UNIT-5

SOFTWARE MAINTENANCE

INTRODUCTION

- Software maintenance is the last stage of s/w life cycle .
- After the product has been released, the maintenance phase keeps the s/w up to date with environment changes & changing user requirements.
- It consumes about 40-70% of the cost of the entire life cycle.
- Maintenance can only happen efficiently if the earlier phases are done properly.

SOFTWARE AS AN EVOLUTIONARY ENTITY

- 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.

WHAT IS SOFTWARE MAINTENANCE

- 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 MAINTENANCE

- Software 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.

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

- There are four types of software maintenance:

1. **Corrective maintenance :**
   This refer to modifications initiated by defects in the software.

2. **Adaptive maintenance:**
   It includes modifying the software to match changes in the ever changing environment.

3. **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.

4. 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.

SOFTWARE MAINTENANCE

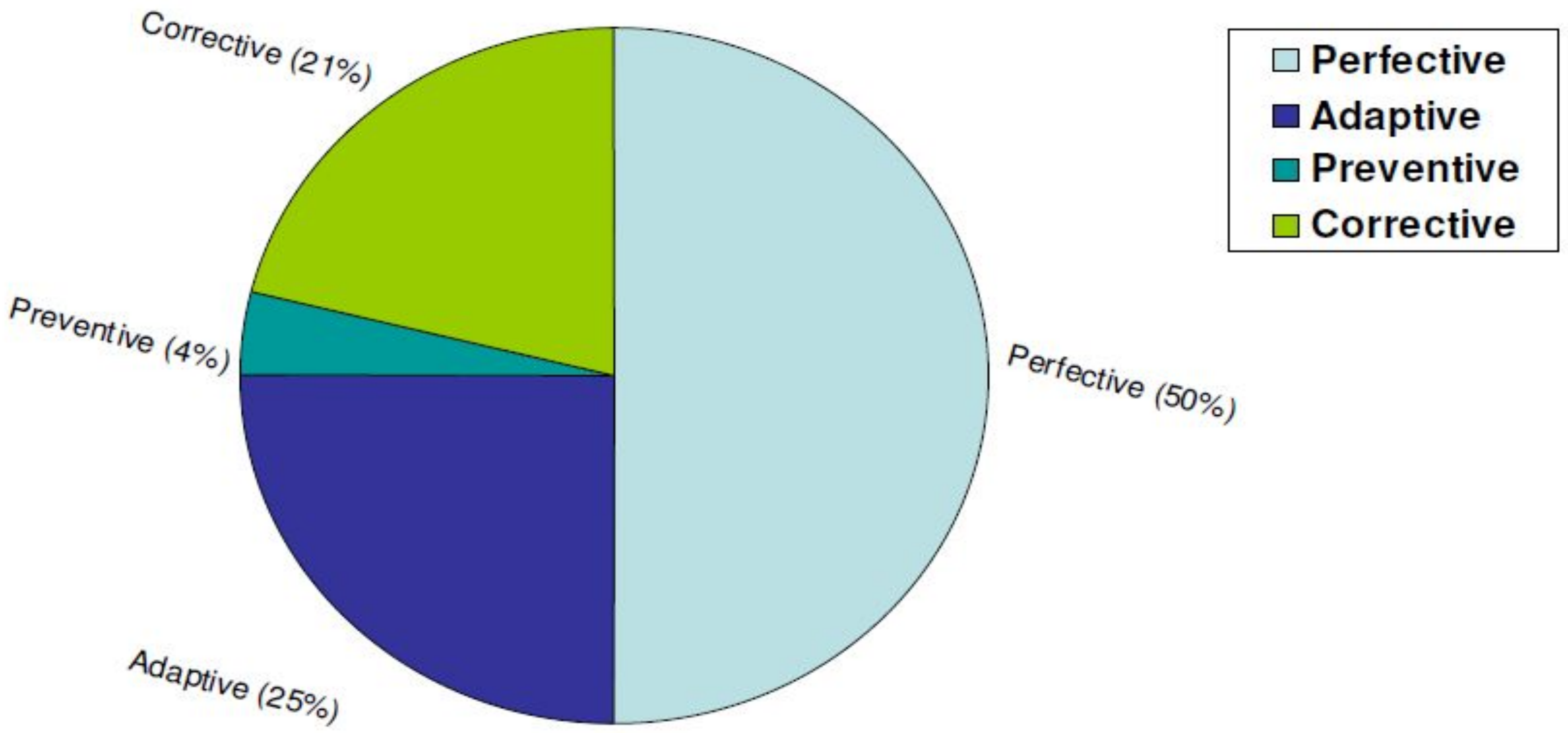


Fig: Distribution of maintenance effort

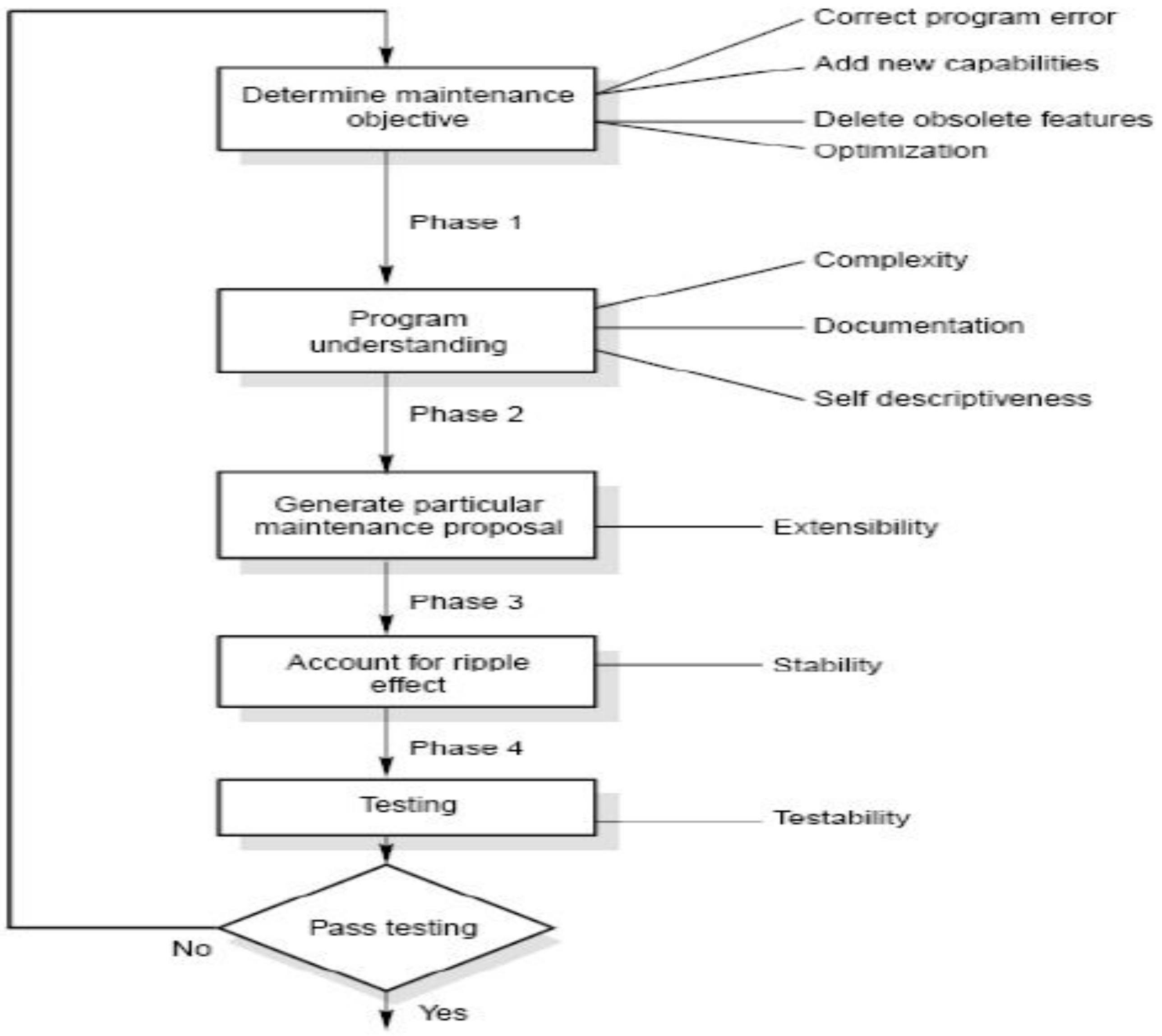
Problems during Maintenance

- Often the program is written by another person or group of persons.
- Often the program is changed by person who did not understand it clearly.
- Program listings are not structured.
- High staff turnover.
- Information gap.
- Systems are not designed for change.

Potential Solutions to Maintenance Problems

- Budget and effort reallocation
- Complete replacement of the system
- Maintenance of existing system

The 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.

- **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.

  How easy is to maintain a program depends on how easy is to understand it.

COST OF MAINTENANCE

- To estimate maintenance cost, two models have been proposed. They are:

1. Belady and Lehman model
2. Boehm model

Belady and Lehman Model

This model indicates that the **effort and cost can increase exponentially if poor Software development approach is used** & the person or group that used the approach is no longer available to perform maintenance. The basic equation is given by:

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

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.

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)

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)

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

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

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

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”.

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

AME = ACT x SDE

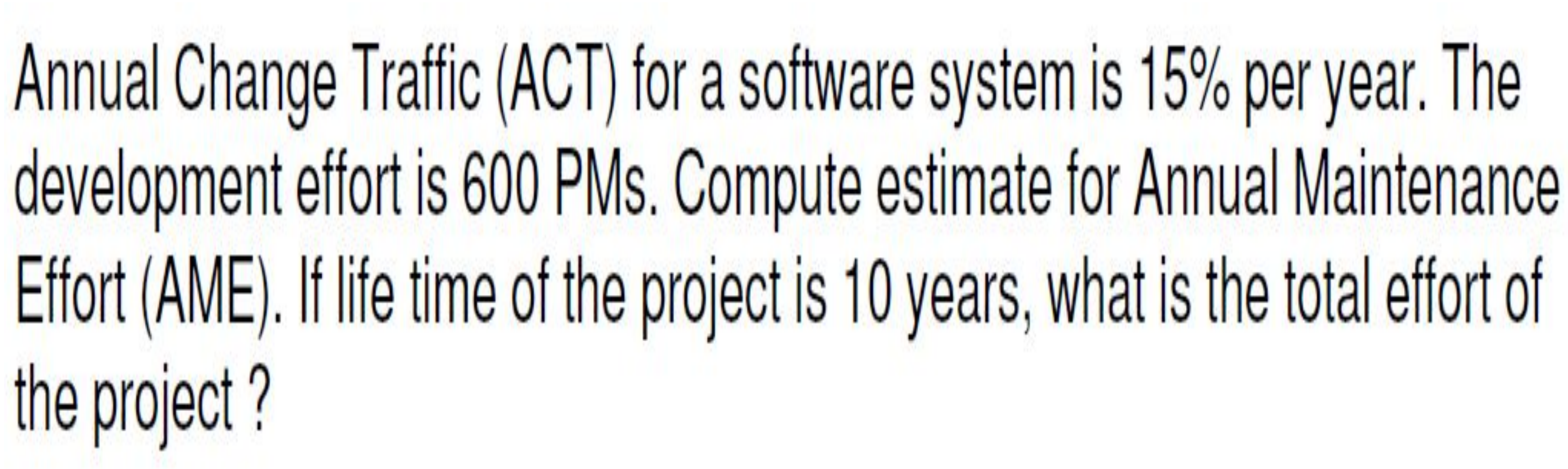
Where, **SDE :** Software development effort in person months

ACT : Annual change Traffic

EAF : Effort Adjustment Factor

AME = ACT * SDE * EAF

Example



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 ?

