Software Maintenance & Software Reuse

UNIT - V

Part- I : Software Maintenance :

- Software maintenance is a part of the Software Development Life Cycle.

- Its primary goal is to modify and update software product after delivery to correct errors and to improve performance.

- Software Maintenance is an inclusive activity that includes error corrections, enhancement of capabilities, deletion of obsolete capabilities, and optimization.

- Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product.

- Software maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes. A common perception of maintenance is that it merely involves fixing defects.

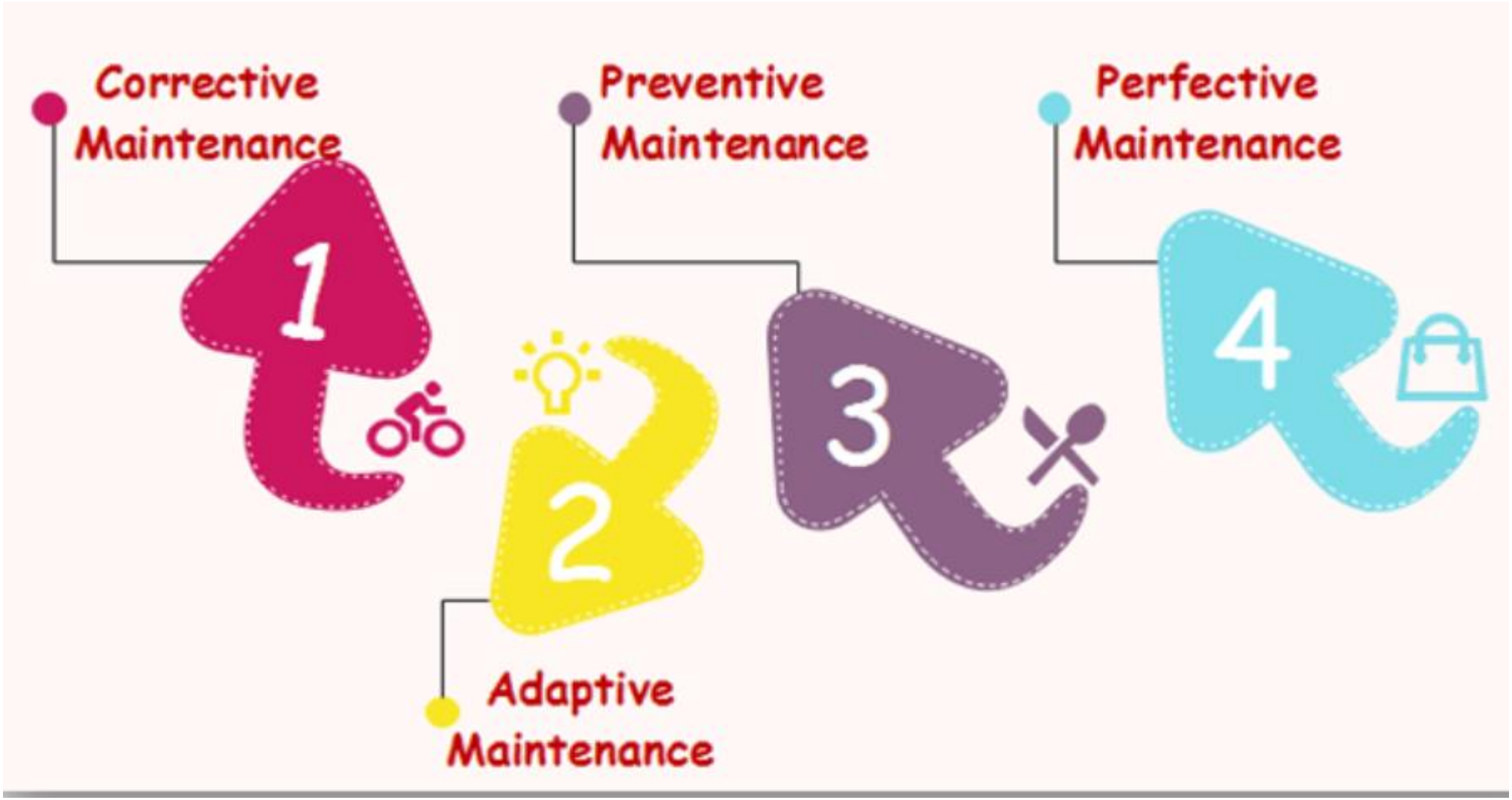
5.1 Software Maintenance is needed for:-

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

why modifications are required, some of them are briefly mentioned here:-

- Market Conditions - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping etc.,
- Client Requirements - Over the time, customer may ask for new features or functions in the software.
- Host Modifications - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- Organization Changes - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

Types of Software Maintenance :



1. Corrective Maintenance:

Corrective maintenance aims to correct any remaining errors regardless of where they may cause specifications, design, coding, testing, and documentation, etc.

2. Adaptive Maintenance:

It contains modifying the software to match changes in the ever-changing environment

3. 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 old technology is re-engineered using new technology. This maintenance prevents the system from dying out.

4. Perfective Maintenance:

It defines improving processing efficiency or performance or restricting the software to enhance changeability. This may contain enhancement of existing system functionality, improvement in computational efficiency, etc.

Characteristics of Software Maintenance(Software Evolution):-

Lehman and Belady have studied the characteristics of evolution of several software products.

observations in the form of laws.

1st law : A software product must change continually

2nd law : structure of a program tends to degrade as more and more maintenance is carried out on it.

3rd law : over a program's lifetime, its rate of development is approximately constant.

