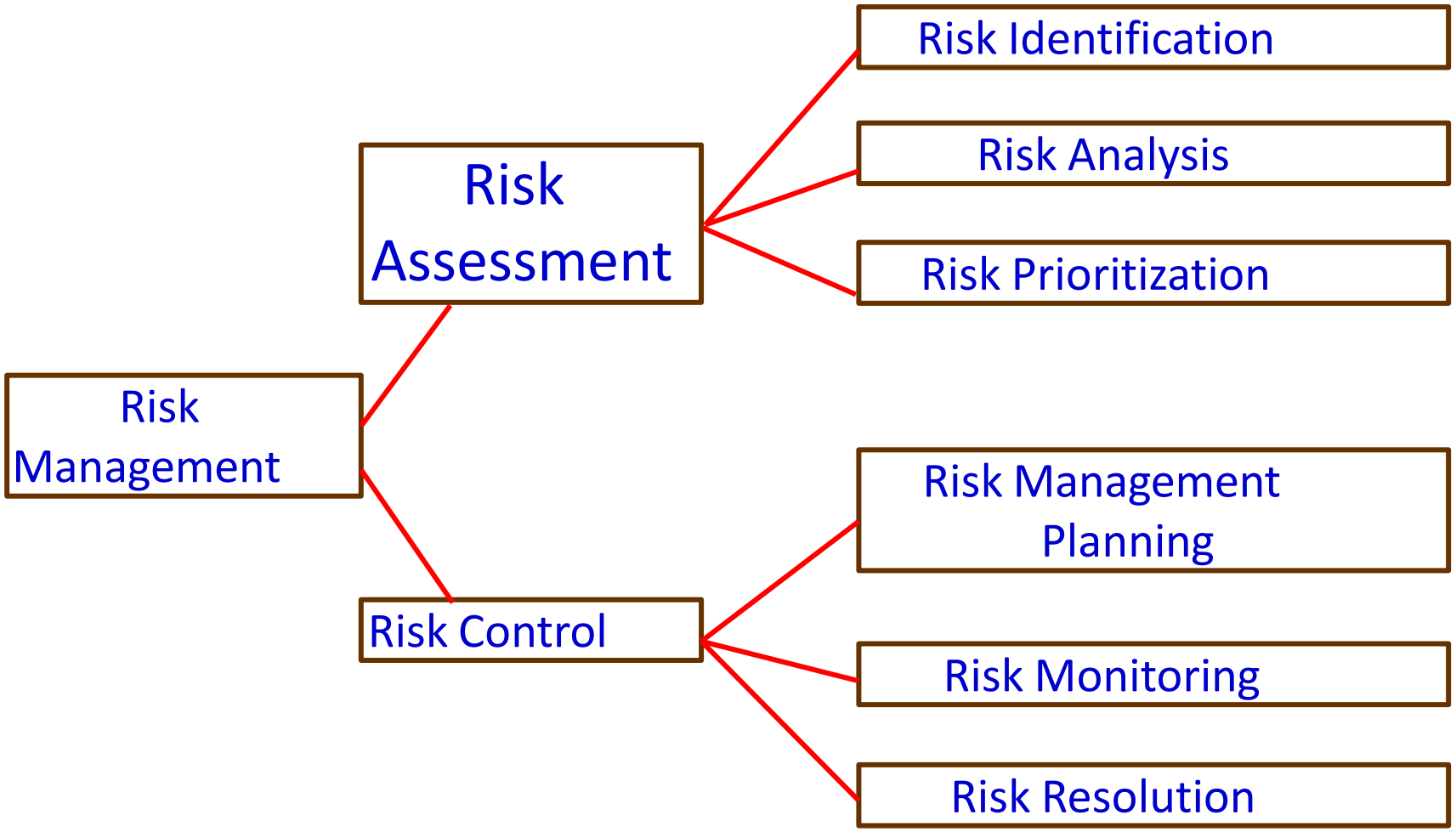


Fig. Risk Management Activities

Clean Room Software Engineering

The clean room approach was developed by Dr. Harlan Mills of IBM's Federal Systems Division, and it was released in the year 1981 but got popularity after 1987 when IBM and other organizations started using it.

Clean room software engineering is a software development approach to producing quality software. It is different from classical Software Engineering as in classical software engineering QA (Quality Assurance) is the last phase of development that occurs at the completion of all development stages while there is a chance of less reliable and fewer quality products full of bugs, and errors and upset client, etc. But in clean room software engineering, an efficient and good quality software product is delivered to the client as QA (Quality Assurance) is performed each and every phase of software development.

Processes of Clean Room Development

1.Management- It is persistent throughout the whole project lifetime which consists of project mission, schedule, resources, risk analysis, training, configuration management, etc.