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.

AME = ACT x SDE

= 0.15 x 600 = 90 PM

Maintenance effort for 10 years	= 10 x 90 = 90 PM
Total effort	= 600 + 900 = 1500 PM

Introduction to Re-engineering

- Re-engineering means to **re-implement systems to make more maintainable.**
- In this, the functionality & architecture of the system remains the same but it includes redocumenting, organizing, modifying & updating the system.
- When re-engineering principles are applied to business process then it is called **Business Process Reengineering (BPR).**

Steps in Re-engineering

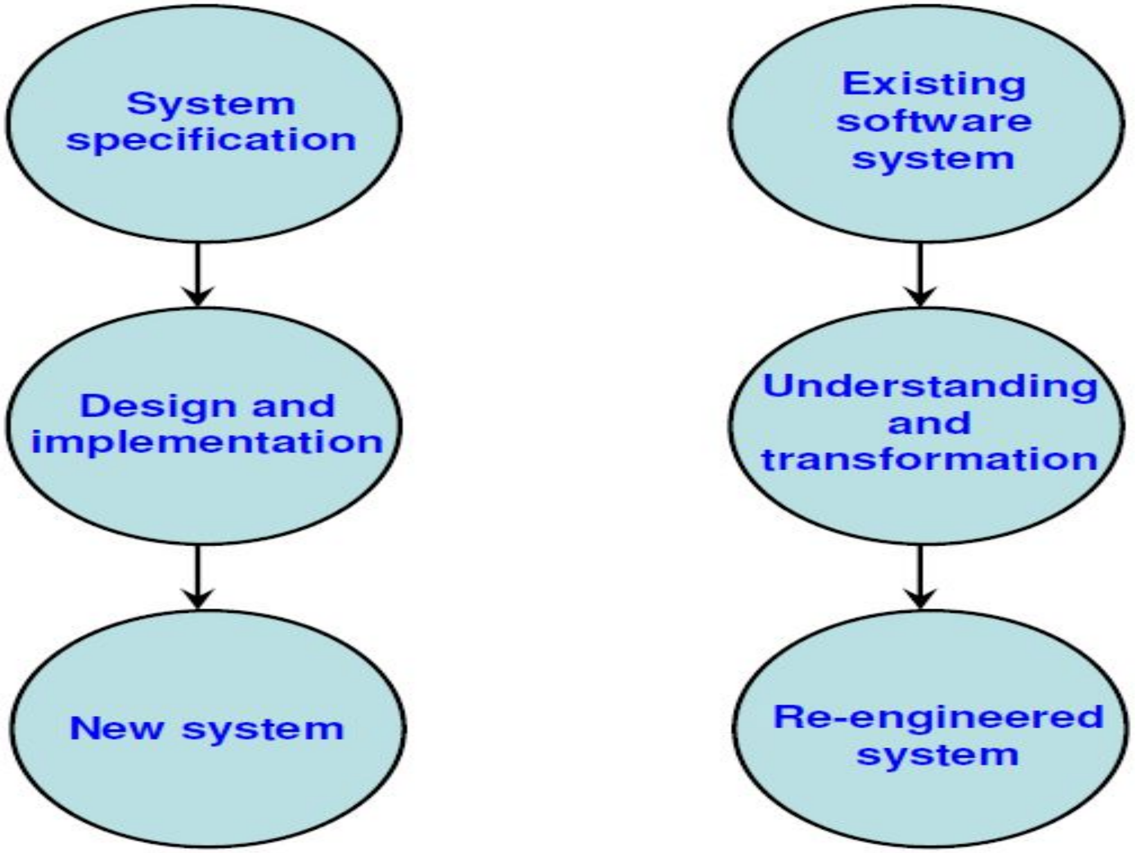
- The steps involved in re-engineering are-
  - Goal setting
  - Critical analysis of existing scenario such as process, task, design, methods etc.
  - Identifying the problems & solving them by new innovative thinking.

  In other words, it uses new technology to gain a significant breakthrough improvement.

Software RE-Engineering

- The principles of re-engineering when applied to s/w development processes , it is called s/w reengineering.
- Acc to **IBM user group guide, “**the process of modifying the internal mechanism of a system or program or the data structures of a system or program without changing its functionality”.

- 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



Comparison of new software development with 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

Source Code Translation

1. **Hardware platform update:** The organization may wish to change its standard hardware platform. Compilers for the original language may not be available on the new platform.

2. **Staff Skill Shortages:** There may be lack of trained maintenance staff for the original language. This is a particular problem where programs were written in some non standard language that has now gone out of general use.

3. **Organizational policy changes:** An organization may decide to standardize on a particular language to minimize its support software costs. Maintaining many versions of old compilers can be very expensive.

Program Restructuring

1. **Control flow driven restructuring:** This involves the imposition of a clear control structure within the source code and can be either inter modular or intra modular in nature.

2. **Efficiency driven restructuring:** This involves restructuring a **function or algorithm** to make it more efficient. A simple example is the replacement of an IF-THEN-ELSE-IF-ELSE construct with a CASE construct.

```
IF Score > = 75 THEN Grade: = 'A'
ELSE IF Score > = 60 THEN Grade: = 'B'
ELSE IF Score > = 50 THEN Grade: = 'C'
ELSE IF Score > = 40 THEN Grade: = 'D'
ELSE IF Grade = 'F'
END
```

(a)

```
CASE Score of
75, 100: Grade: = 'A'
60, 74: Grade: = 'B';
50, 59: Grade: = 'C';
40, 49: Grade: = 'D';
ELSE Grade: = 'F'
END
```

(b)

Restructuring a program

3. **Adaption driven restructuring:** This involves **changing the coding style** in order to adapt the program to a new programming language or new operating environment, for instance changing an imperative program in PASCAL into a functional program in LISP.

Advantages of Re-engineering

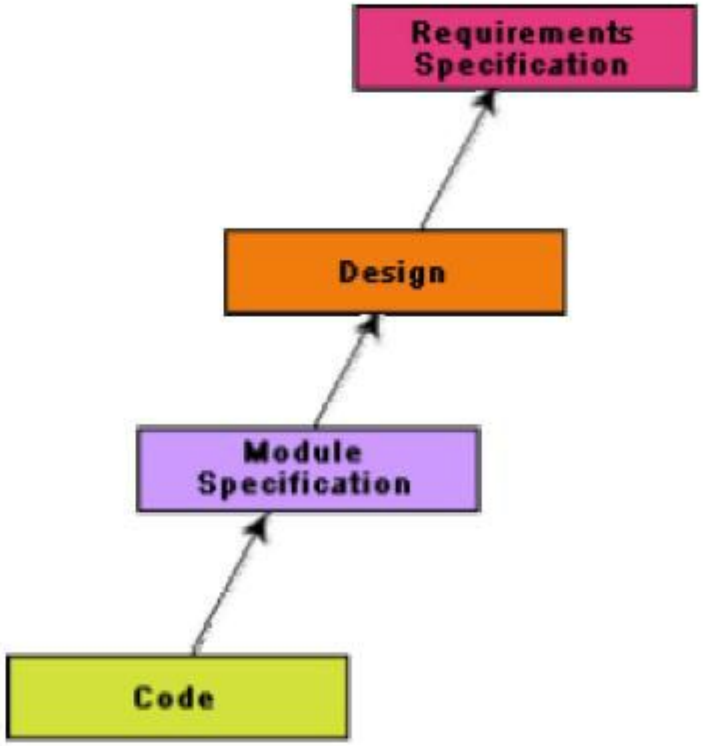
- Lower costs
- Lower risks
- Better use of existing staff
- Incremental development

Disadvantages of Re-engineering

- Major architectural changes or radical reorganizing of the system data management has to be done manually.
- Re-engineered system is not likely to be as maintainable as a new system developed using modern s/w engineering methods.

Reverse Engineering

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



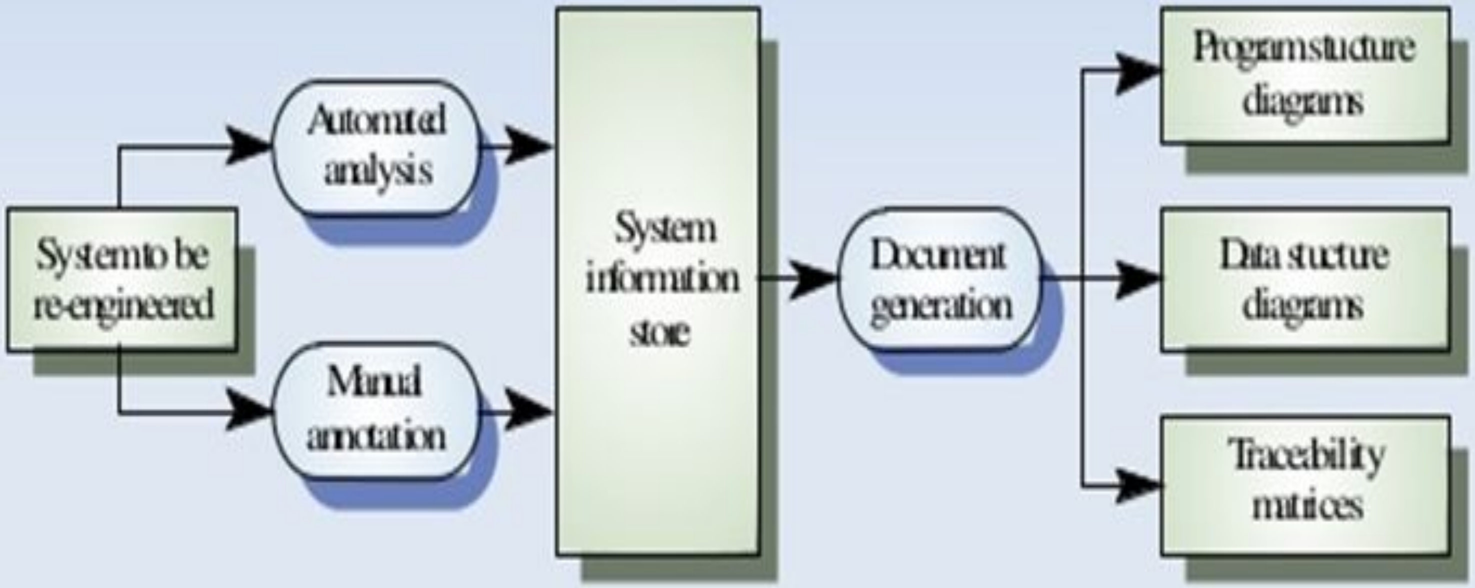
- Software reverse engineering is the **process of recovering the design and the requirements specification** of a product from an analysis of its code.
- The **purpose** of reverse engineering is to **facilitate maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.**