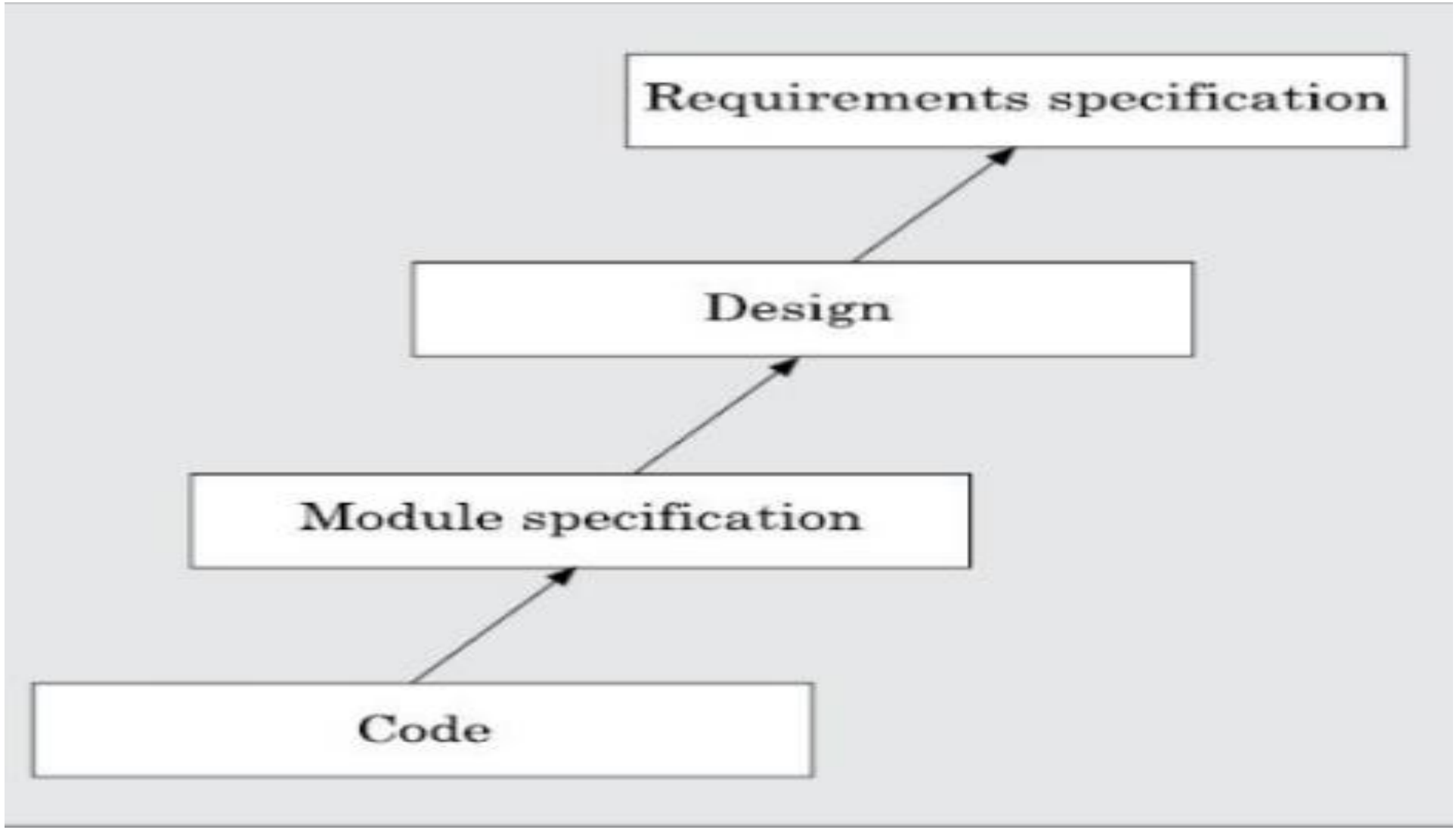
Software Reverse Engineering:-

- Is the process of recovering the design and the requirements specification of a product from an analysis of its code.
- When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.
- Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache.
- For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

Re-Engineering Process :

- Decide what to re-engineer. Is it whole software or a part of it?
- Perform Reverse Engineering, in order to obtain specifications of existing software.
- Restructure Program if required. For example, changing function-oriented
- programs into object-oriented programs.
- Re-structure data as required.
- Apply Forward engineering concepts in order to get re-engineered software.