2.Specification- It is considered the first process of each increment which consists of requirement analysis, function specification, usage specification, increment planning, etc.

3.Development- It is considered the second process of each increment which consists of Software Engineering , correctness verification, incremental design, etc.

4.Certification- It is considered the final process of each increment which consists of usage modeling and test planning, statistical training and certification process, etc.

Tasks in Clean Room Engineering Process

1.Requirements gathering: Compile and record the functional and non-functional specifications for the software.

2.Incremental planning: As you divide the project into smaller pieces, plan the development process in incremental steps.

3.Formal design: Make a formal design in accordance with the specifications. Detailed paperwork and mathematical requirements are frequently a part of this design process.

4.Correctness verification: Verify the design's reliability using formal techniques. This includes using other methods, such as mathematical proofs, to make sure the design satisfies the requirements.

5.Code generation and inspection: Create code by using the formal design as a guide. Code inspection includes a thorough examination of the code to ensure that it is accurate and follows coding guidelines.

6.Statical test planning: Create a plan for statistical testing based on the requirements and formal design. This plan describes how the software will be tested to make sure it satisfies its requirements.

Box Structure in Clean Room Process

Box structure is a modeling approach that is used in clean room engineering. A box is like a container that contains details about a system or aspects of a system. All boxes are independent of other boxes to deliver the required information/details. It generally uses three types of boxes i.e.

1.Black box –It identifies the behavior of the system.

2.State box –It identifies state data or operations.

3.Clear box –It identifies the transition function used by the state box.

Benefits of Clean Room Software engineering

1.Excellent Dependability: The goal of Clean Room procedures is to create extremely dependable software. The likelihood of mistakes and flaws in the finished work is decreased by the emphasis on formal methods and correctness verification.

2.Decreasing Defect Density: Defect density is significantly reduced through thorough code inspections, formal design and verification processes and other methods.

3.Early Error Identification: Early error detection by formal procedures, code inspections, and statistical testing is highly valued in the process.

4.Official Confirmation: The software design and implementation adhere to predetermined requirements thanks to the application of formal methodologies and mathematical verification tools.

5.Client Self-Assurance: The emphasis on formal verification, predictability, and dependability gives stakeholders and customers trust in the software's quality.

Software Maintenance

Software Maintenance refers to the process of modifying and updating a software system after it has been delivered to the customer. Software maintenance is a continuous process that occurs throughout the entire life cycle of the software system.

•The goal of software maintenance is to keep the software system working correctly, efficiently, and securely, and to ensure that it continues to meet the needs of the users.

•This can include fixing bugs, adding new features, improving performance, or updating the software to work with new hardware or software systems.
•It is also important to consider the cost and effort required for software maintenance when planning and developing a software system.

•It is important to have a well-defined maintenance process in place, which includes testing and validation, version control, and communication with stakeholders.
•It's important to note that software maintenance can be costly and complex, especially for large and complex systems. Therefore, the cost and effort of maintenance should be taken into account during the planning and development phases of a software project.

•It's also important to have a clear and well-defined maintenance plan that includes regular maintenance activities, such as testing, backup, and bug fixing.

Software Maintenance

Several Key Aspects of Software Maintenance

1.Bug Fixing: The process of finding and fixing errors and problems in the software.

2.Enhancements: The process of adding new features or improving existing features to meet the evolving needs of the users.

3.Performance Optimization: The process of improving the speed, efficiency, and reliability of the software.

4.Porting and Migration: The process of adapting the software to run on new hardware or software platforms.

5.Re-Engineering: The process of improving the design and architecture of the software to make it more maintainable and scalable.

6.Documentation: The process of creating, updating, and maintaining the documentation for the software, including user manuals, technical specifications, and design documents.

NIET Greater Noida GET FUTURE READY Estd 2001 AN AUTONOMOUS INSTITUTE

Several Types of Software Maintenance

1.Corrective Maintenance: This involves fixing errors and bugs in the software system.

2.Patching: It is an emergency fix implemented mainly due to pressure from management. Patching is done for corrective maintenance but it gives rise to unforeseen future errors due to lack of proper impact analysis.

3.Adaptive Maintenance: This involves modifying the software system to adapt it to changes in the environment, such as changes in hardware or software, government policies, and business rules.

4.Perfective Maintenance: This involves improving functionality, performance, and reliability, and restructuring the software system to improve changeability.

5.Preventive Maintenance: This involves taking measures to prevent future problems, such as optimization, updating documentation, reviewing and testing the system, and implementing preventive measures such as backups.

Need for Maintenance

Software Maintenance must be performed in order to:

•Correct faults.