Scope and Tasks

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

1. Program comprehension
2. Redocumentation 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. Understanding high level system description

The Reverse Engineering Process



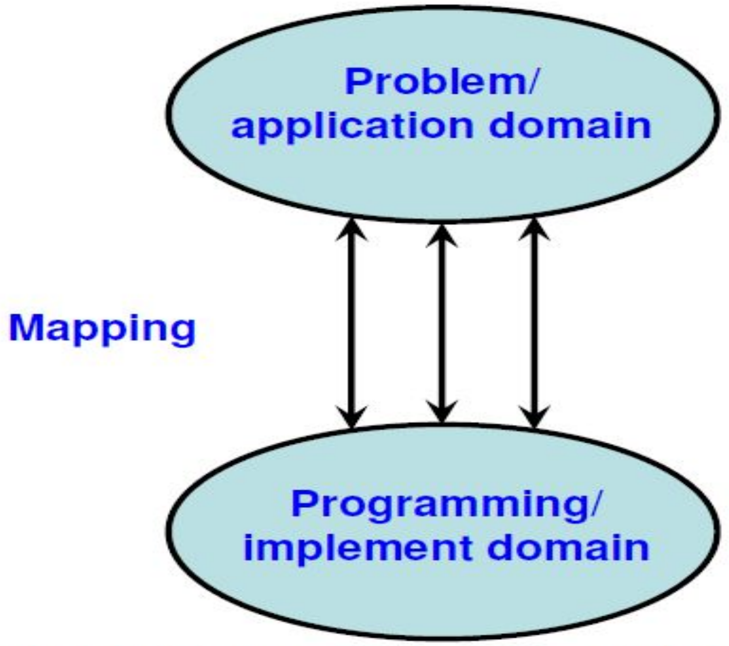
The reverse engineering process is shown in fig. The process starts with an analysis phase. During this phase, the system is analyzed using automated tools to discover its structure. They add information to this, which they have collected by understanding the system. This information is maintained as a directed graph that is linked to the program source code.

Information store browsers are used to compare the graph structure & the code & to annotate the graph with extra information. Documents of various types such as program & data structure diagrams & traceability matrices. Traceability matrices show where entities in the system are defined and referenced.

Reverse Engineering tasks

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. Few of these are discussed below:

1. Mapping between application and program domains



Mapping between application and domains program

2.Mapping between concrete and abstract levels

3. Rediscovering high level structures

4. Finding missing links between program syntax and semantics

5. To extract reusable component

Levels of Reverse Engineering

- Reverse Engineers detect low level implementation constructs and replace them with their high level counterparts.
- The process eventually results in an incremental formation of an overall architecture of the program.

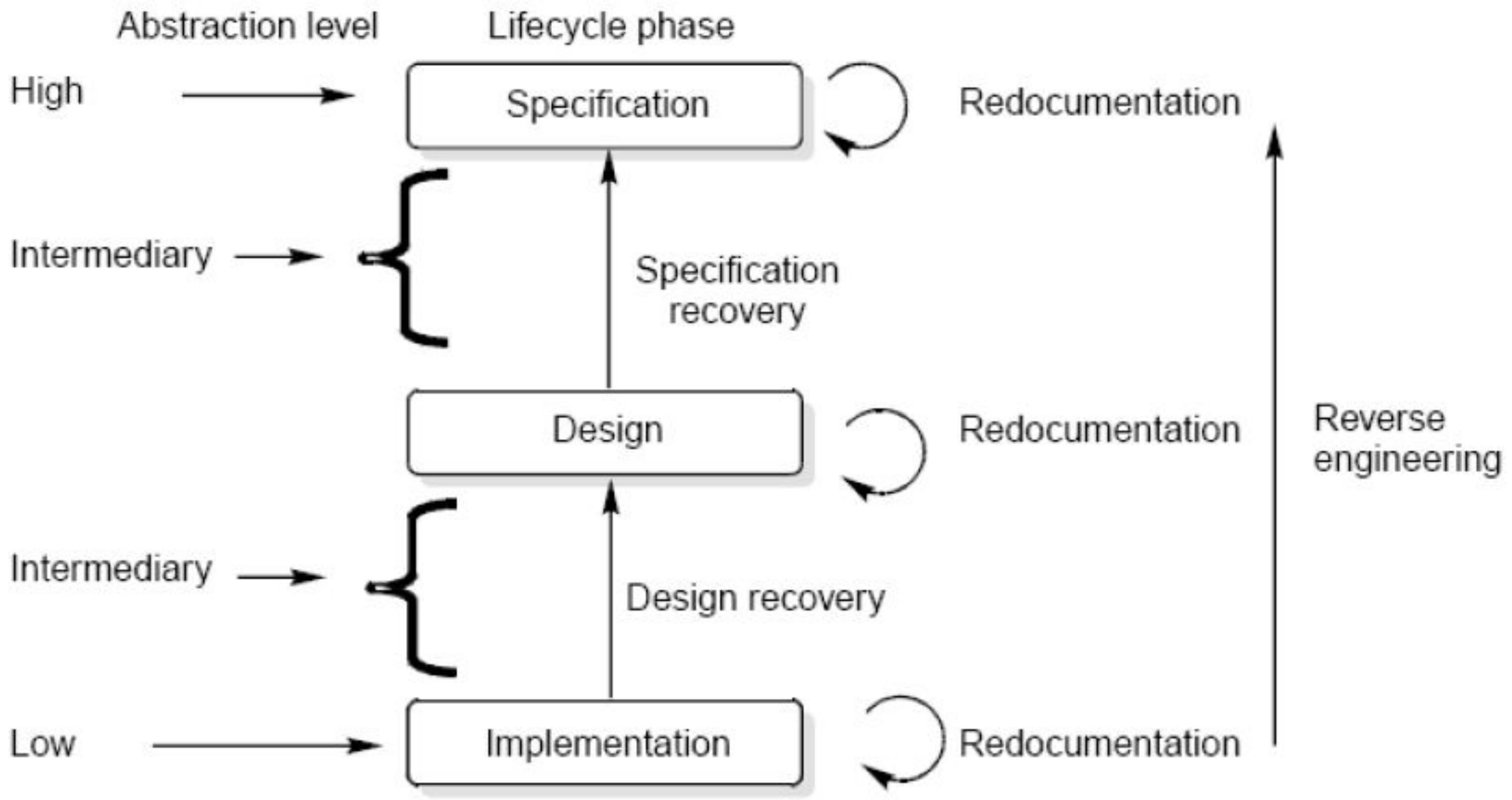


Fig: Levels of abstraction

- **Redocumentation**

  Redocumentation is the recreation of a semantically equivalent representation within the same relative abstraction level.

- **Design recovery**

  Design recovery entails identifying and extracting meaningful higher level abstractions beyond those obtained directly from examination of the source code. This may be achieved from a combination of code, existing design documentation, personal experience, and knowledge of the problem and application domains.

Advantages of Reverse Engineering

- It concentrates on **recovering the lost information** from the programs
- It provides the **abstract information from the detailed source code implementation**.
- It **improves system documentation** that is either incomplete or out of date.
- It **detects the adverse effects of modification in the s/w system**.

Difference between Reverse, Forward & Reengineering

- **Reverse Engineering**

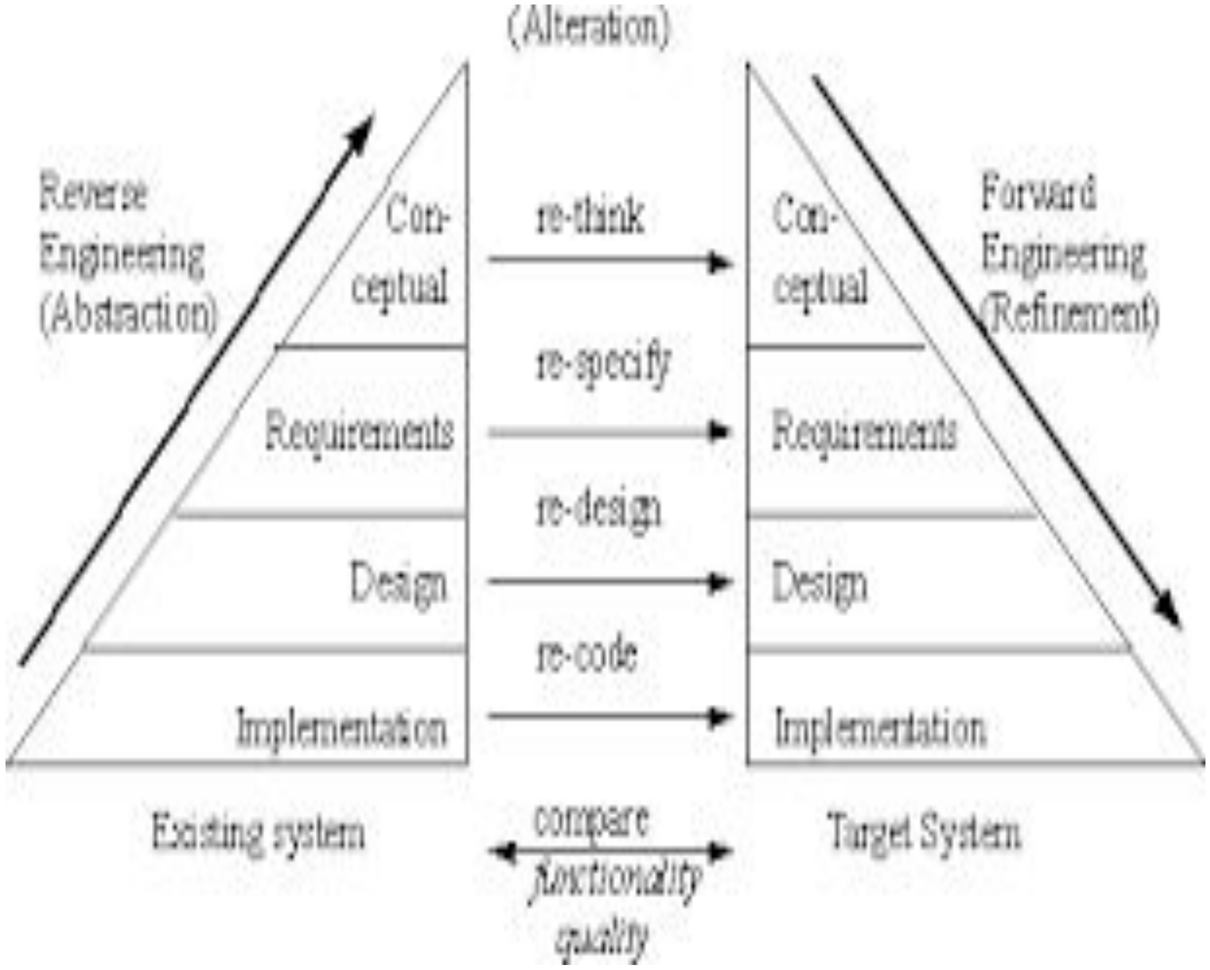
  It performs transformation from a large abstraction level to a higher one.

