

Software As An Evolutionary Entity (CO5)

- **Software Maintenance** includes **error correction, enhancements of capabilities, deletion of obsolete capabilities & optimization**. As changes cannot be avoided, we should develop mechanism for evaluating, controlling & making modifications. Hence any work done to change the s/w after its operation is considered to be a maintenance work. The term “evolution” has been used with reference to s/w since 1960’s to signify the growth dynamics of s/w.

Software as an Evolutionary Entity

- Software Maintenance is a very broad activity that includes error corrections, enhancements of capabilities, deletion of obsolete capabilities, and optimization. **As per IEEE**, it is a modification of s/w product after delivery to **correct faults**, to **improve performance** or other attributes or to adapt the product to a modified environment. **As per ISO**, it is a set of activities performed when s/w undergoes modifications to code & associated documentation due to a problem or the need for improvement or adaptation.

Need For Software Maintenance (CO5)

Maintenance is needed for:-

- Correct errors.
- Change in user requirement with time.
- Changing hardware/software environment.
- To improve system efficiency
- To optimize the code to run faster
- To modify the components
- To eliminate any unwanted side effects.
- Thus, the maintenance is needed to ensure that the system continues to satisfy user requirements.

Objective of Software Maintenance

AIM of Software Maintenance

To correct errors

- To enhance the s/w by changing its functions.
- To update the s/w.
- To adapt the s/w to cope with changes in the environment.

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.

Categories Of Software Maintenance

- ❖ **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.

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.

Other types of Maintenance

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.

% Effort

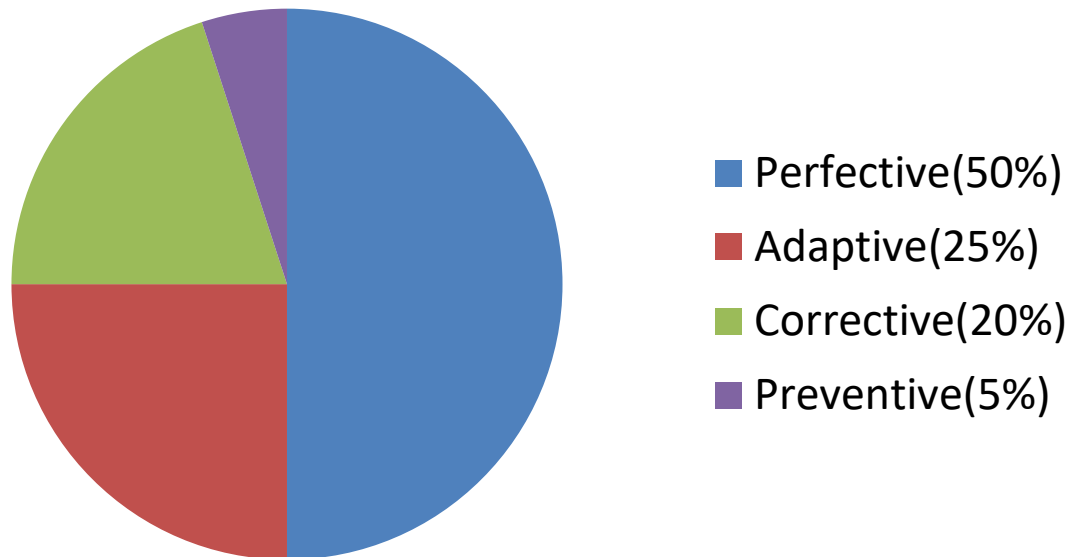


Fig: Distribution of maintenance effort

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.

Cost of maintenance (CO5)

Estimation of maintenance costs

Phase	Ratio
Analysis	1
Design	10
Implementation	100

Defect repair ratio

Belady and Lehman Model

$$M = P + Ke^{(c-d)}$$

where

- M : Total effort expended
- P : Productive effort that involves analysis, design, coding, testing and evaluation.
- K : An empirically determined constant.
- c : Complexity measure due to lack of good design and documentation.
- d : Degree to which maintenance team is familiar with the software.

Boehm Model

Boehm used a quantity called Annual Change Traffic (ACT).

“The fraction of a software product’s source instructions which undergo change during a year either through addition, deletion or modification”.

$$ATC = \frac{KLOC_{added} + KLOC_{deleted}}{KLOC_{total}}$$

$$AME = ACT \times SDE$$

Where, SDE : Software development effort in person months

ACT : Annual change Traffic

EAF : Effort Adjustment Factor

$$AME = ACT * SDE * EAF$$

Example –

The development effort for a software project is 500 person months. The empirically determined constant (K) is 0.3. The complexity of the code is quite high and is equal to 8. Calculate the total effort expended (M) if

- (i) maintenance team has good level of understanding of the project ($d=0.9$)
- (ii) maintenance team has poor understanding of the project ($d=0.1$)

Example

Solution

Development effort (P) = 500 PM

$$K = 0.3$$

$$C = 8$$

(i) maintenance team has good level of understanding of the project (d=0.9)

$$\begin{aligned} M &= P + Ke^{(c-d)} \\ &= 500 + 0.3e^{(8-0.9)} \\ &= 500 + 363.59 = 863.59 \text{ PM} \end{aligned}$$

(ii) maintenance team has poor understanding of the project (d=0.1)

$$\begin{aligned} M &= P + Ke^{(c-d)} \\ &= 500 + 0.3e^{(8-0.1)} \\ &= 500 + 809.18 = 1309.18 \text{ PM} \end{aligned}$$

Example – 9.2

Annual Change Traffic (ACT) for a software system is 15% per year. The development effort is 600 PMs. Compute estimate for Annual Maintenance Effort (AME). If life time of the project is 10 years, what is the total effort of the project ?

Example

Solution

The development effort = 600 PM

Annual Change Traffic (ACT) = 15%

Total duration for which effort is to be calculated = 10 years

The maintenance effort is a fraction of development effort and is assumed to be constant.

$$\begin{aligned} \text{AME} &= \text{ACT} \times \text{SDE} \\ &= 0.15 \times 600 = 90 \text{ PM} \end{aligned}$$

$$\text{Maintenance effort for 10 years} = 10 \times 90 = 900 \text{ PM}$$

$$\text{Total effort} = 600 + 900 = 1500 \text{ PM}$$

Reverse Engineering

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

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

Scope and Tasks

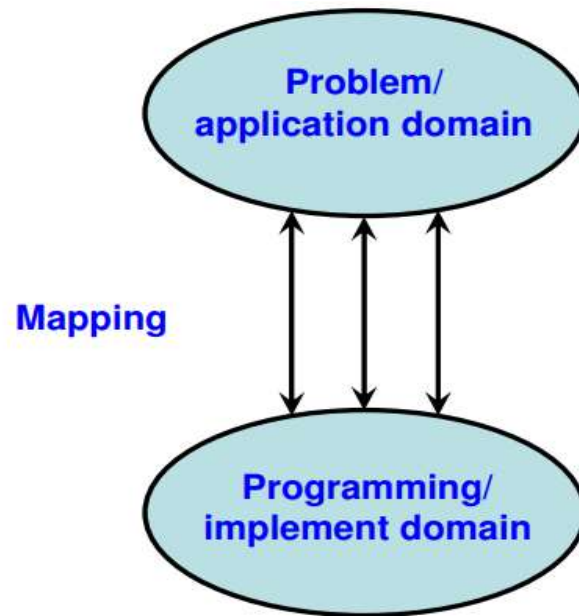
The areas where reverse engineering is applicable include (but not limited to):