•Improve the design.

•Implement enhancements
•.
•Interface with other systems.

•Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.

•Migrate legacy software.

•Retire software.

•Requirement of user changes.

•Run the code fast

Challenges in Software Maintenance

•There is a lack of Code Comments.
•**Lack of documentation**: Poorly documented systems can make it difficult to understand how the system works, making it difficult to identify and fix problems.
•**Legacy code:** Maintaining older systems with outdated technologies can be difficult, as it may require specialized knowledge and skills.
•**Complexity:** Large and complex systems can be difficult to understand and modify, making it difficult to identify and fix problems.
•**Changing requirements:** As user requirements change over time, the software system may need to be modified to meet these new requirements, which can be difficult and time-consuming.
•**Interoperability issues:** Systems that need to work with other systems or software can be difficult to maintain, as changes to one system can affect the other systems.
•**Lack of test coverage**: Systems that have not been thoroughly tested can be difficult to maintain as it can be hard to identify and fix problems without knowing how the system behaves in different scenarios.
•**Lack of personnel**: A lack of personnel with the necessary skills and knowledge to maintain the system can make it difficult to keep the system up-to-date and running smoothly.
•**High-Cost:** The cost of maintenance can be high, especially for large and complex systems, which can be difficult to budget for and manage.

Categories of Software Maintenance

Maintenance can be divided into the following categories.

•**Corrective maintenance:** Corrective Maintanance of a software product may be essential either to rectify some bugs observed while the system is in use, or to enhance the performance of the system.

•**Adaptive maintenance:** This includes modifications and updations when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware and software.

•**Perfective maintenance:** A software product needs maintenance to support the new features that the users want or to change different types of functionalities of the system according to the customer's demands.

•**Preventive maintenance:** This type of maintenance includes modifications and updations to prevent future problems with the software. It goals to attend to problems, which are not significant at this moment but may cause serious issues in the future.

Advantages of Software Maintenance

•**Improved Software Quality:** Regular software maintenance helps to ensure that the software is functioning correctly and efficiently and that it continues to meet the needs of the users.

•**Enhanced Security:** Maintenance can include security updates and patches, helping to ensure that the software is protected against potential threats and attacks.

•**Increased User Satisfaction:** Regular software maintenance helps to keep the software up-to-date and relevant, leading to increased user satisfaction and adoption.

•**Extended Software Life:** Proper software maintenance can extend the life of the software, allowing it to be used for longer periods of time and reducing the need for costly replacements.

•**Cost Savings:** Regular software maintenance can help to prevent larger, more expensive problems from occurring, reducing the overall cost of software ownership.

•**Better Alignment with business goals:** Regular software maintenance can help to ensure that the software remains aligned with the changing needs of the business. This can help to improve overall business efficiency and productivity.

Advantages of Software Maintenance

•**Competitive Advantage:** Regular software maintenance can help to keep the software ahead of the competition by improving functionality, performance, and user experience.

•**Compliance with Regulations:** Software maintenance can help to ensure that the software complies with relevant regulations and standards. This is particularly important in industries such as healthcare, finance, and government, where compliance is critical.

•**Improved Collaboration:** Regular software maintenance can help to improve collaboration between different teams, such as developers, testers, and users. This can lead to better communication and more effective problem-solving.

•**Reduced Downtime:** Software maintenance can help to reduce downtime caused by system failures or errors. This can have a positive impact on business operations and reduce the risk of lost revenue or customers.

•**Improved Scalability:** Regular software maintenance can help to ensure that the software is scalable and can handle increased user demand. This can be particularly important for growing businesses or for software that is used by a large number of users.

Disadvantages of Software Maintenance

•**Cost:** Software maintenance can be time-consuming and expensive, and may require significant resources and expertise.

•**Schedule disruptions:** Maintenance can cause disruptions to the normal schedule and operations of the software, leading to potential downtime and inconvenience.

•**Complexity:** Maintaining and updating complex software systems can be challenging, requiring specialized knowledge and expertise.

•Risk of introducing new bugs: The process of fixing bugs or adding new features can introduce new bugs or problems, making it important to thoroughly test the software after maintenance.

•**User resistance:** Users may resist changes or updates to the software, leading to decreased satisfaction and adoption.

•**Compatibility issues:** Maintenance can sometimes cause compatibility issues with other software or hardware, leading to potential integration problems.

•**Lack of documentation:** Poor documentation or lack of documentation can make software maintenance more difficult and time-consuming, leading to potential errors or delays.

Disadvantages of Software Maintenance

•**Skill gaps:** Maintaining software systems may require specialized skills or expertise that may not be available within the organization, leading to potential outsourcing or increased costs.