- **Forward Engineering**

  It performs transformation from a higher abstraction level to a lower one.

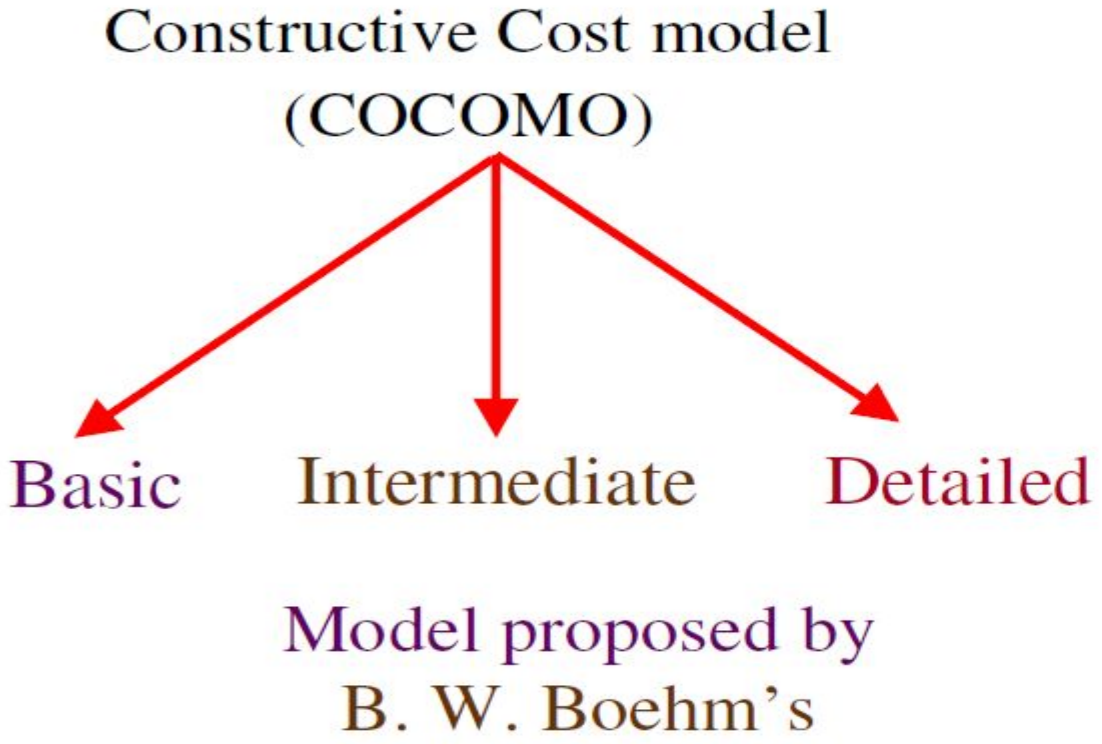
- **Reengineering**

  It transforms an existing s/w system into a new but more maintainable system.


(Alteration)
Reverse Engineering (Abstraction)
Con-ceptual
Requirements
Design
Implementation
re-think
re-specify
re-design
re-code
Con-ceptual
Requirements
Design
Implementation
Forward Engineering (Refinement)
Existing system
compare
functionality
quality
Target System

The Constructive Cost Model (COCOMO)

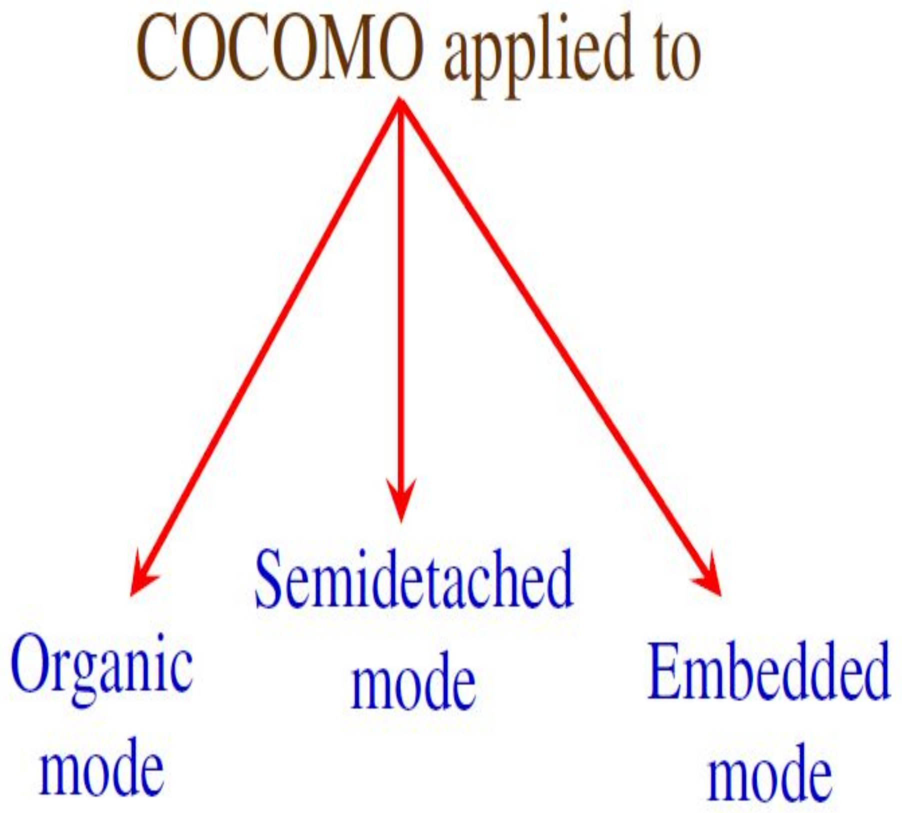
It is a hierarchy of software costs estimation models, which include basic, intermediate & detailed sub models.


Constructive Cost model (COCOMO)
Basic
Intermediate
Detailed
Model proposed by B. W. Boehm's

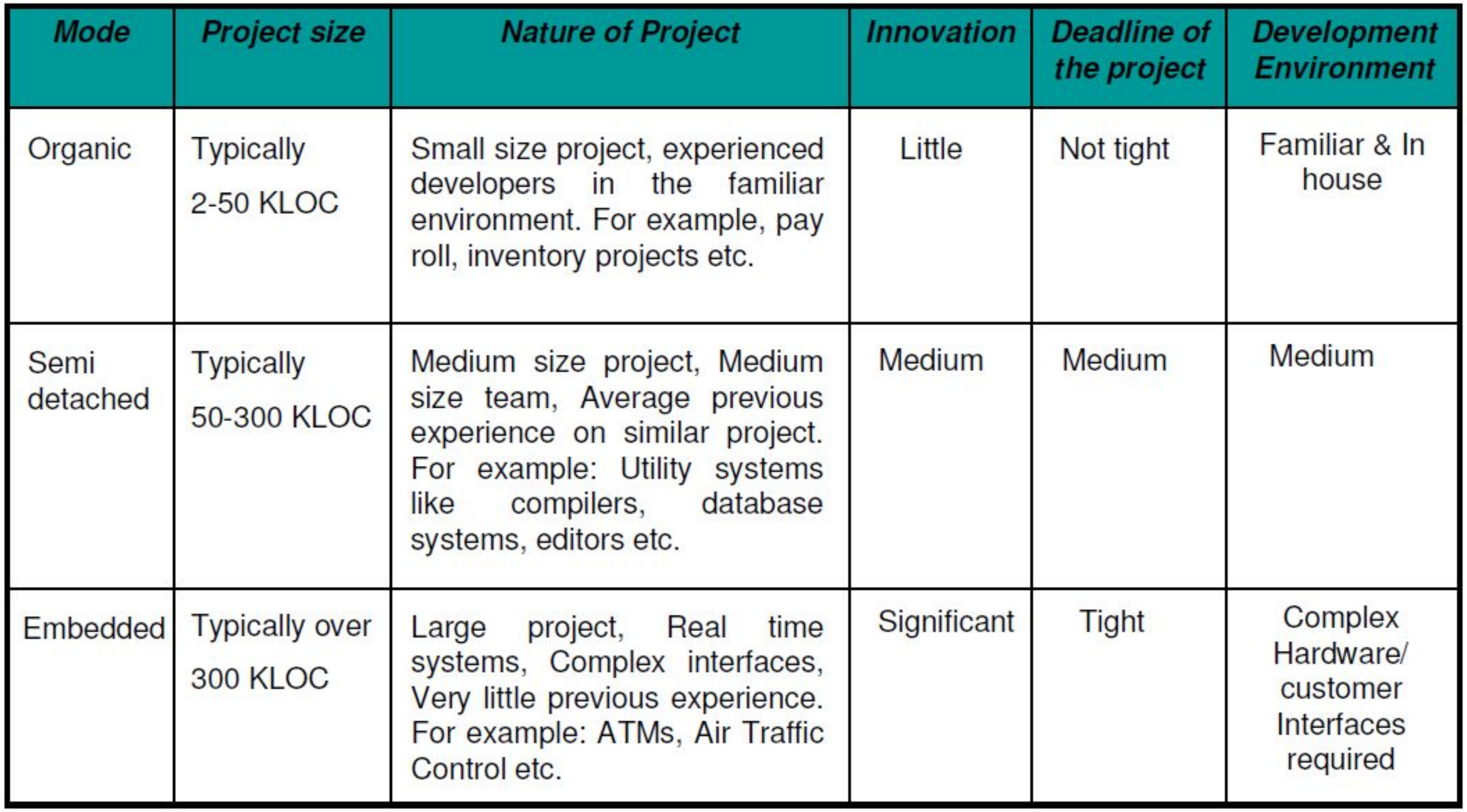
BASIC MODEL

It aims at estimating small to medium sized software projects.

Three modes of development are considered in this model: organic, semidetached and embedded.



Comparison of three COCOMO models



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 –

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

$$D = c_b (E)^{d_b}$$

where E is effort applied in Person-Months and D is the development time in months and the coefficients a_b, b_b, c_b, 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

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:

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

Example

Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes i.e., organic, semidetached and embedded.

Solution

The basic COCOMO equation take the form:

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

$$D = c_b (KLOC)^{d_b}$$

Estimated size of the project = 400 KLOC

(i) Organic mode

$$E = 2.4(400)^{1.05} = 1295.31 \text{ PM}$$

$$D = 2.5(1295.31)^{0.38} = 38.07 \text{ PM}$$

(ii) Semidetached mode

$$E = 3.0(400)^{1.12} = 2462.79 \text{ PM}$$

$$D = 2.5(2462.79)^{0.35} = 38.45 \text{ PM}$$

(iii) Embedded mode

$$E = 3.6(400)^{1.20} = 4772.81 \text{ PM}$$

$$D = 2.5(4772.8)^{0.32} = 38 \text{ PM}$$

Example

A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the effort, development time, average staff size and productivity of the project.