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

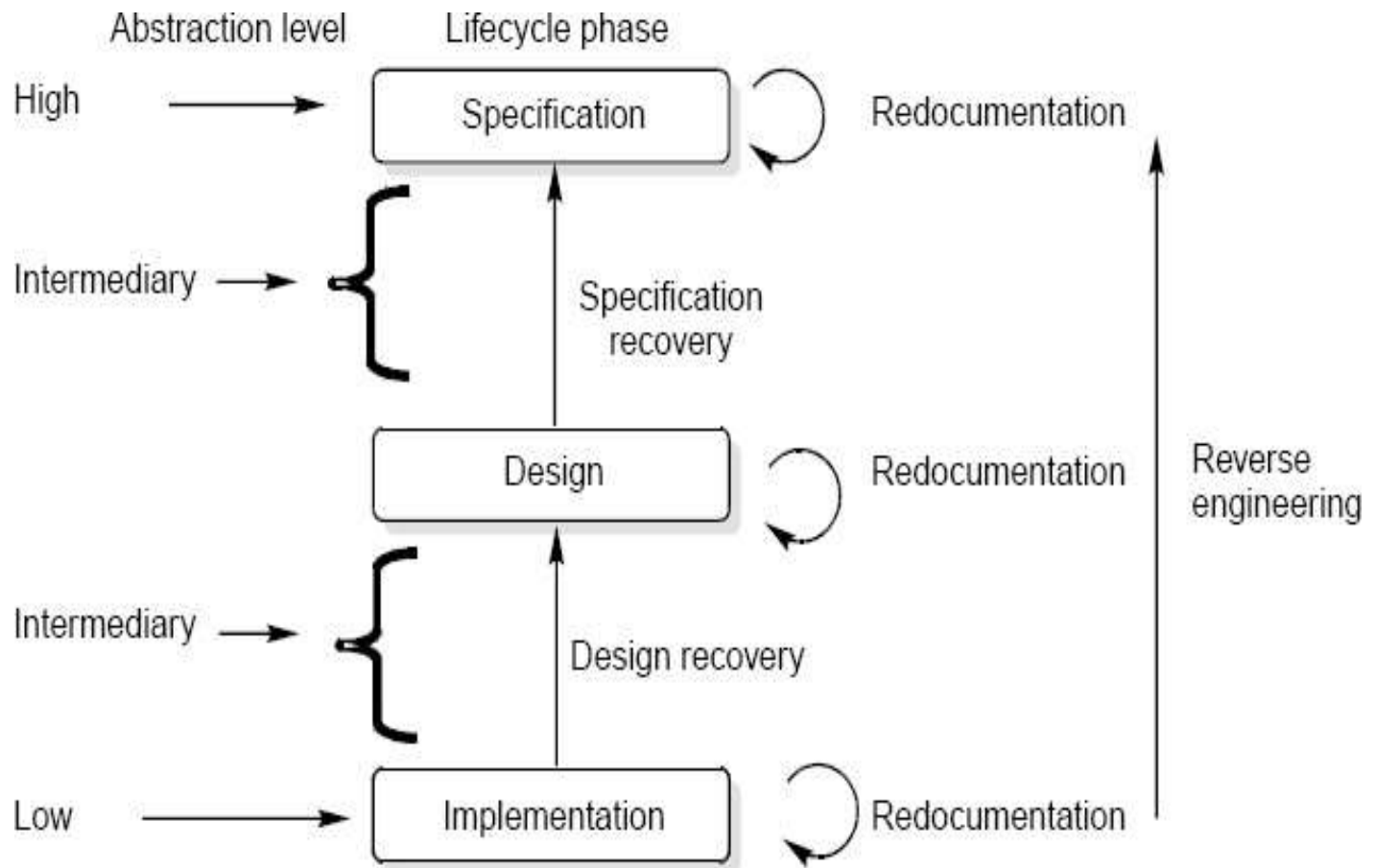
Reverse Engineering

Reverse Engineering encompasses a wide array of tasks related to understanding and modifying software system. This array of tasks can be broken into a number of classes.

- Mapping between application and program domains



Reverse Engineering



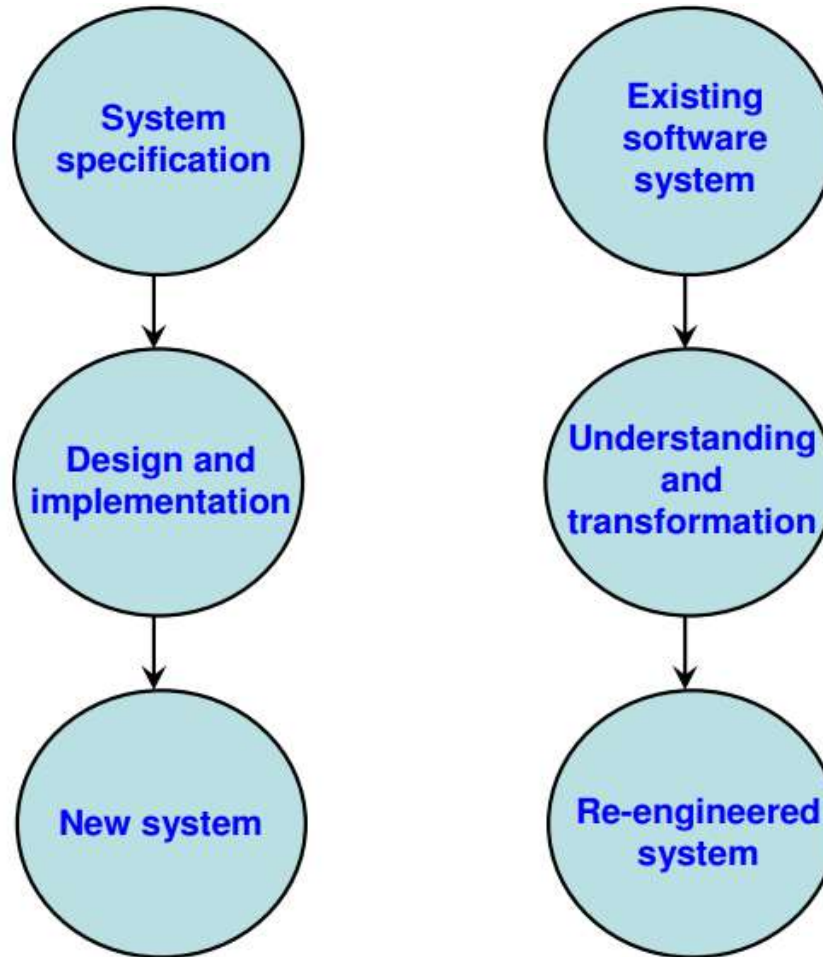
Reverse Engineering

- In it, the old code is analyzed (abstracted) to extract the module specifications.
 - The module specifications are then analyzed to produce the design.
 - The design is analyzed (abstracted) to produce the original requirements specification.
 - The change requests are then applied to this requirements specification to arrive at the new requirements specification.
 - At the design, module specification, and coding a substantial reuse is made from the reverse engineered products.
- **Advantage**
- it produces a more structured design compared to what the original product had,
 - It produces good documentation, and very often results in increased efficiency.