•**Inadequate testing:** Inadequate testing or incomplete testing after maintenance can lead to errors, bugs, and potential security vulnerabilities.

•**End-of-life:** Eventually, software systems may reach their end-of-life, making maintenance and updates no longer feasible or cost-effective. This can lead to the need for a complete system replacement, which can be costly and time-consuming.

•**Technical debt:** Over time, software maintenance can lead to technical debt, where the cost of maintaining and updating the software becomes increasingly higher than the cost of developing a new system.

Old Question Papers

Printed Pages : 1 Roll No. | | | | | | | | | | ECS602

B. TECH.

THEORY EXAMINATION (SEM–VI) 2016-17

SOFTWARE ENGINEERING

Time : 3 Hours *Max. Marks : 100*

Note : Be precise in your answer.

SECTION – A

1. Attempt all parts of the following questions: 10 x 2 = 20

(a) What is the software crisis?
(b) Write major software characteristics.
(c) Write the methods of requirements elicitation.
(d) Write the differences between software and software engineering.
(e) What is the difference between Verification and Validation?
(f) How software design can be classify?
(g) Write major software Design Tools.
(h) Write the names of design principles.
(i) Write the differences between Top- downs and bottom-up approaches.
(j) What is software quality?

SECTION – B

2. Attempt any five parts of the following questions: 5 x 10 = 50

(a) What is meant by "Formal Technical Review"? Should it access both programming style as well as correctness of software? Give reasons.
(b) Compare ISO and SEE-CMI model.
(c) What is Risk management? How are project risks different from technical risks?
(d) What is a data flow diagram? Explain rules for drawing good data flow diagrams with the help of a suitable example.
(e) Explain software quality assurance (SQA) with life cycle.
(f) Explain software development life cycle. Discuss various activities during SDLC.
(g) List five desirable characteristics of good SRS document. Discuss the relative advantages of formal and informal requirement specifications.
(h) What are the characteristics of a software process?

SECTION – C

Attempt any two parts of the following questions: 2 x 15 = 30

3. What do you understand by coupling and cohesion? What roles they play in software design? Describe the properties of best coupling and cohesion giving examples of each.
4. What is a Structure Charts? Explain rules for drawing good Structure Charts diagrams with the help of a suitable example.
5. Define the following:
(i) Water fall Model (ii) Spiral Model

Old Question Papers

Printed Pages: 02
Paper Id: 1 1 0 6 0 2

Sub Code: NCS602/ECS602
Roll No.

B.TECH
(SEM VI) THEORY EXAMINATION 2017-18
SOFTWARE ENGINEERING

Time: 3 Hours *Total Marks: 100*

Note: 1. Attempt all Sections. If require any missing data; then choose suitably.

SECTION A

1. Attempt *all* questions in brief. 2 x10 = 20

a. What do you understand by software crisis?
b. What are different software quality attributes?
c. Write the difference between verification and validation.
d. What is Decision Tree?
e. Write principles of Software Design.
f. What is Pseudo Code? How it differs from Algorithm?
g. Explain Code Inspection.
h. What are stub and driver?
i. Define CASE tools.
j. What is Adaptive and Corrective Maintenance?

SECTION B

2. Attempt any *three* of the following: 10 x 3 = 30

a. Explain Spiral Model? Also write it's advantages and disadvantages.
b. Explain CMM Model. Compare ISO and CMM.
c. Explain different methods of verification in detail.
d. What is Structure Chart? Explain different basic blocks used to build structure chart with suitable example.
e. What is cost analysis in context of software? Explain COCOMO Model.

SECTION C

3. Attempt any *one/two* part of the following: 10 x 1 = 10

(a) Explain different phases of SDLC.
(b) Explain Iterative Enhancement Model. Write its advantages and disadvantages.

4. Attempt any *one/two* part of the following: 10 x 1 = 10

(a) What do you understand by DFD? Explain basic blocks, which are used to build DFD with suitable example.
(b) What is SRS? Explain characteristics of a good SRS.

5. Attempt any *one/two* part of the following: 10 x 1 = 10

(a) What is objective of software design? Explain different approaches for software design.
(b) What is Cyclomatic complexity? Write all methods, which are used to calculate the Cyclomatic complexity of a control, flow graph.

Old Question Papers

6. **Attempt any *one/two* part of the following:** **10 x 1 = 10**

(a) What is Regression Testing? Explain the process of test case prioritization in regression testing.

(b) What is Integration Testing? Explain different approaches used for integration testing.

7. **Attempt any *one/two* part of the following:** **10 x 1 = 10**

(a) Explain various software configuration management activities.

(b) Explain Software Risks Analysis and Management process.

Thank You