Process Model for Reverse Engineering:-



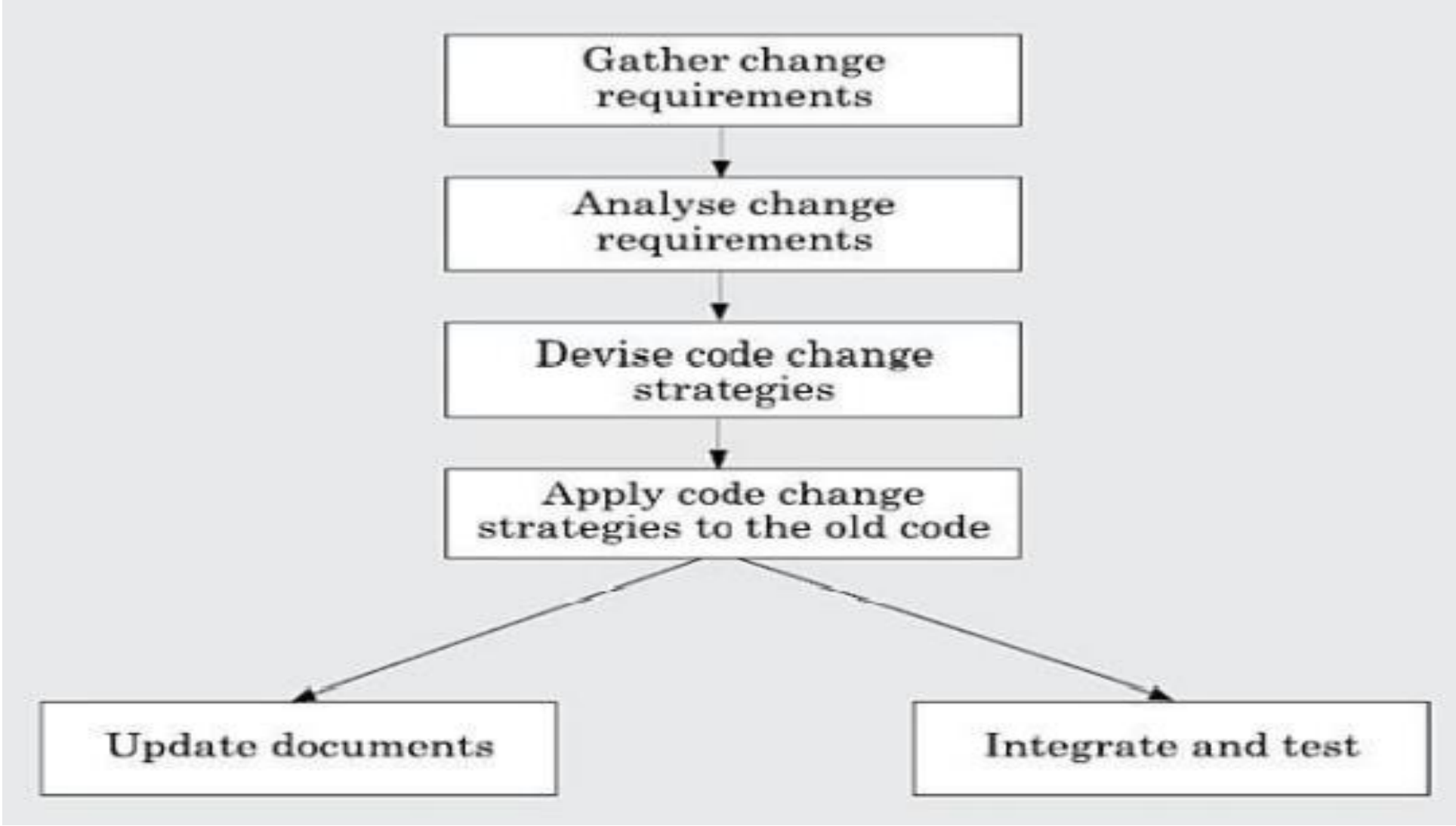
5.2 Software Maintenance Process Models:

The activities involved are

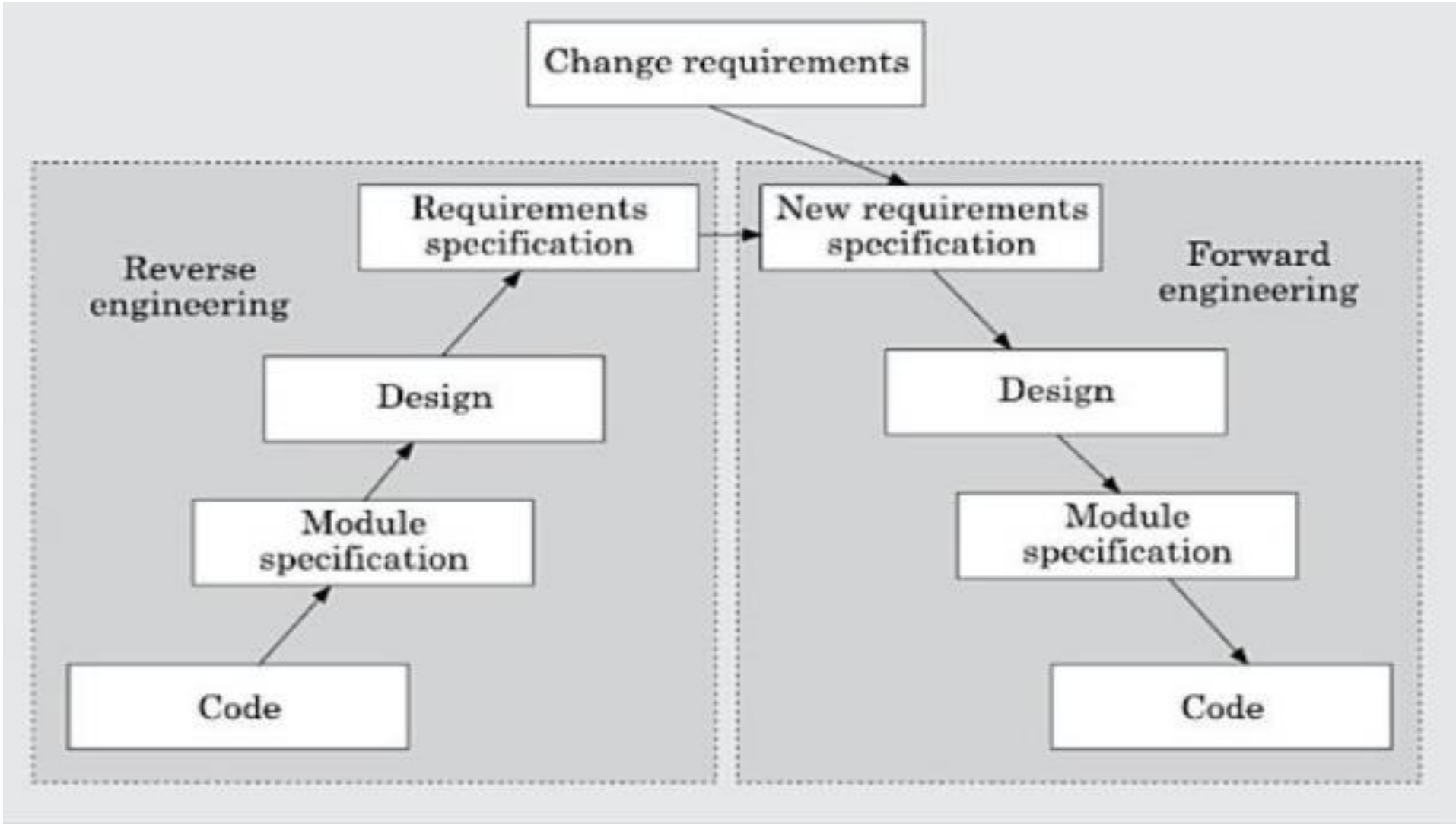
- The extent of modification to the product required
- The resources available to the maintenance team
- The conditions of the existing product (ex. How structured it is, How well documented it is etc.,)
- The expected project risks etc.,