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

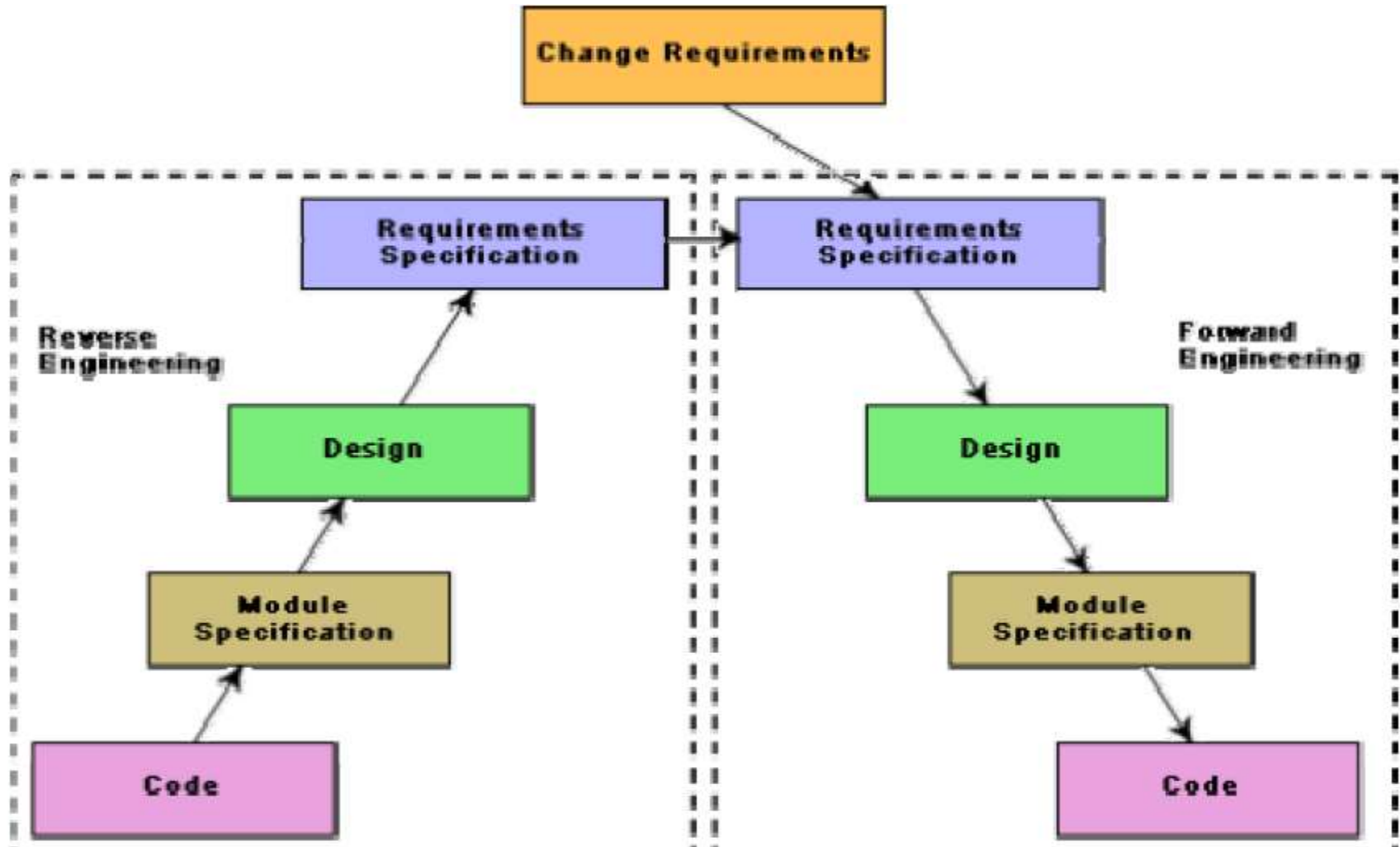
Objective of -Engineering

- 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



The following suggestions may be useful for the modification of the legacy code:

- ✓ Study code well before attempting changes
- ✓ Concentrate on overall control flow and not coding
- ✓ Heavily comment internal code
- ✓ Create Cross References
- ✓ Build Symbol tables
- ✓ Use own variables, constants and declarations to localize the effect
- ✓ Keep detailed maintenance document
- ✓ Use modern design techniques

Software Configuration Management (CO5)

Configuration Management

- SCM is used to controlled 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 parts 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)

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

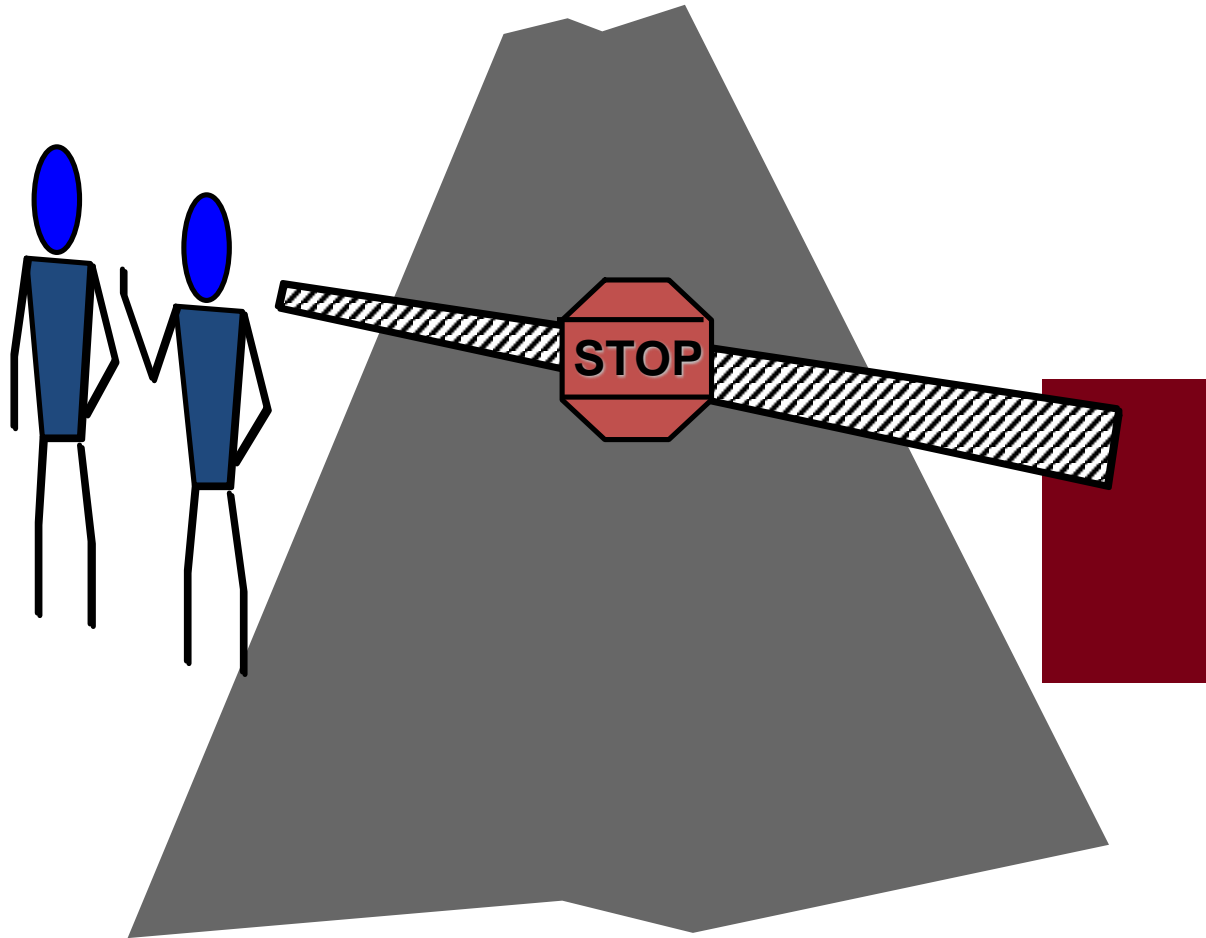
- Name of each source code component, including the variations and revisions
- The versions of the various compilers and linkers used
- The name of the software staff who constructed the component
- The date and the time at which it was constructed

Change Control Process (CO5)