Solution

The semi-detached mode is the most appropriate mode; keeping in view the size, schedule and experience of the development team.

Hence $E = 3.0(200)^{1.12} = 1133.12$ PM

$D = 2.5(1133.12)^{0.35} = 29.3$ PM

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

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

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

$$P = 176\, LOC / PM$$

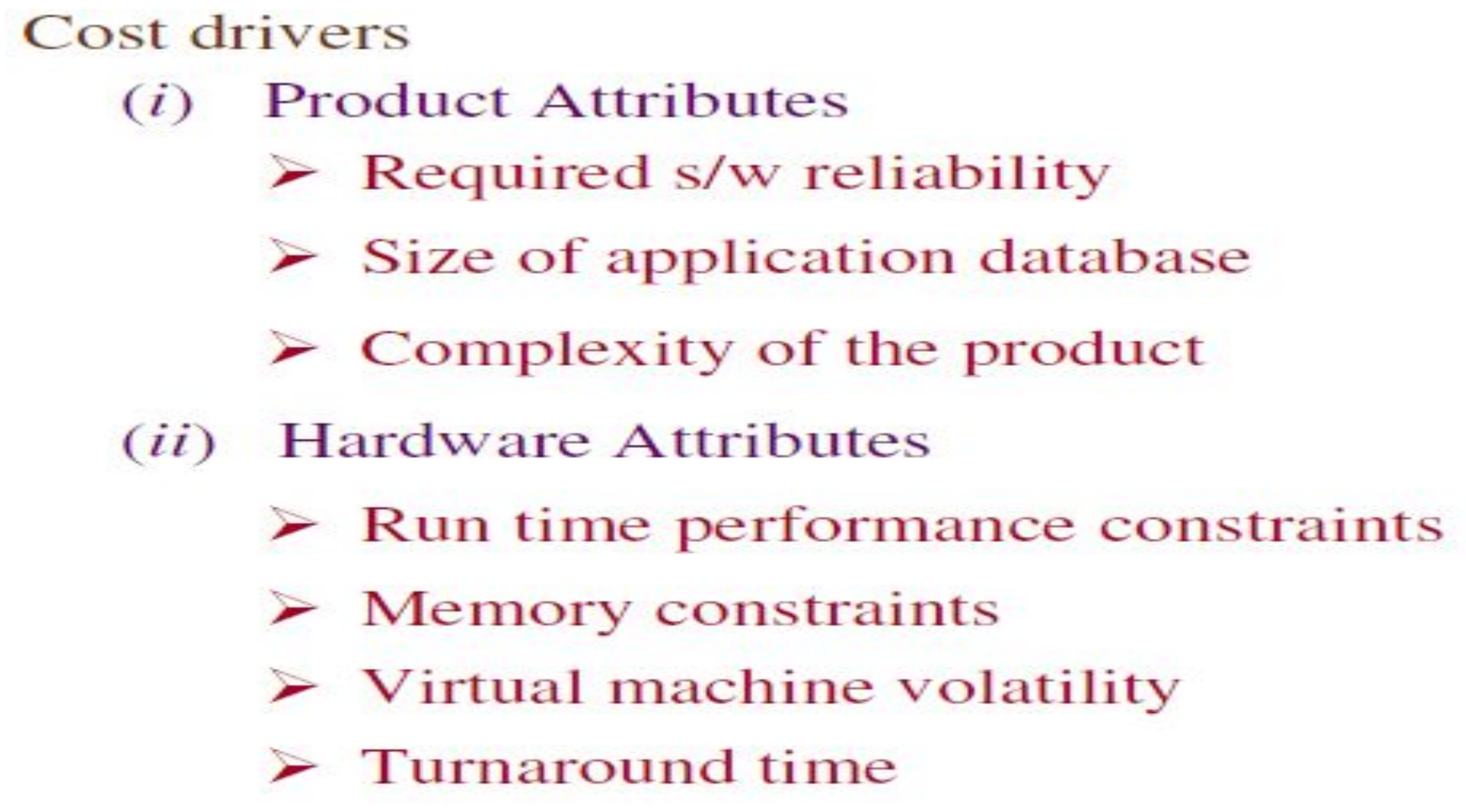
Intermediate Model

- The basic model allowed for a quick and rough estimate, but it resulted in a **lack of accuracy**.

- Boehm introduced an additional set of **15 predictors called cost drivers** in the intermediate model to take account of the software development environment.

- **Cost drivers are used to adjust the nominal cost of a project to the actual project environment, hence increasing the accuracy of the estimate.**

The cost drivers are grouped into four categories: -



Cost drivers

(*i*) Product Attributes

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

(*ii*) Hardware Attributes

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

(iii) Personal Attributes

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

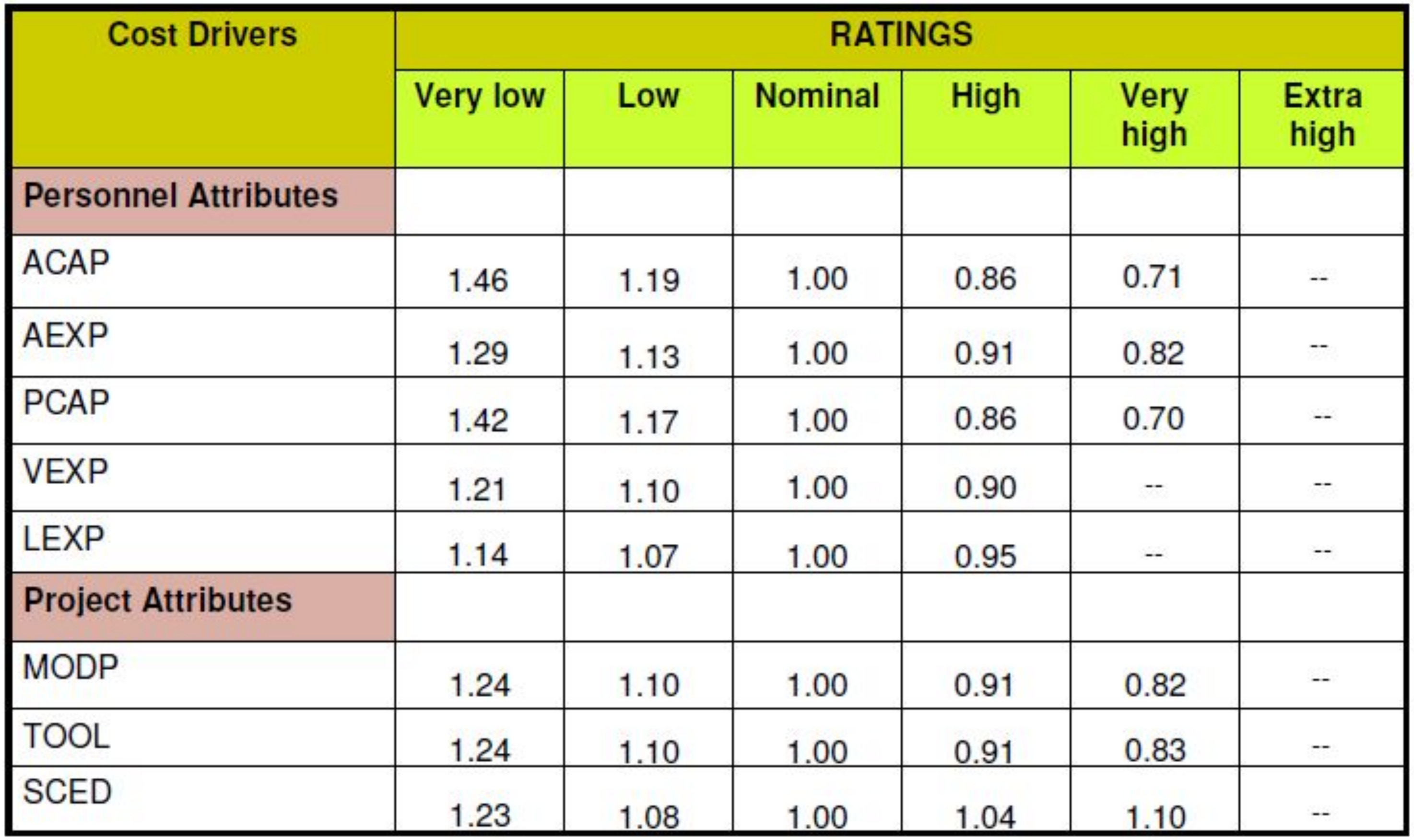
(iv) Project Attributes

- ➢ 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	--

Multiplier Values for Effort Calculation



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	--

The multiplying factors for all 15 cost drivers are multiplied to get the effort adjustment factor(EAF).