Two broad categories of process models can be proposed.

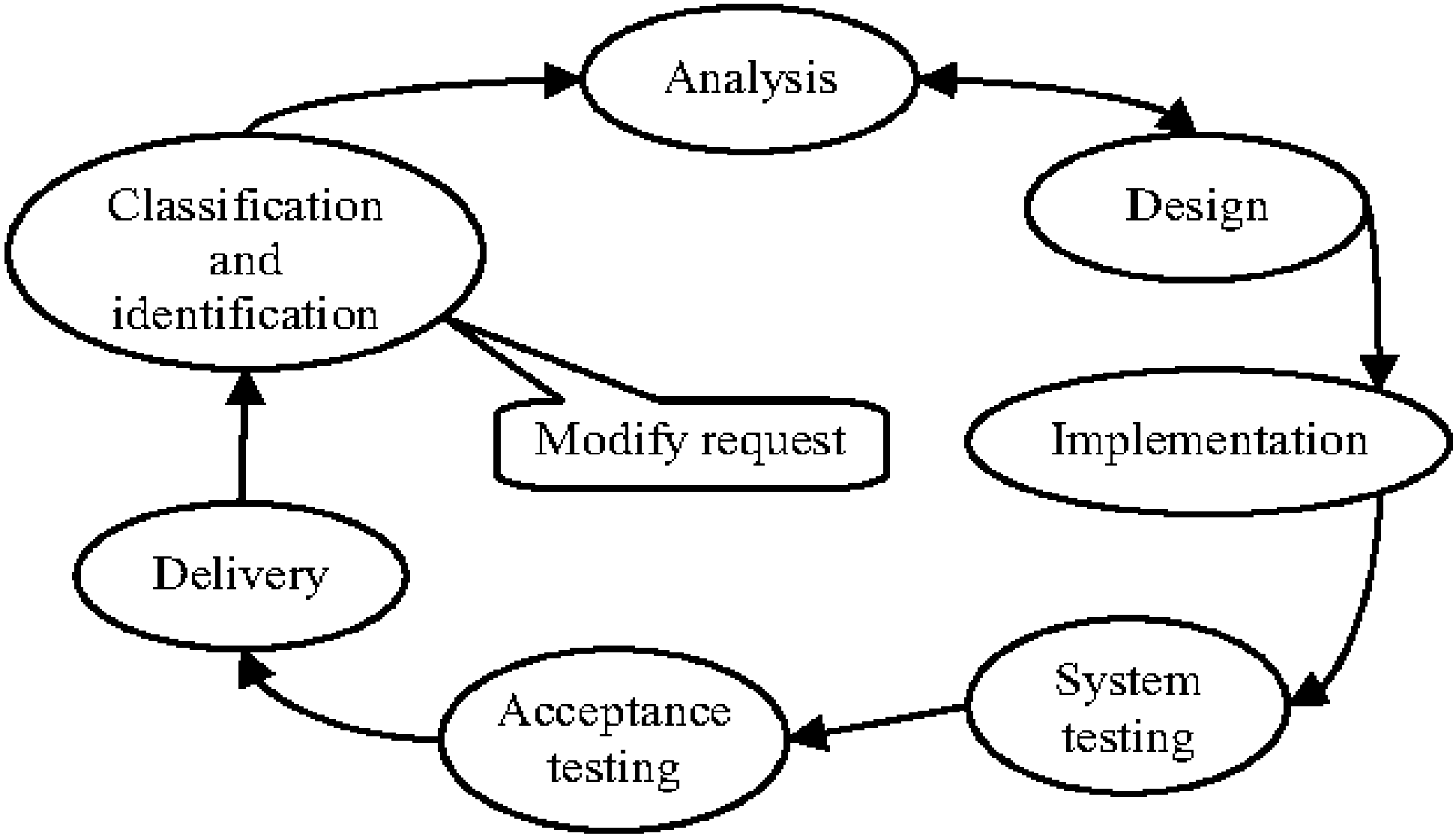
Software Maintenance Process Models: First Model