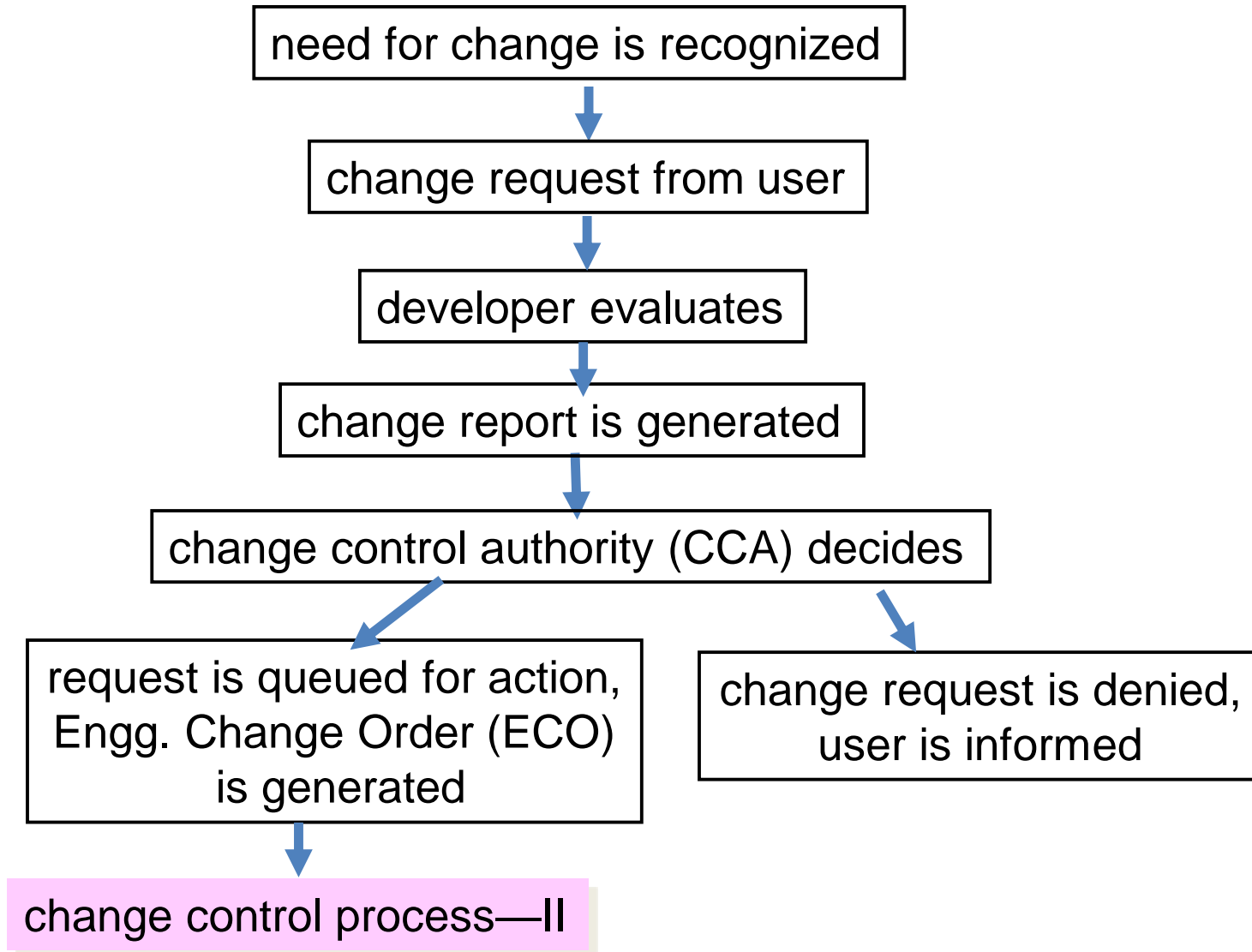
Change control process comes into effect when the software and associated documentation are delivered to configuration management change request form, which should record the recommendations regarding the change.

- Change control 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?