Intermediate COCOMO equations

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

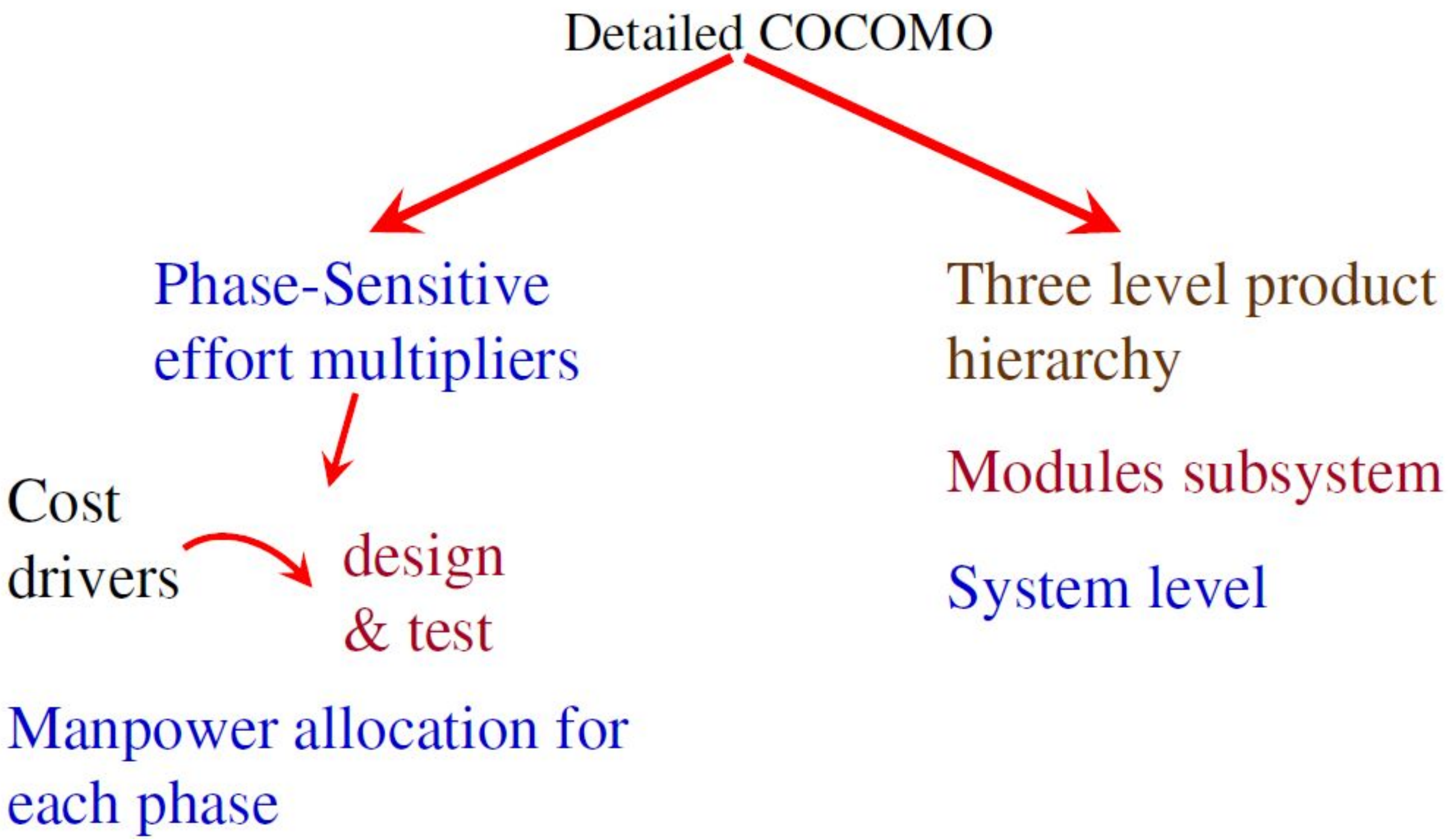
$$D = c_i (E)^{d_i}$$

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

- It offers a means for **processing all the project characteristics to construct a s/w estimate**. The detailed model introduces two more capabilities:
  - **Phase sensitive effort multiplier:** Some phases are more affected than others by factors defined by the cost drivers. The detailed model provides a set of phase sensitive effort multipliers for each cost driver. **This helps in determining the manpower allocation for each phase of the project**.
  - **Three-level product hierarchy:** Three product levels are defined. These are **module, subsystem and system levels.** The ratings of the cost drivers are done at appropriate level, i.e the level at which it is most susceptible to variation.

Detailed COCOMO Model


Detailed COCOMO
Phase-Sensitive effort multipliers
Cost drivers
design & test
Manpower allocation for each phase
Three level product hierarchy
Modules subsystem
System level

CONFIGURATION MANAGEMENT ACTIVITIES

- The software may be considered as configurations of software components. These software components are released in the form of executable code whereas supplier company keeps the source code. This source code is the representation of an executable equivalent.
- Source code can be modified without any effect upon executable versions in use. If strict controls are not kept, the source code which is the exact representation of a particular executable version may no longer exist.
- **The means by which the process of software development and maintenance is controlled is called configuration management.**

Goals of SCM

- Goals of SCM are-
  - SCM activities are **planned**.
  - **Changes to identified software work products are controlled**.
  - **Affected groups & individuals are informed of the status and content of s/w baselines.**
  - Selected s/w work products are identified, controlled and available.

Functions of SCM

- Identification
- Change Control
- Status Accounting
- Auditing

- **Identification** -identifies those items whose configuration needs to be controlled, usually consisting of hardware, software, and documentation.

- **Change Control - establishes procedures for proposing or requesting changes**, evaluating those changes for desirability, obtaining authorization for changes, publishing and tracking changes, and implementing changes. This function also identifies the people and organizations who have authority to make changes at various levels.

- **Status Accounting** -is the documentation function of CM. Its primary purpose is to maintain formal records of established configurations and make regular reports of configuration status. These records should accurately describe the product, and are used to verify the configuration of the system for testing, delivery, and other activities.

- **Auditing** -Effective CM requires regular evaluation of the configuration. This is done through the auditing function, where the physical and functional configurations are compared to the documented configuration. The purpose of auditing is to maintain the integrity of the baseline and release configurations for all controlled products

Following documents are required -

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

- All the components of the system’s configuration are recorded along with all relationships & dependencies b/w them.
- Any change/addition, deletion or modification must be recorded and its effect upon the rest of the system’s components should be checked.
- After a change has been made, a new configuration is recorded. There is need to know who is responsible for every procedure & process along the way.
- It is a management task both to assign these responsibilities and to conduct audit to see that they are carried out.

Principal Activities of SCM

- Configuration mgmt is carried out through two principal activities:
  - **Configuration identification** involves deciding which parts of the system should be kept track of.
  - **Configuration control** ensures that changes to a system happen smoothly.

Change Control Process

- Change control involves procedures and tools to bring order in the change process. Larger projects have a formal **change control board (CCB)**, whose responsibility is to review and approve or disapprove change.

- It is the CCB responsibility to provide the mechanism to maintain orderly change processes. For a large software engineering project, uncontrolled change rapidly leads to chaos. For such projects, change control combines human procedures & automated tools to provide a mechanism for the control of change.

Goals of Change Control Process

- Offer a **mechanism for accepting changes that enhance the product while rejecting those that degrade it**.
- **Offer revision control & backup safety** for work products during their formative development.
- Facilitate changes to work products during their initial formative development while avoiding unnecessary overhead or formality.
- Permit for formal acceptance of work products after their initial formative development has been finished.
- Permit all parties materially affected by proposed changes to accepted work products to assess the resource, schedule or product impact of the changes.
- **Notify interested parties on the periphery of development regarding change proposals, their assessed impact and whether the changes were approved or rejected.**

Software Version Control

- During software maintenance at least two versions of the software system are produced , one being the old version and the other one or more new versions.
- As a s/w system comprises of s/w components, there will also be two or more versions of each component that has been changed. Thus, the software maintenance team has to cope with multiple versions of the software

- 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

- A **version control system** is a repository of files . Every change made to the source is tracked, along with **who made the change, why they made it and references to problems fixed or enhancements introduced**, by the change.

- User's may wish to compare today's version of some s/w with yesterday's version or last year's version. Since version control system keep track of every version of the s/w, this becomes a straightforward task.

- Version control activity is divided in four sub-activities:

1. Identifying new versions
2. Numbering scheme : It will have the following format

Version X.Y.Z…

3. Visibility
4. Tracking

1. **Identifying new versions:** A s/w configuration item (SCI) will get a new version no. when there has been a change to its established baseline. Each previous version will be stored in a corresponding directory like version 0, version1, version2 etc.

2. **Numbering scheme:** It will have the following format

Version X.Y.Z…

- (X) denotes the entire SCI. Therefore, **changes made to the entire configuration item, or changes large enough to warrant a completely new release of the item**, will cause the first digit to increase.
- (Y) denotes the component of SCI. this digit will sequentially **increase if a change is made to a component or multiple components**.

- (Z) denotes the **section of the component** of a SCI. This no. will only be visible if a component of an SCI can be divided into individual sections**. Changes made at this level** of detail will need a sequential change of third digit.

3. **Visibility:** the version no. can be seen either in frame or below the title.

4. **Tracking :** the best way to keep track of the different versions is with a version evolution graph.

CASE

- CASE stands for **C**omputer **A**ided **S**oftware **E**ngineering. It means, **development and maintenance** of software projects with help of various automated software tools.

CASE Tools

- Computer aided software engineering tool assist software engineering managers & practitioners in every activity associated with the software process.

- Thus, a CASE tool means any tool used to automate some activity associated with software development. Many CASE tools are available. Some of these CASE tools assist in **phase related tasks** such as specification, structure analysis, design, coding, testing etc; and others to **non-phase activities** such as project management and configuration management.

CASE TOOLS

- CASE tools are software programs that are designed to assist human programmers with the complexity of the processes & the artifacts of software engineering.
- As we know, software engineering is a systematic approach to the development, operations, maintenance & retirement of software.

- Software engineering mainly includes the following processes, which may be aided by CASE:-
  - Translation of user needs into software requirements.
  - Transformation of software requirements into design specifications.
  - Implementation of design into code.
  - Testing of the code for the operational use.
  - Documentation.

Reasons for using CASE Tools

- The primary reason for using a CASE Tool are:
  - **To increase productivity:** Automation of various activities of system development and management processes increases productivity of the development team.
  - **To help produce better quality software at low maintenance cost:** Use of CASE tools makes the software easy to maintain and hence reduce the maintenance costs.

Other major benefits include the following-

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

- **Improved productivity:** CASE tools provide automation & reduce the time to complete many tasks, hence improvements in productivity.

- **Better documentation:** By using CASE Tools, vast amount of documentation are produced along the way.

- **Reduced Lifetime Maintenance:** As a result of better design, better analysis and automatic code generation, automatic testing & debugging overall system's quality improves. There is better documentation also. Thus, the net effort & cost involved with maintenance is reduced.

- **Improved Accuracy:** CASE tools can provide ongoing debugging & error checking which is very vital for early defect removal, which actually played a major role in shaping modern software.

- **Opportunity to non-programmers:** With the increased movement towards object oriented technology & client server bases, programming can also be done by people who don’t have a complete programming background. By using the lower case tools, it could be possible to develop software from the initial design & analysis phase.

Characteristics of a CASE Tool

A CASE tool must have the following characteristics in order to be used efficiently:

- **A standard methodology**
- **Flexibility**
- **Strong Integration**
- **Integration with testing software**
- **Support for reverse engineering**
- **On-line help**

- **A standard methodology:** A CASE tool must support a standard software development methodology and standard modeling techniques. In the present scenario most of the CASE tools are moving towards UML.

- **Flexibility:** Flexibility in use of editors and other tools. The CASE tool must offer flexibility and the choice for the user of editors' development environments.

- **Strong Integration:** The CASE tools should be integrated to support all the stages. This implies that if a change is made at any stage, for example, in the model, it should get reflected in the code documentation and all related design and other documents, thus providing a cohesive environment for software development.

- **Integration with testing software:** The CASE tools must provide interfaces for automatic testing tools that take care of regression and other kinds of testing software under the changing requirements.

- **Support for reverse engineering:** A CASE tools must be able to generate complex models from already generated code.

- **On-line help:** The CASE tools provide an online tutorial.