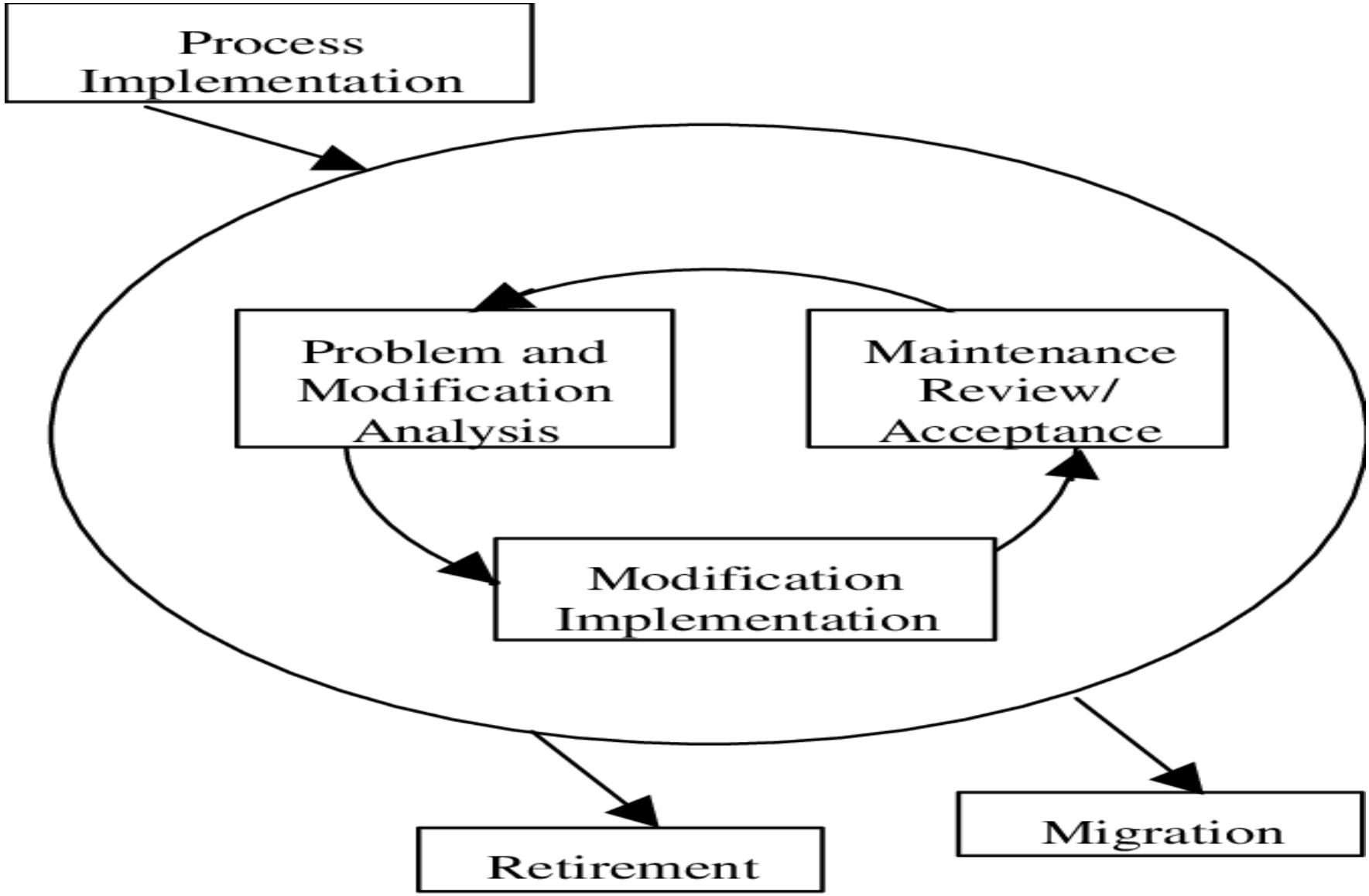
Second Model :



IEEE Model:


Analysis
Design
Classification and identification
Modify request
Implementation
Delivery
System testing
Acceptance testing

ISO Model:


Process Implementation
Problem and Modification Analysis
Maintenance Review/ Acceptance
Modification Implementation
Retirement
Migration

5.3 Estimation of Maintenance Cost:

- Maintenance costs vary widely from one application domain to another.
- For embedded systems, the maintenance cost can be as much as 2 to 4 times the development cost.
- Boehm proposed a formula for estimating maintenance costs as part of his Constructive Cost Model- COCOMO cost estimation model.
- Boehm's maintenance cost estimation is made in terms of a quantity called the Annual Change Traffic (ACT).

Estimation of Maintenance Cost:

- The annual change traffic(ACT) is multiplied with the total development cost to arrive at the maintenance cost.