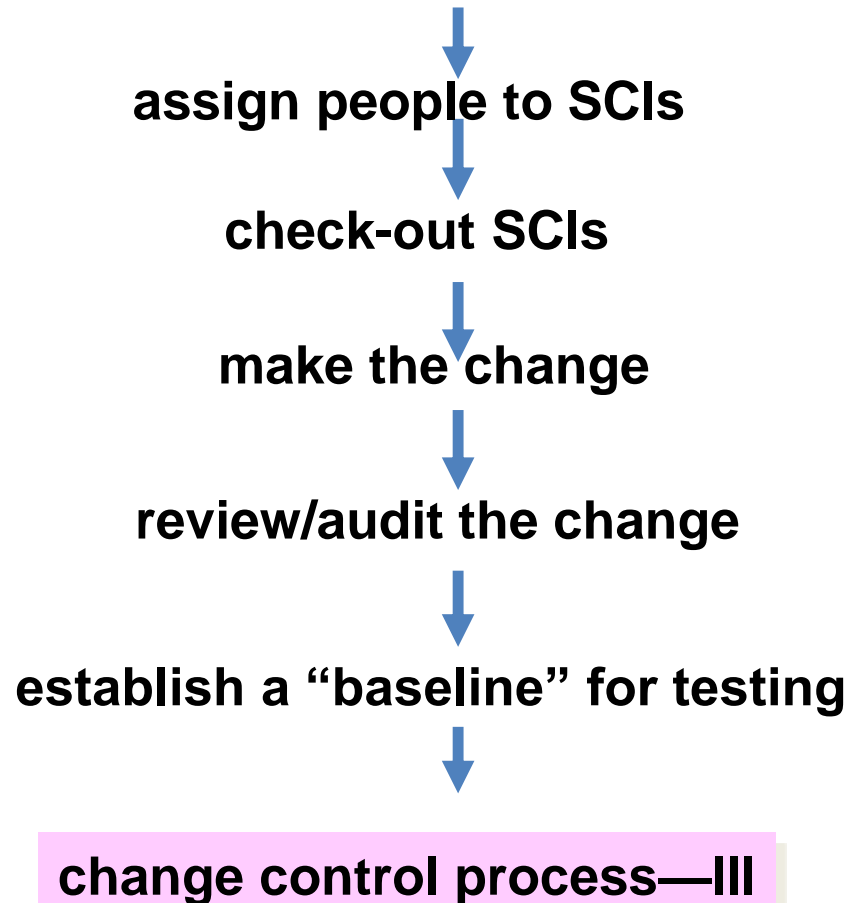
Change control



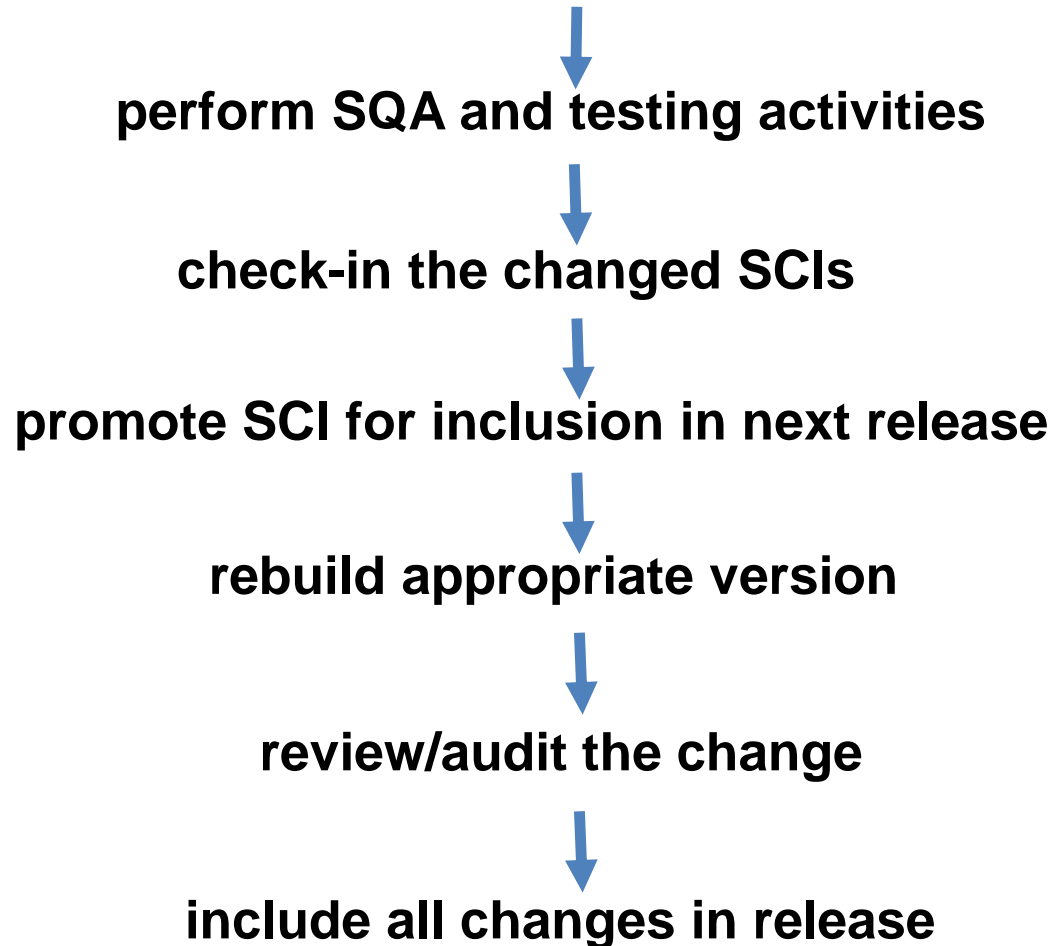
Change Control Process—I



Change Control Process—II



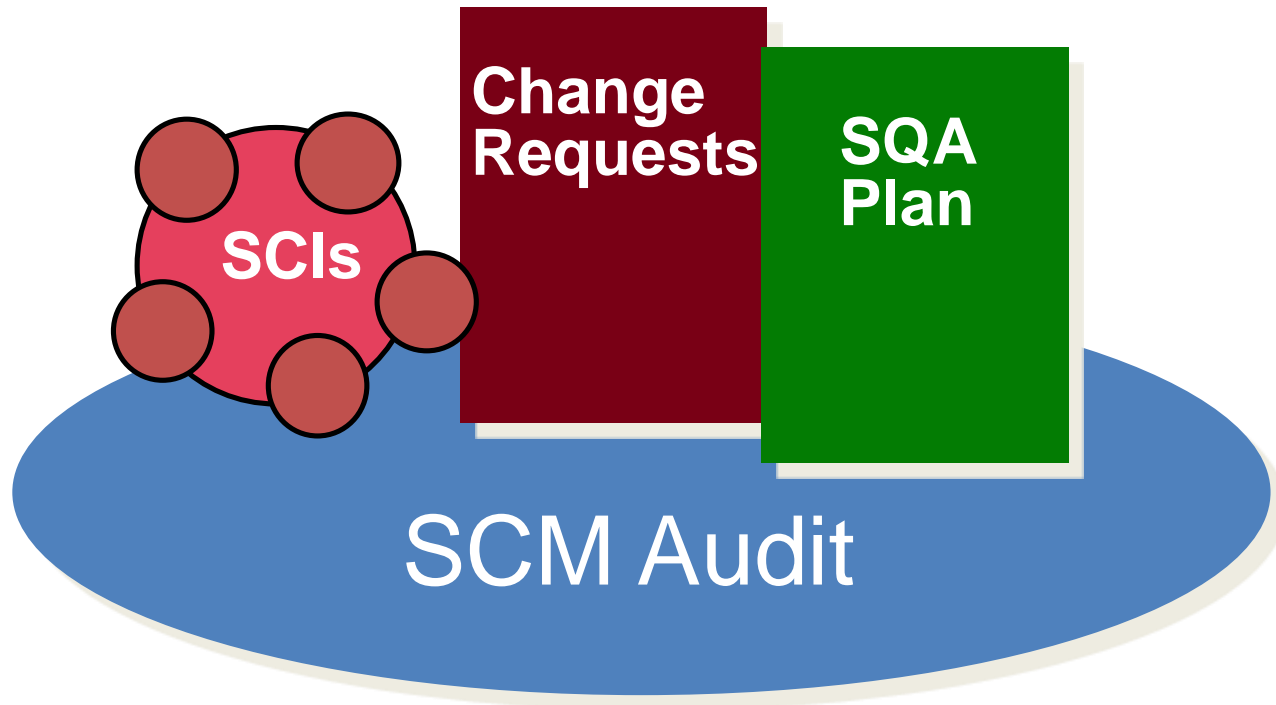
Change Control Process—III



- 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

Configuration audit

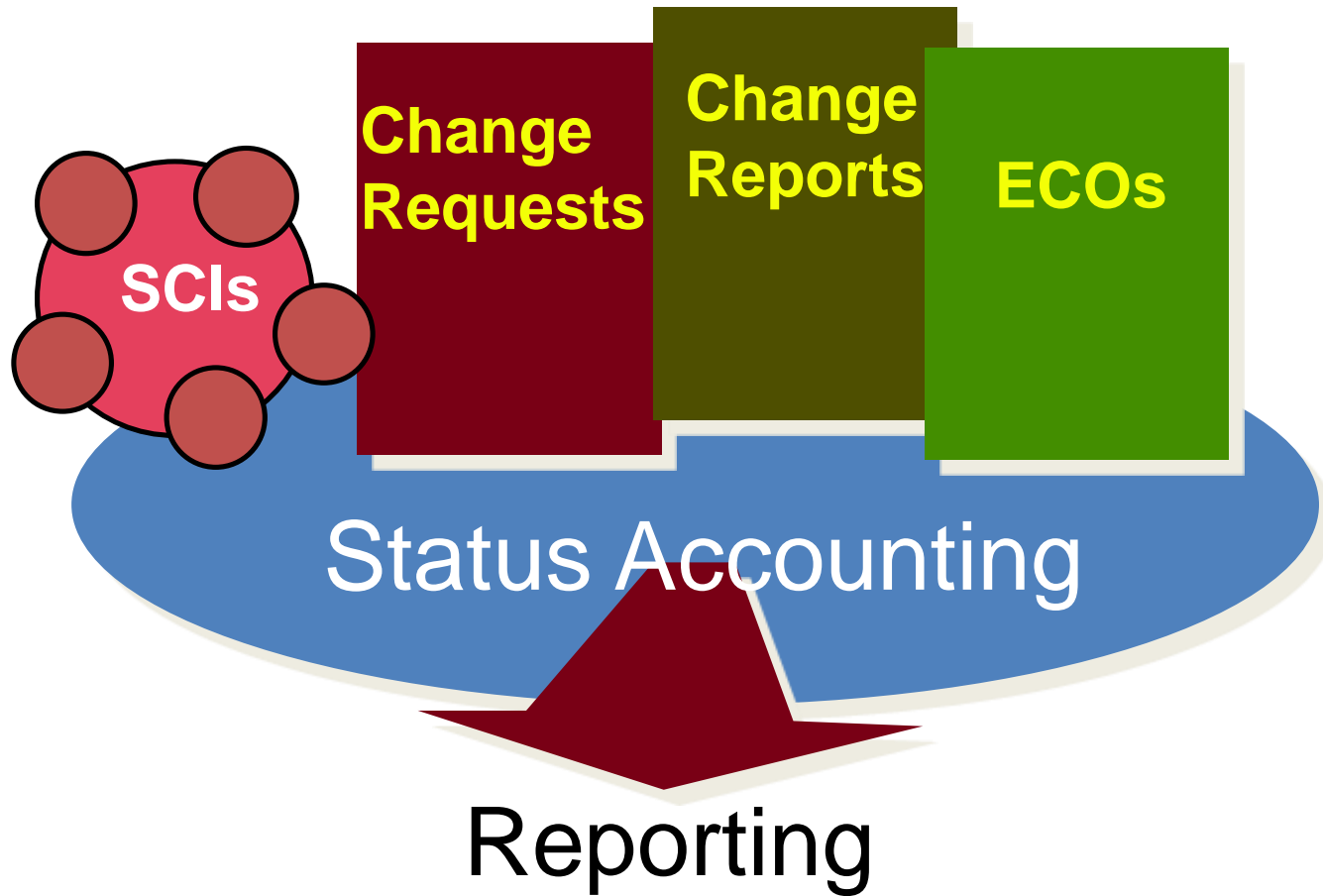
Auditing



- The audit asks and answers the following questions
 - Has the change specified in the ECO been made? Have any additional modifications been incorporated?
 - Has the change been 'highlighted' in the SCI? Have the change date and change author been specified? Do the attributes of the configuration object reflect the change?
 - Have SCM procedures for noting the change, recording it and reporting it been followed?
 - Have all related SCIs been properly updated?

Configuration Status Reporting (CSR)

- SCM task that answers these questions
 - What happened?
 - Who did it?
 - When did it happen?
 - What else will be affected?
- When many people are involved, it is likely that ‘the left hand not knowing what the right hand is doing’ syndrome will occur