Categories of CASE Tools:

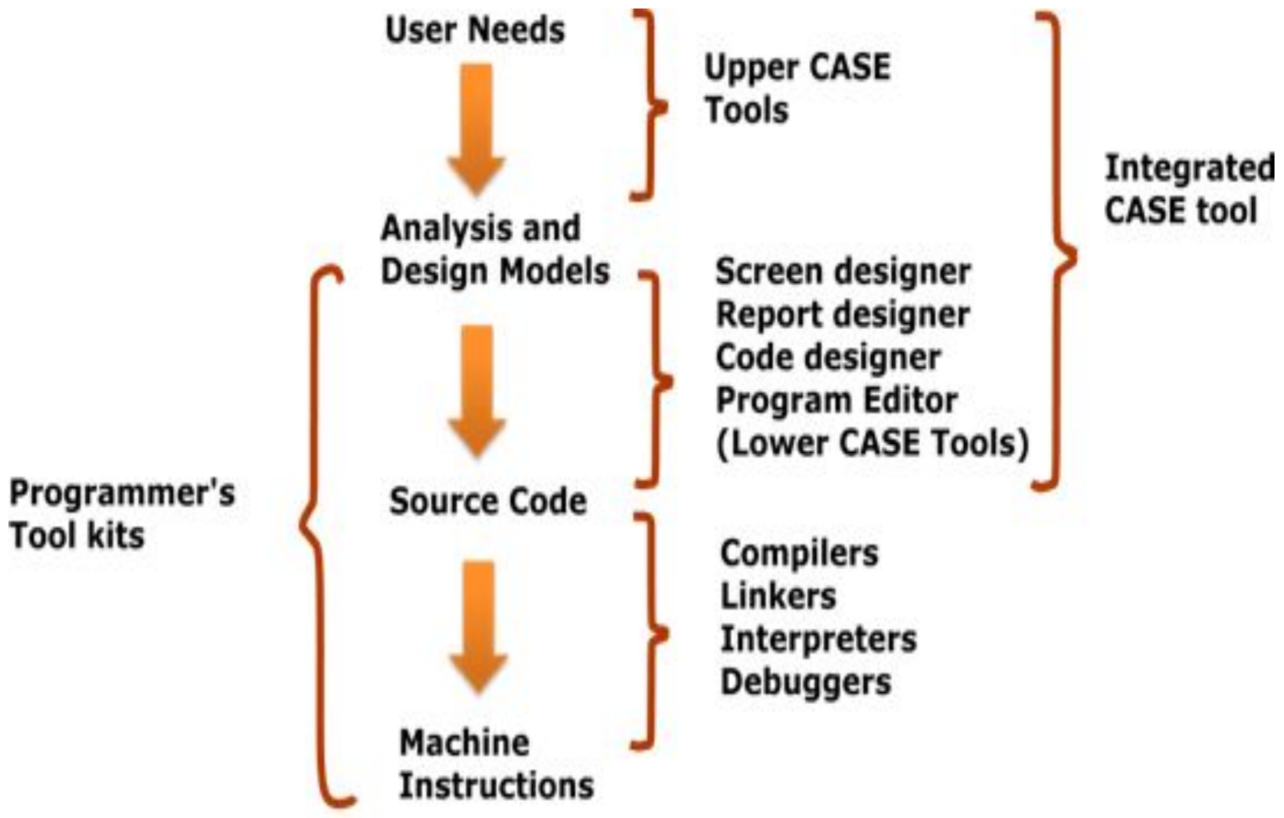
- Sometimes CASE tools are classified in to following categories due to their activities:
  - UPPER CASE Tools
  - LOWER CASE Tools
  - INTEGRATED CASE Tools

Upper CASE tools: They support the **analysis and the design phase**. They include tools for analysis modeling, reports and forms generation.

Lower CASE tools: They support the **coding phase, configuration management**, etc.

Integrated CASE tools: It is known as I-CASE and also supports analysis, design and coding phases.

Positioning of CASE tools in a Software Application development:



CASE Tool And Application Development

Limitations of CASE tools

- The major limitations of using CASE tools are:
  - **Cost**
  - **Learning Curve**
  - **Tool mix**

- **Cost:** Using CASE tools is a very costly affair. In fact, most firms engaged in software development on a small scale do not invest in CASE tools because they think that the benefits of CASE are justifiable only in the development of large systems.

- **Learning Curve:** In most cases, programmer productivity may fall in the initial phase of implementation, because user need time to learn the technology. In fact, a CASE consulting industry has evolved to support uses of CASE tools. The consultants offer training & on-site services that can be crucial to accelerate the learning curve and to the development & use of the tools.

- **Tool mix:** It is important to make an appropriate selection of tool mix to get cost advantage CASE integration & data integration across all platforms is also very important. The ability to share the results of work done on one CASE tool with another CASE tool is perhaps the most important type of CASE integration.

CASE Support in Software Life Cycle

- There are various types of support that CASE provides during the different phases of a software life cycle.

  – Prototyping Support
  – Structured analysis & design
  – Code Generation
  – Test CASE generator

Prototyping Support: The prototyping is useful to understand the requirements of complex software products, to market new ideas and so on. The prototyping CASE tools requirements are as follows:

- Define user interaction
- Define the system control flow
- Store & retrieve data required by the system
- Incorporate some processing logic

Structured Analysis & Design: Several diagramming techniques are used for structured analysis and structured design. The following supports might be available from CASE tools.

- A CASE tool should support one or more of the structured analysis and design techniques.
- It should **support effortlessly drawing analysis and design diagrams**.
- It should **support drawing for fairly complex diagrams**, preferably through a hierarchy of levels.
- The CASE tool should provide easy navigation through the different levels and through the design and analysis.
- The tool must **support completeness and consistency checking across the design and analysis and through all levels of analysis hierarchy**. Whenever it is possible, the system should disallow any inconsistent operation, but it may be very difficult to implement such a feature. Whenever there arises heavy computational load while consistency checking, it should be possible to temporarily disable consistency checking.

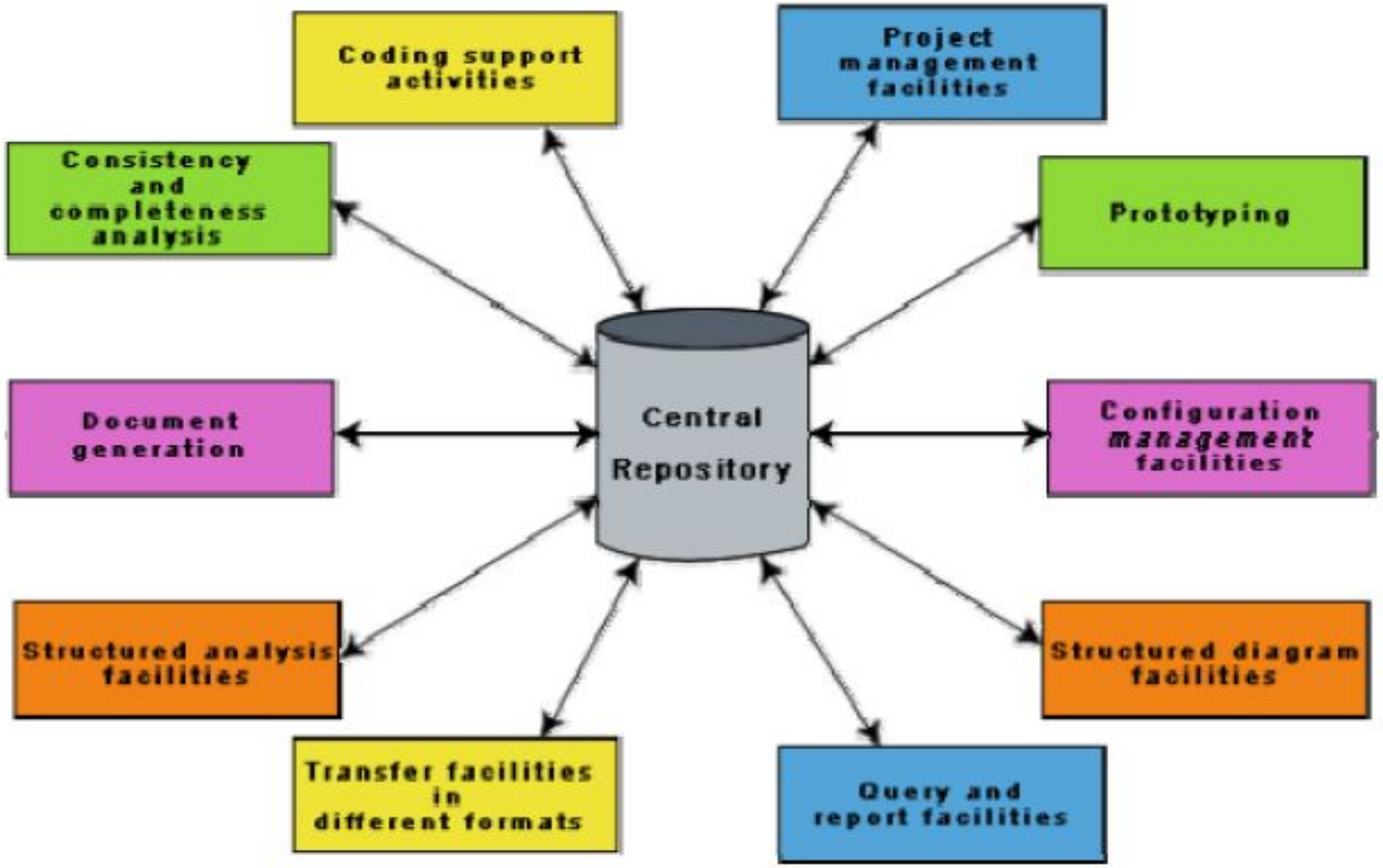
Code Generation: As far as code generation is concerned, the general expectation of a CASE tool is quite low. A reasonable requirement **is traceability from source file to design data**. More pragmatic supports expected from a CASE tool during code generation phase are the following:

- The CASE tool should **support generation of module skeletons or templates in one or more popular languages**. It should be possible to include copyright message, brief description of the module, author name and the date of creation in some selectable format.
- The tool should generate records, structures, class definition automatically from the contents of the data dictionary in one or more popular languages.
- It should **generate database tables for relational database management systems.**
- The tool should generate code for user interface from prototype definition for X window and MS window based applications.

Test CASE Generator: The CASE tool for test case generation should have the following features:

- It should support both design and requirement testing.
- It should generate test set reports in ASCII format which can be directly imported into the test plan document.

CASE Environment


Coding support activities
Project management facilities
Consistency and completeness analysis
Prototyping
Document generation
Central Repository
Configuration management facilities
Structured analysis facilities
Structured diagram facilities
Transfer facilities in different formats
Query and report facilities

- Although individual CASE tools are useful, the true power of a tool set can be realized only when these set of tools are integrated into a common framework or environment. CASE tools are characterized by the stage or stages of software development life cycle on which they focus. Since different tools covering different stages share common information, it is required that they integrate through some central repository to have a consistent view of information associated with the software development artifacts. This central repository is usually a data dictionary containing the definition of all composite and elementary data items. Through the central repository all the CASE tools in a CASE environment share common information among themselves. Thus a CASE environment facilities the automation of the step-by-step methodologies for software development. A schematic representation of a CASE environment is shown in fig.