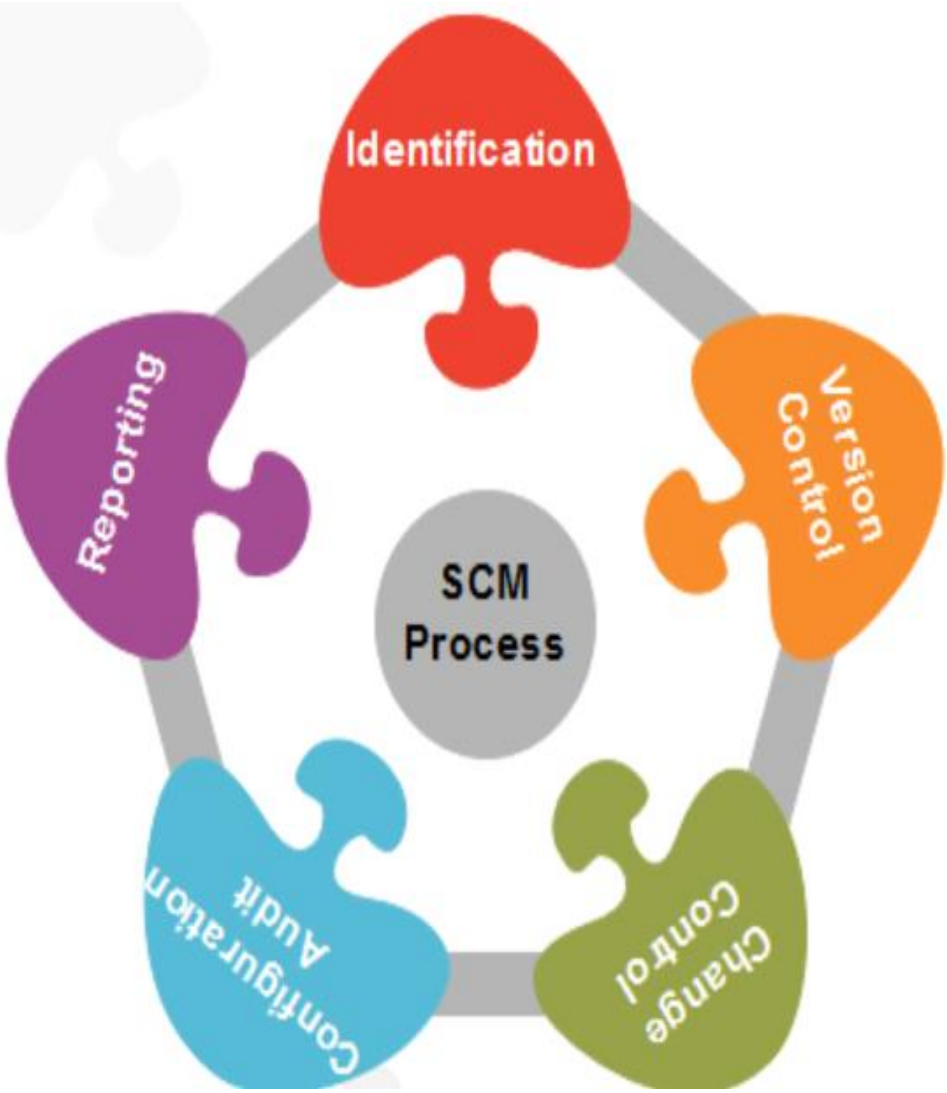
Maintenance cost= ACT x Development cost.

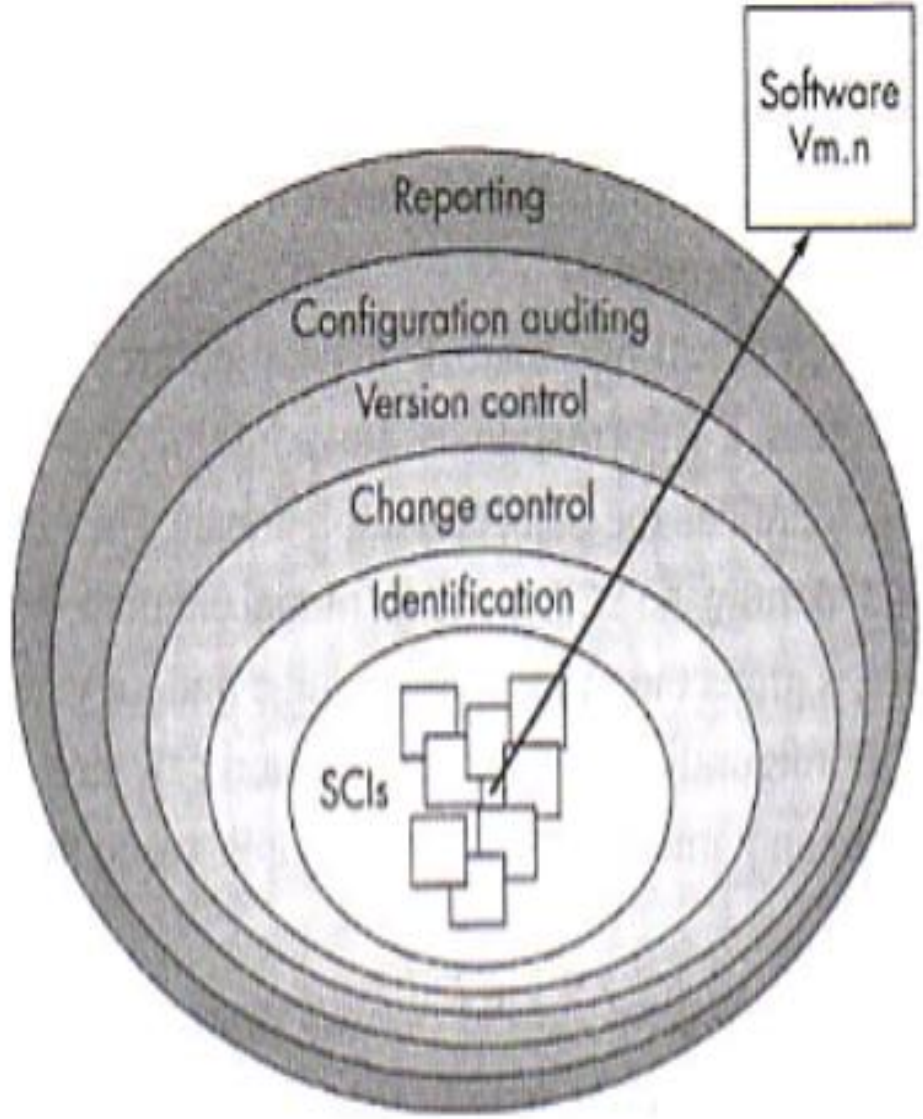
- COCOMO (Constructive Cost Model) is a regression model based on LOC, i.e. number of Lines of Code. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.