- CSR helps to eliminate this problem by improving communication among all the people involved



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

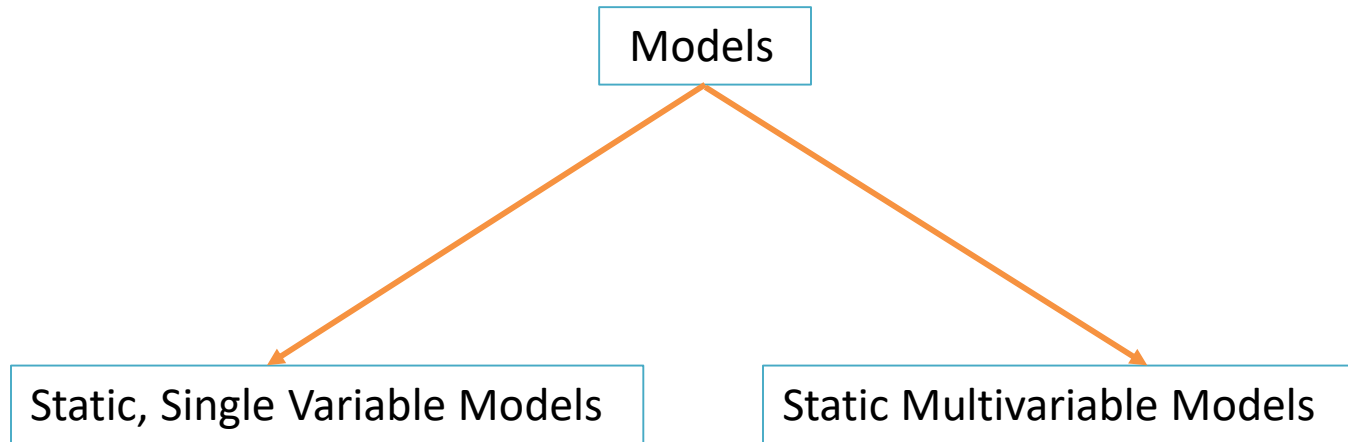
List of CASE Tool

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 find 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

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)

$$c = aL^b$$

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.

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

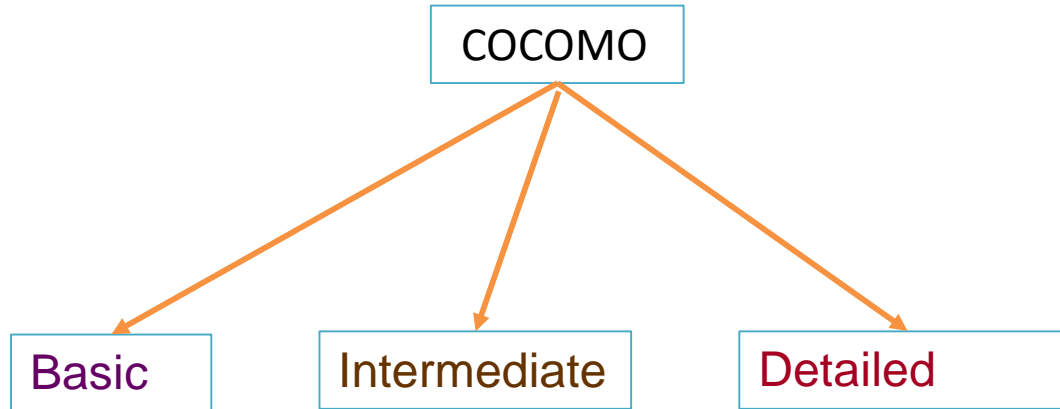
$$D = 4.1 L^{0.36}$$

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

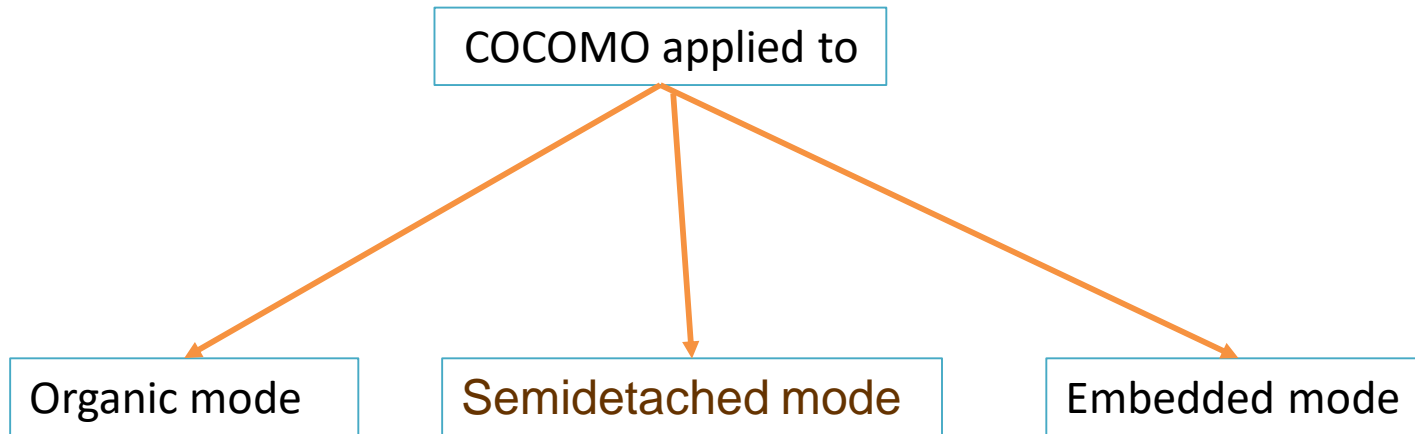
$$I = \sum_{i=1}^{29} W_i X_i$$

COCOMO (CO5)

Constructive Cost model



Constructive Cost model



Comparison of three COCOMO modes

<i>Mode</i>	<i>Project size</i>	<i>Nature of Project</i>	<i>Innovation</i>	<i>Deadline of the project</i>	<i>Development Environment</i>
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 COCOMO model takes the form

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

$$D = c_b (E)$$

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:

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

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

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

Basic COCOMO coefficients

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

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	--

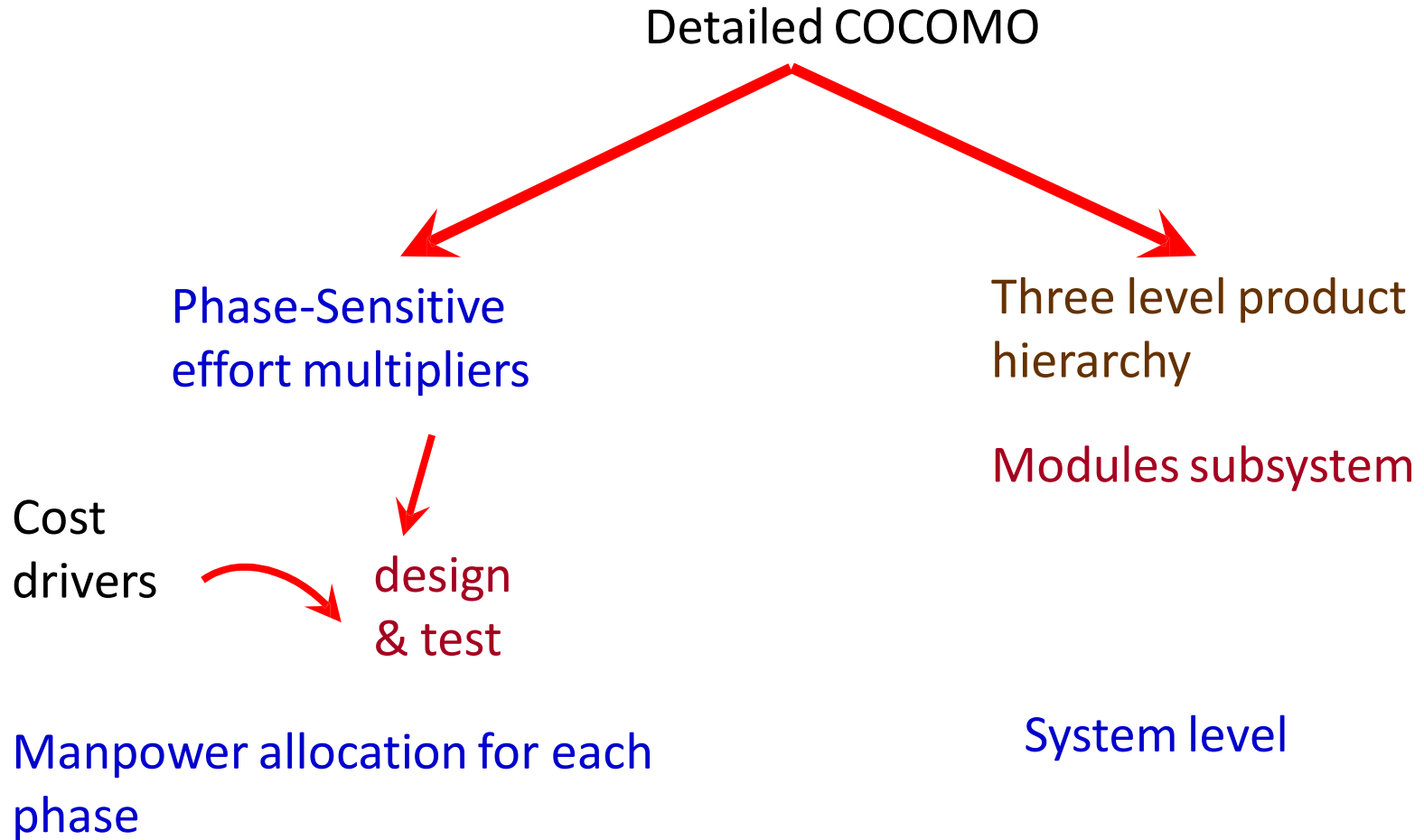
Intermediate COCOMO equations

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

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

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.

$$A = 0.4 \text{ DD} + 0.3 \text{ C} + 0.3 \text{ I}$$

The size equivalent is obtained by

$$S (\text{equivalent}) = (S \times A) / 100$$

$$E_p = \mu_p E$$

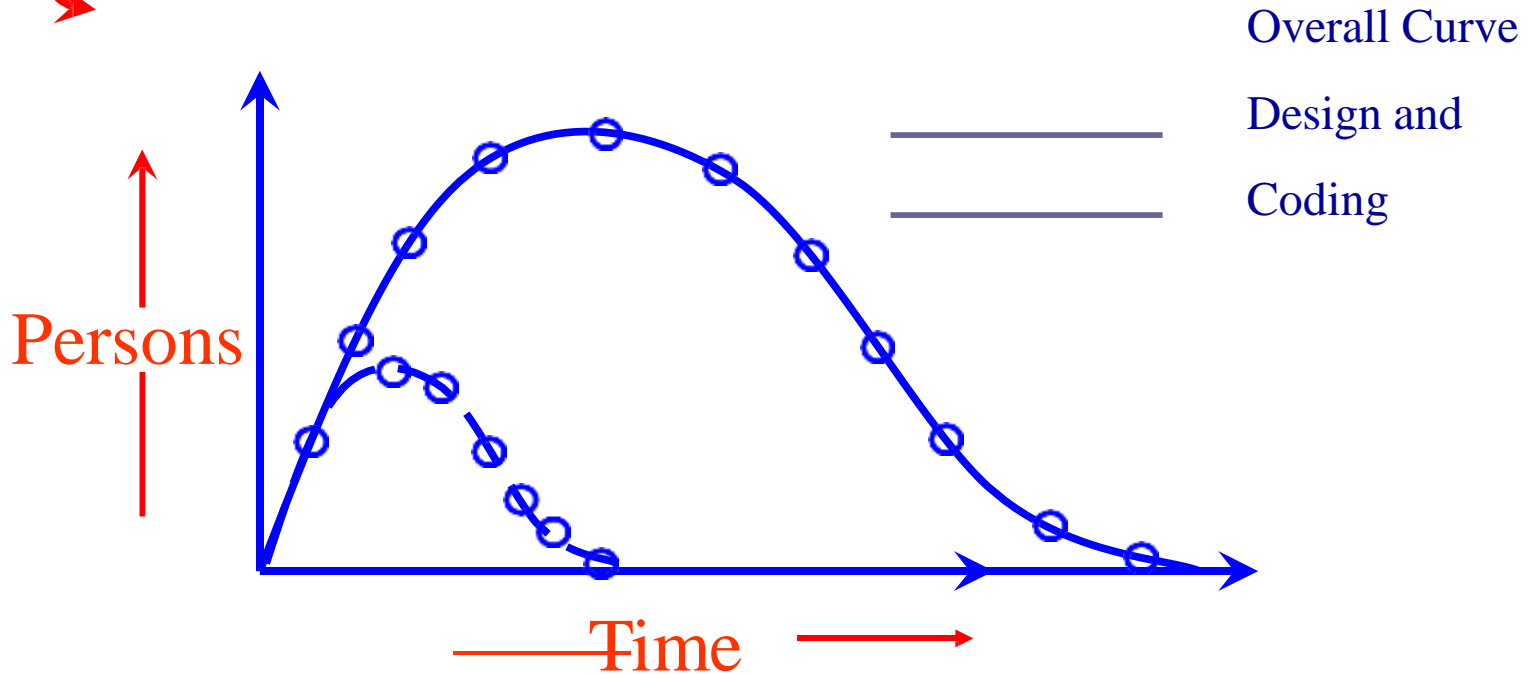
$$D_p = \tau_p D$$

Resource Allocation Model (CO5)

Norden of IBM

Rayleigh curve

Model for a range of hardware development projects.