Benefits of CASE

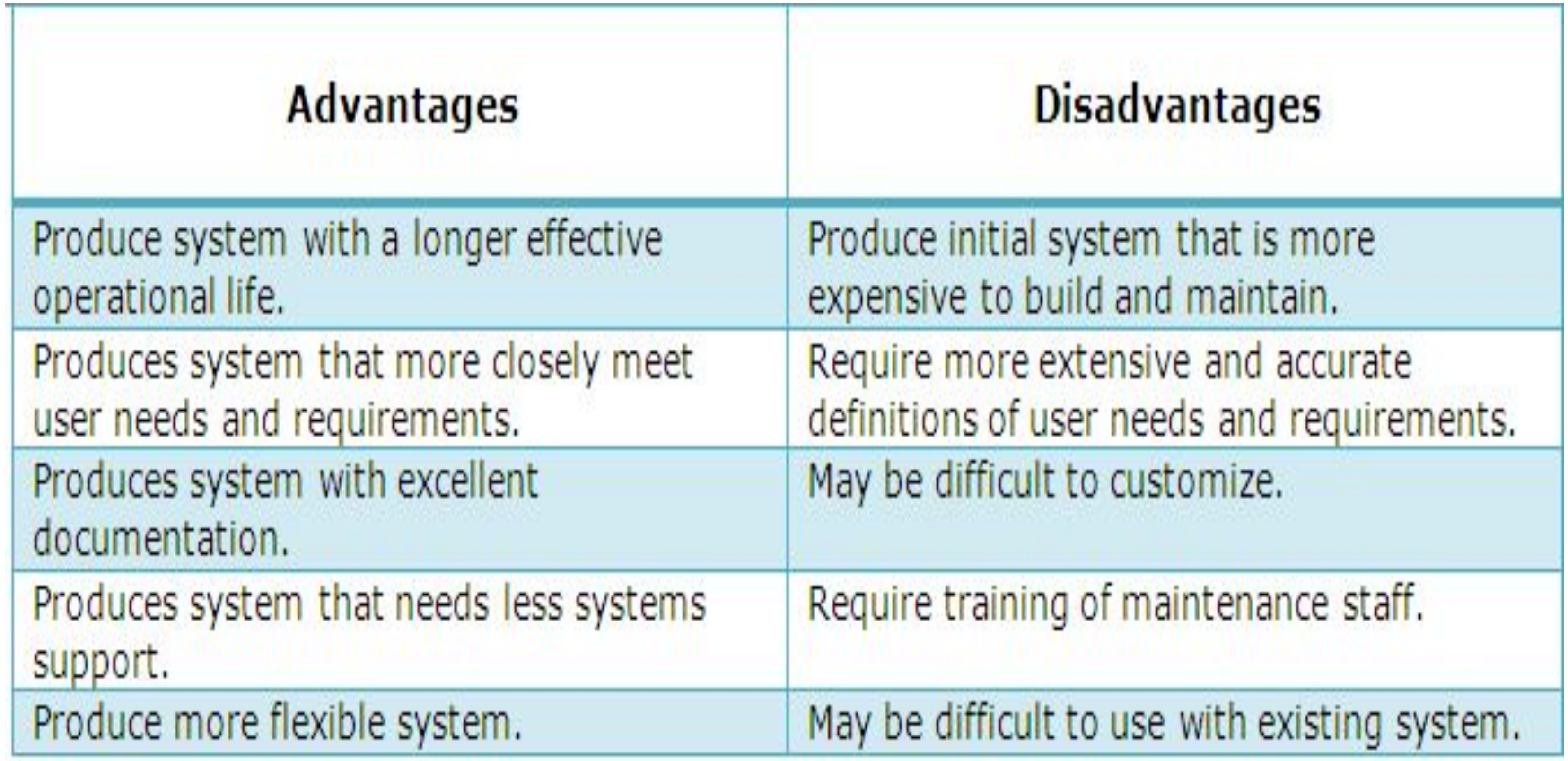
- A key benefit arising out of the use of a CASE environment is **cost saving through all development phases**. Different studies carry out to measure the impact of CASE put the effort reduction between 30% to 40%.

- Use of CASE **tools leads to considerable improvements to quality**. This is mainly due to the facts that one can effortlessly iterate through the different phases of software development and the chances of **human error are considerably reduced**.

- CASE tools help produce **high quality and consistent documents.** Since the important data relating to a software product are maintained in a central repository, redundancy in the stored data is reduced and therefore chances of inconsistent documentation is reduced to a great extent.

- CASE tools have led to revolutionary **cost saving in software maintenance efforts.** This arises not only due to the tremendous value of a CASE environment in traceability and consistency checks, but also due to the systematic information capture during the various phases of software development as a result of adhering to a CASE environment.

Advantages and Disadvantages of CASE Tools:



Advantages	Disadvantages
Produce system with a longer effective operational life.	Produce initial system that is more expensive to build and maintain.
Produces system that more closely meet user needs and requirements.	Require more extensive and accurate definitions of user needs and requirements.
Produces system with excellent documentation.	May be difficult to customize.
Produces system that needs less systems support.	Require training of maintenance staff.
Produce more flexible system.	May be difficult to use with existing system.

Software Risk Management

- We Software developers are extremely optimists.
- We assume, everything will go exactly as planned.
- Other view

  ↘ not possible to predict what is going to happen ?

  Software surprises

  ↘ Never good news

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”.

Types of Risks

- **Development Process Risks**
  - are **development errors, natural disasters, disgruntled (dissatisfied) employees** and **poor management objectives.**

- **Product Risks**
  - Includes **Project risks, Technical risks, Business risks**

Another general categorization of risks is -

- **Known Risks** are those that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources

- **Predictable Risks** are basis from past project experience.

- **Unpredictable risks** can and do occur, but they are extremely difficult to identify in advance.

RISK ANALYSIS AND MANAGEMENT

- Risk analysis and management are a series of steps that help a software team to understand and manage uncertainty. Many problems can plague a software project. A risk is a potential problem – it might happen, it might not.

- So, the key idea in risk management is not to wait for a risk to materialize and become a problem. The **objective** of risk management is to ensure that for each perceived risk , we know very well how to tackle it in advance.

- According to the risk management guru **Barry Boehm,** ***"Risk management focuses the project manager's attention on those portions of the project most likely to cause trouble"***

REACTIVE VS PROACTIVE RISK STRATEGIES

- A **reactive strategy** monitors the project for likely risks. Resources are set aside to deal with them, should they become actual problems. More commonly, the software team does nothing about risks, until something goes wrong. Then, the team flies into action in an attempt to correct the problem rapidly. This is often called a ***fire fighting mode.***

- A **proactive strategy** begins long before technical work is initiated. Potential risks are identified, their probability and impact are assessed, and they are ranked by importance. Then, the software team establishes a plan for managing risk. The primary objective is to avoid risk, but because not all risks can be avoided, the team works to develop a contingency plan that will enable it to respond in a controlled and effective manner.

Risk Management Activities

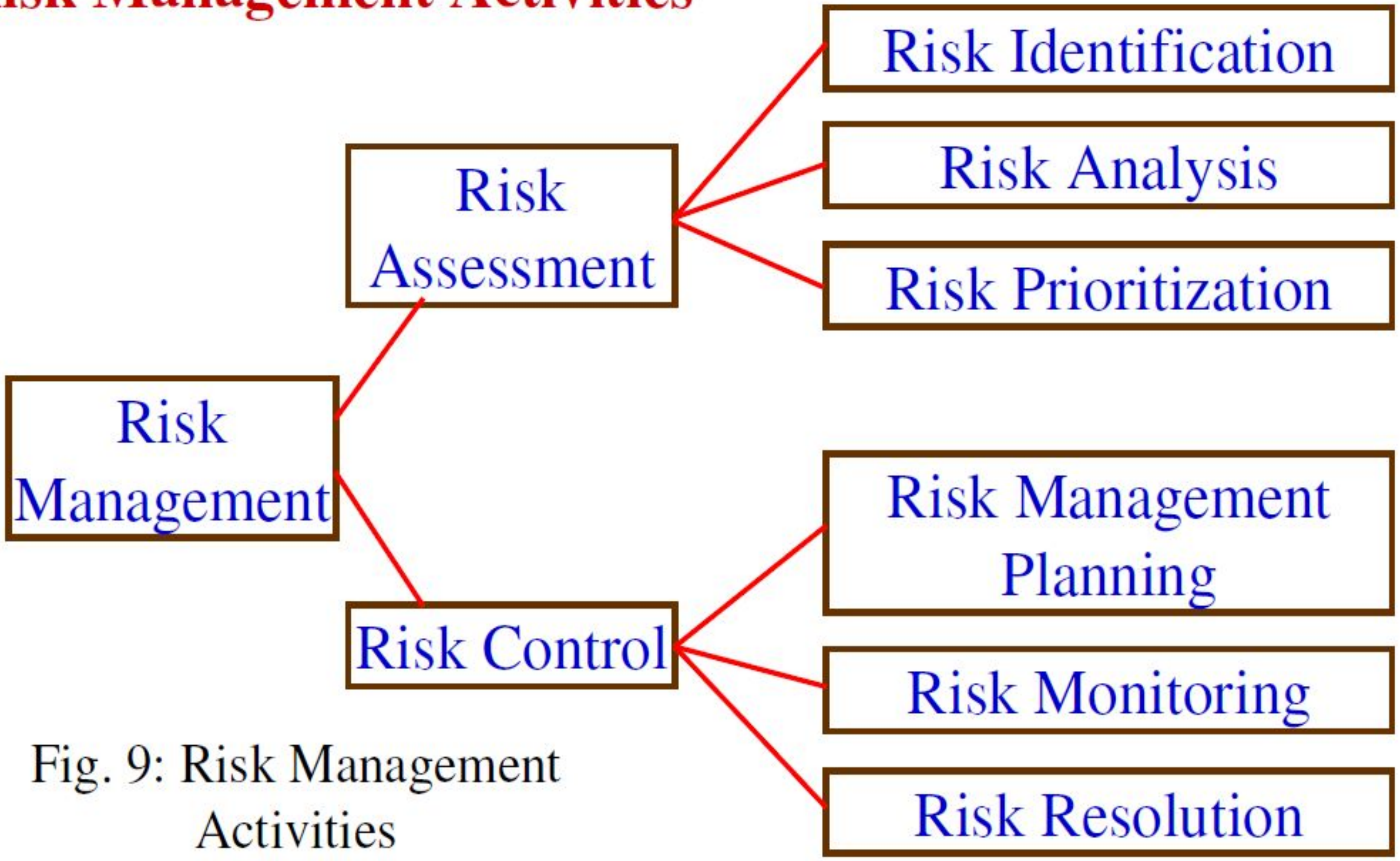


Fig. 9: Risk Management Activities

- **Risk Identification**

  With the identification phase, several activities occur. The main activities are:
  - **Identify risks**
  - **Define risk attributes**
  - **Documentation**
  - **Communicate**

- **Risk Analysis**

  The main activities in this phase are –
  - **Group similar risks**
  - **Determine risk drivers**
  - **Determine source of risks**
  - **Estimate risk exposure**
  - **Evaluate against criteria**

- **Risk Planning**
  - **Risk avoidance**
  - **Risk minimization**
  - **Risk Contingency Plans**

END OF UNIT-5

****All the Best****