5.4 Software Configuration Management:

- SCM is the task of tracking and controlling changes in the software part of the larger cross-disciplinary field of configuration management.
- SCM practices include revision control and the establishment of baselines. If something goes wrong, SCM can determine what was changed and who changed it.
- When we develop software, the product (software) undergoes many changes in their maintenance phase we need to handle these changes effectively.

- Several individuals (programs) works together to achieve these common goals. This individual produces several work product e.g., Intermediate version of modules or test data used during debugging, parts of the final product.
- The elements that comprise all information produced as a part of the software process are collectively called a software configuration. As software development progresses, the number of Software Configuration elements grow rapidly.
- These are handled and controlled by SCM. Configuration of the product refers not only to the product's constituent but also to a particular version of the component.


Identification
Version Control
Change Control
Configuration Audit
Reporting
SCM Process


Software Vm.n
Reporting
Configuration auditing
Version control
Change control
Identification
SCIs

- The results/deliverables of a large software development effort typically consist of a large number of objects. E.g., Source code, design document, SRS document, test document, user's manual etc.,

- Software Configuration Management deals with effectively tracking and controlling the configuration of a software during its life cycle.

- Software revision vs version
- Necessity of software configuration management
- Configuration management activities.

Part- II : Software Reuse:

- Software reuse is the process of creating software systems from existing software rather than building software systems from scratch.

- A good software reuse process facilitates the increase of productivity, quality, and reliability, performance and the decrease of costs, effort, risk and implementation time.

Software Reuse: Introduction

- Something that was originally written for a different project and implementation will usually be recognized as reuse.
- Code reuse is the idea that a partial or complete computer program written at one time can be, should be, or is being used in another program written at a later time.
- The reuse of programming code is a common technique which attempts to save time and energy by reducing redundant work.

Software Reuse: Introduction

- Software products are expensive. Therefore, software project managers are always worried about the high cost of software development .
- A possible way to reduce development cost is to reuse parts from previously developed software.
- In addition to reduced development cost and time, reuse also leads to higher quality of the developed products since the reusable components are ensured to have high quality.
- A reuse approach that is of late gaining prominence is component-based development.

Software Reuse: Introduction

- Component-based software development is different from the traditional software development in the sense that software is developed by assembling software from off-the-shelf components.

- Software development with reuse is very similar to a modern hardware engineer building an electronic circuit by using standard types of ICs and other components.

5.5 What can be Reused?

- It is important to deliberate about the kinds of the artifacts* associated with software development that can be reused.
- Almost all artifacts associated with software development, including project plan and test plan can be reused.
- However, the prominent items that can be effectively reused are:
  - Requirements specification
  - Design
  - Code
  - Test cases
  - Knowledge

*one of many kinds of tangible by-products produced during the development of software. Some artifacts (e.g., use cases, class diagrams, and other Unified Modeling Language (UML) models, requirements and design documents) help describe the function, architecture, and design of software

What can be Reused?

- Knowledge is the most abstract development artifact that can be reused.
- Out of all the reuse artifacts, reuse of knowledge occurs automatically without any conscious effort in this direction.
- A planned reuse of knowledge can increase the effectiveness of reuse.
- For this, the reusable knowledge should be systematically extracted and documented. But, it is usually very difficult to extract and document reusable knowledge.

5.6 Why Almost No Reuse So Far?

- Engineers working in software development organizations often have a feeling that the current system that they are developing is similar to the last few systems built.
- However, no attention is paid on how not to duplicate what can be reused from previously developed systems.
- Everything is being built from the old system.
- The current system falls behind schedule and no one has time to figure out how the similarity between the current system and the systems developed in the past can be exploited.

Why Almost No Reuse So Far?

- Even those organizations which start the process of reusing programs
- Creation of components that are reusable in different applications is a difficult problem.
- It is very difficult to anticipate the exact components that can be reused across different applications.
- But, even when the reusable components are carefully created and made available for reuse, programmers prefer to create their own, because the available components are difficult to understand and adapt to the new applications.

Why Almost No Reuse So Far?

- The following observation is significant:
  - The routines of mathematical libraries are being reused very successfully by almost every programmer.
  - No one in their mind would think of writing a routine to compute sine or cosine.
  - Let us investigate why reuse of commonly used mathematical functions is so easy.
  - Everyone has clear ideas about what kind of argument should implement, the type of processing to be carried out and the results returned.
  - Secondly, mathematical libraries have a small interface.

5.7 Basic Issues in any Reuse Program

- The following are some of the basic issues that must be clearly understood for starting any reuse program:
  - Component creation.
  - Component indexing and storing.
  - Component search.
  - Component understanding.
  - Component adaptation.
  - Repository maintenance.

Basic Issues in any Reuse Program

- Component creation:
  - For component creation, the reusable components have to be first identified. Selection of the right kind of components having potential for reuse is important.
- Component indexing and storing
  - Indexing requires classification of the reusable components so that they can be easily searched when we look for a component for reuse.
  - The components need to be stored in a relational database management system (RDBMS) or an object-oriented database system (ODBMS) for efficient access when the number of components becomes large.

Basic Issues in any Reuse Program

- Component creation:
  - For component creation, the reusable components have to be first identified. Selection of the right kind of components having potential for reuse is important.