The Rayleigh manpower loading curve

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

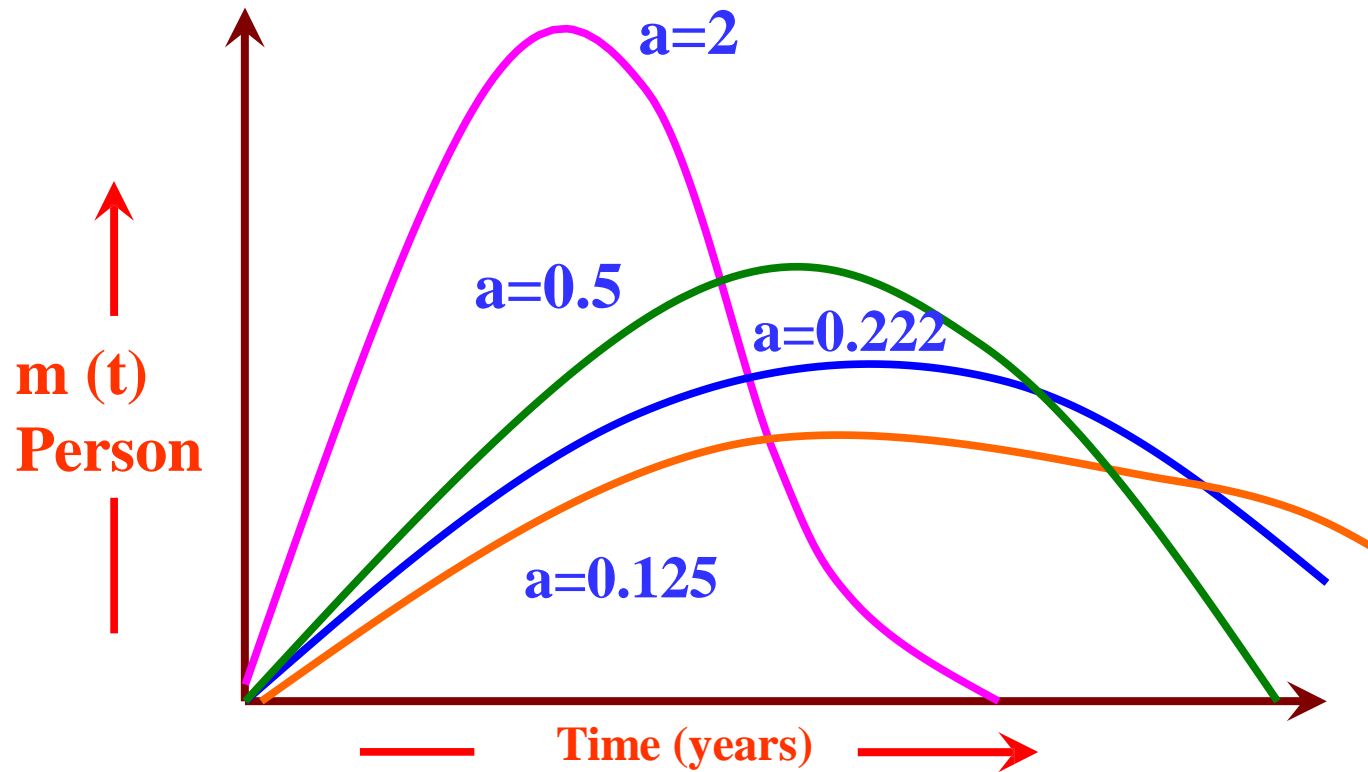


Fig: Influence of parameter 'a' on the manpower distribution

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

2. Requirement issues

Uncertain requirements



Wrong product

or

Right product badly

Either situation results in unpleasant surprises and unhappy customers.

2. Requirement issues

Uncertain requirements



Wrong product

or

Right product badly

Either situation results in unpleasant surprises and unhappy customers.

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

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

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

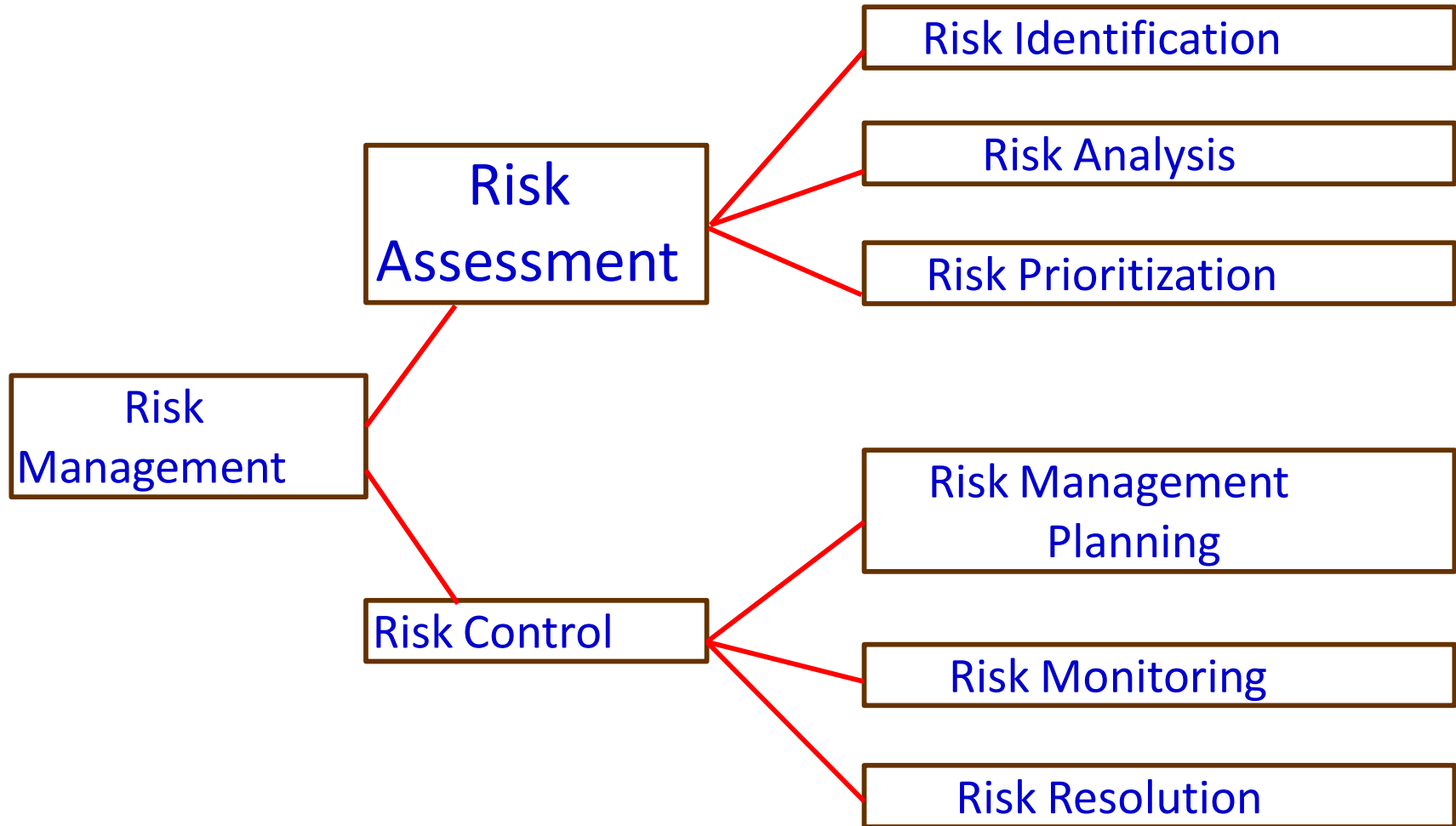


Fig. Risk Management Activities