- Component indexing and storing

  - Indexing requires classification of the reusable components so that they can be easily searched when we look for a component for reuse.
  - The components need to be stored in a relational database management system (RDBMS) or an object-oriented database system (ODBMS) for efficient access when the number of components becomes large.

Basic Issues in any Reuse Program

- Component searching
  - The programmers need to search for right components matching their requirements in a database of components. To be able to search components efficiently, the programmers require a proper method to describe the components that they are looking for.
- Component understanding
  - To facilitate understanding, the components should be well documented and should do something simple.

Basic Issues in any Reuse Program

- Component adaptation
  - Often, the components may need adaptation before they can be reused, since a selected component may not exactly fit the problem at hand.
  - However, tinkering with the code is also not a satisfactory solution because this is very likely to be a source of bugs.
- Repository maintenance
  - A component repository once is created requires continuous maintenance.
  - New components, as and when created have to be entered into the repository.
  - The faulty components have to be tracked.

A Reuse Approach

- The reusable components need to be identified after every development project is completed.
- The reusability of the identified components has to be enhanced and these have to be cataloged into a component library.
- It must be clearly understood that an issue crucial to every reuse effort is the identification of reusable components.
- Domain analysis is a promising approach to identify and create reusable components.

A Reuse Approach

- Domain Analysis
- Reuse domain:
- Evolution of a reuse domain
- Component Classification
- Searching
- Repository Maintenance
- Reuse without Modifications.

5.8 Software Reuse: Reuse at Organization Level

- Reusability should be a standard part in all software development activities including specification, design, implementation, test, etc.
- Ideally, there should be a steady flow of reusable components.
- Extracting reusable components from projects that were completed in the past is difficulty, while extracting a reusable component from an ongoing project—typically, the original developers are no longer available for consultation.
- Reusability ranges from items whose reusability is immediate to those items whose reusability is highly improbable.
- Achieving organization-level reuse requires:
  Assess of an item's potential for reuse.
  Refine the item for greater reusability.
  Enter the product in the reuse repository.

Reuse at Organization Level

Assessing a product's potential for reuse

- Assessment of a components reuse potential can be obtained from an analysis of a questionnaire circulated among the developers.
- The questionnaire can be devised to assess a component's reusability.
- The programmers working in similar application domain can be used to answer the questionnaire about the product's reusability.
- Depending on the answers given by the programmers, either the component be taken up for reuse as it is, it is modified and refined before it is entered into the reuse repository, or it is ignored

Reuse at Organization Level

A sample questionnaire to assess a component's reusability is :

1. Is the component's functionality required for implementation of systems in the future?
2. How common is the component's function within its domain?
3. Would there be a duplication of functions within the domain if the component is taken up?
4. Is the component hardware dependent?
5. Is the design of the component optimized enough?
6. If the component is non-reusable, then can it be decomposed to yield some reusable components?
7. Can we parametrise a non-reusable component so that it becomes reusable?

Reuse at Organization Level

Refining products for greater reusability:

- For a product to be reusable, it must be relatively easy to adapt it to different contexts.
- The following refinements may be carried out:
  - Name generalization
  - Operation generalization
  - Exception generalization
  - Handling portability problems

- Also, programs use some function libraries, which may not be available on all host machines.
- A portability solution to overcome these problems is shown in below figure

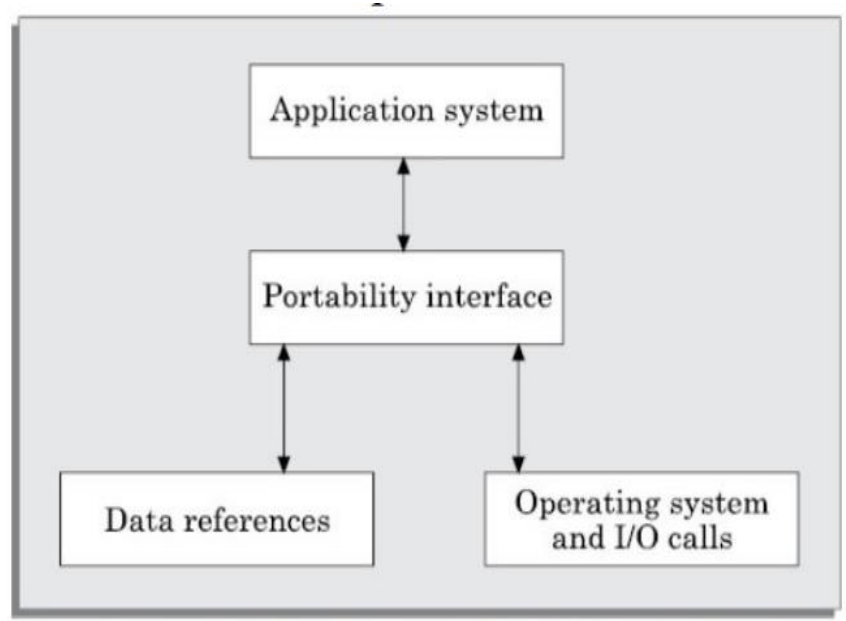


Fig: Improving reusability of a component by using portability interface

➢Also, all platform-related calls should be routed through the portability interface.

- Current State of Reuse:
- In spite of all the shortcomings of the state-of-the-art reuse techniques, it is the experience of several organizations that most of the factors inhibiting an effective reuse program are non-technical.
- Organizations are often hesitant to provide an open access to the reuse repository for the fear of the reuse components finding a way to their competitors

***