

## Software Design

### Unit: III

Software Engineering  
KCS 601

( B Tech 6<sup>th</sup> Sem)



BISWARUP DUTTA  
Assistant Professor  
Department of CSE

- Course Objective
- Course Outcomes
- CO-PO Mapping
- CO-PSO Mapping
- Syllabus
- Prerequisite
- Basic Concept of Software Design,
- Architectural Design
- Low Level Design
- Coupling and Cohesion

- Design Strategies
- Software Measurement and Metrics
- Cyclomatic Complexity Measures
- Video Links
- Daily Quiz
- Weekly Assignment
- MCQ
- Old Question Papers
- Expected Questions for University Exam
- Summary
- References

## Course Objective(unit-3)

- An understanding of implementation issues such as modularity and coding standards.
- Design, implement, and evaluate software-based systems, components, or programs of varying complexity that meet desired needs, satisfy realistic constraints, and demonstrate accepted design and development principles.

# Syllabus

Unit	TOPIC
I	Introduction: Introduction to Software Engineering, Software Components, Software Characteristics, Software Crisis, Software Engineering Processes, Similarity and Differences from Conventional Engineering Processes, Software Quality Attributes. Software Development Life Cycle (SDLC) Models: Water Fall Model, Prototype Model, Spiral Model, Evolutionary Development Models, Iterative Enhancement Models.
II	Software Requirement Specifications (SRS): Requirement Engineering Process: Elicitation, Analysis, Documentation, Review and Management of User Needs, Feasibility Study, Information Modelling, Data Flow Diagrams, Entity Relationship Diagrams, Decision Tables, SRS Document, IEEE Standards for SRS. Software Quality Assurance (SQA): Verification and Validation, SQA Plans, Software Quality Frameworks, ISO 9000 Models, SEI-CMM Model.
III	Software Design: Basic Concept of Software Design, Architectural Design, Low Level Design: Modularization, Design Structure Charts, Pseudo Codes, Flow Charts, Coupling and Cohesion Measures, Design Strategies: Function Oriented Design, Object Oriented Design, Top-Down and Bottom-Up Design. Software Measurement and Metrics: Various Size Oriented Measures: Halstead's Software Science, Function Point (FP) Based Measures, Cyclomatic Complexity Measures: Control Flow Graphs.

# Syllabus

Unit	TOPIC
<b>IV</b>	Software Testing: Testing Objectives, Unit Testing, Integration Testing, Acceptance Testing, Regression Testing, Testing for Functionality and Testing for Performance, TopDown and BottomUp Testing Strategies: Test Drivers and Test Stubs, Structural Testing (White Box Testing), Functional Testing (Black Box Testing), Test Data Suit Preparation, Alpha and Beta Testing of Products. Static Testing Strategies: Formal Technical Reviews (Peer Reviews), Walk Through, Code Inspection, Compliance with Design and Coding Standards.
<b>V</b>	Software Maintenance and Software Project management: Software as an Evolutionary Entity, Need for Maintenance, Categories of Maintenance: Preventive, Corrective and Perfective Maintenance, Cost of Maintenance, Software Re- Engineering, Reverse Engineering. Software Configuration Management Activities, Change Control Process, Software Version Control, An Overview of CASE Tools. Estimation of Various Parameters such as Cost, Efforts, schedule/Duration, Constructive Cost Models (COCOMO), Resource Allocation Models, Software Risk Analysis and Management.

# Objective of Topics

TOPIC	Objective
Software Design	To Understand the basic concept of design
Low Level Design	Study of Low Level Design
Coupling and Cohesion Measures	To Understand and compare the Coupling and Cohesion
Design Strategies	To examine the different design Strategies
Test Data Suit Preparation, Alpha and Beta	Study of Software Measurement and Metrics
Cyclomatic Complexity Measures	To find the Cyclomatic Complexity

At the end of the Course, the student will be able

Course Outcomes (CO)		Bloom's Knowledge Level (KL)
NCS601.1	Explain various software characteristics and analyze different software Development Models.	K1, K2
NCS601.2	Demonstrate the contents of a SRS and apply basic software quality assurance practices to ensure that design, development meet or exceed applicable standards.	K1, K2
NCS601.3	Compare and contrast various methods for software design	K2, K3
NCS601.4	Formulate testing strategy for software systems, employ techniques such as unit testing, Test driven development and functional testing.	K3
NCS601.5	Manage software development process independently as well as in teams and make use of Various software management tools for development, maintenance and analysis.	K3

## CO-PO Correlation Matrices

Correlation levels are taken 1, 2 and 3 as defined below:

**1:** Slight (Low)      **2:** Moderate (Medium)      **3:** Substantial (High)

Software Engineering (Code: KCS-601)									Year of Study: 2020-21			
CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
<b>C601.1</b>	2	3	3	3	2	-	-	-	-	-	3	3
<b>C601.2</b>	3	3	3	3	3	-	-	-	-	-	2	3
<b>C601.3</b>	3	2	3	2	2	-	-	-	-	-	3	3
<b>C601.4</b>	2	2	2	2	3	3	-	3	3	-	3	3
<b>C601.5</b>	2	2	3	2	3	3	-	3	-	3	3	3

# CO-PO and PSO Mapping

## Program Specific Outcomes and Course Outcomes Mapping

CO	PSO1	PSO2	PSO3	PSO4
CO1	3	3	-	3
CO2	3	3	2	3
CO3	3	3	-	3
CO4	3	3	-	3
CO5	3	3	-	3

\*3= High

\*2= Medium

\*1=Low

## Topic mapping with CO

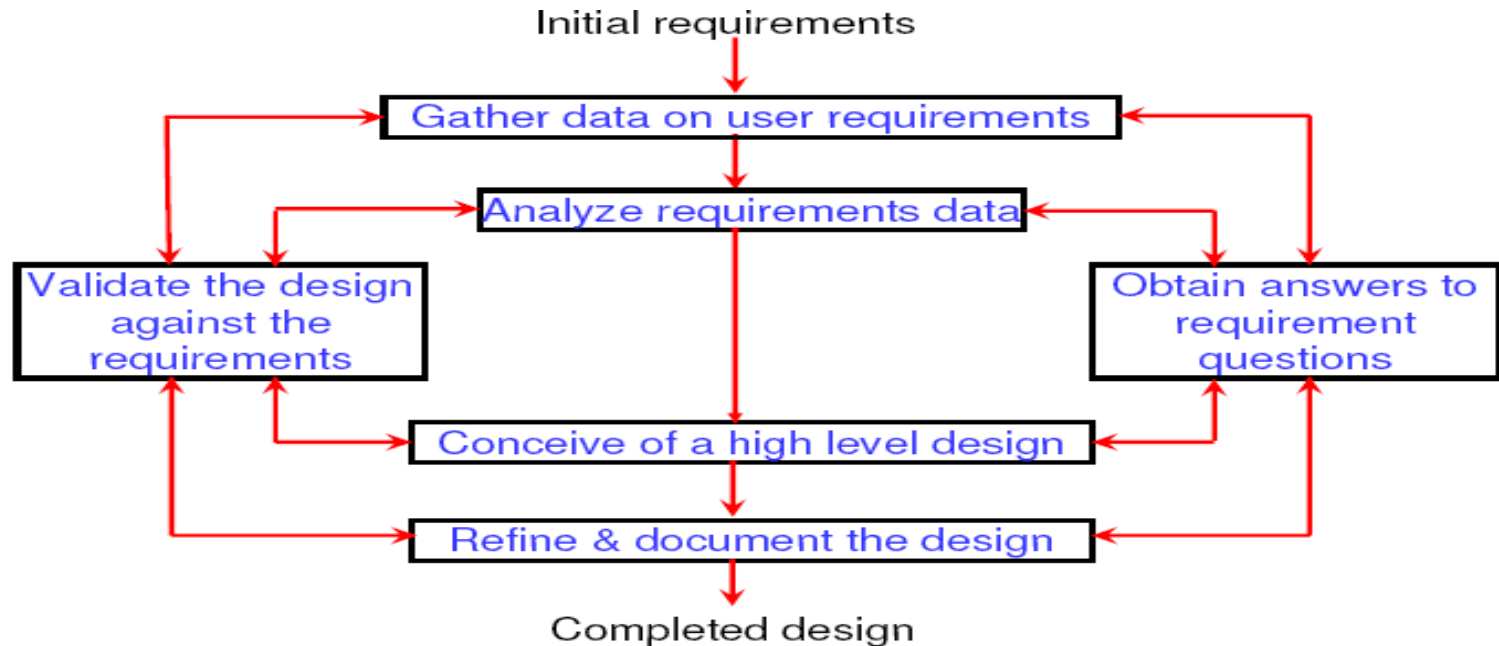
TOPIC	CO
Software Design: Basic Concept of Software Design, Architectural Design,	CO3
Low Level Design: Modularization, Design Structure Charts, Pseudo Codes, Flow Charts,	CO3
Coupling and Cohesion Measures	CO3
Design Strategies: Function Oriented Design, Object Oriented Design, Top-Down and Bottom-Up Design.	CO3
Software Measurement and Metrics: Various Size Oriented Measures: Halstead's Software Science, Function Point (FP) Based Measures,	CO3
Cyclomatic Complexity Measures: Control Flow Graphs.	CO3

## Prerequisite and Recap

- Basic Programming Skills
- Innovative Thinking.
- Enthusiasm to learn Management concepts.

## Software Design (CO3)

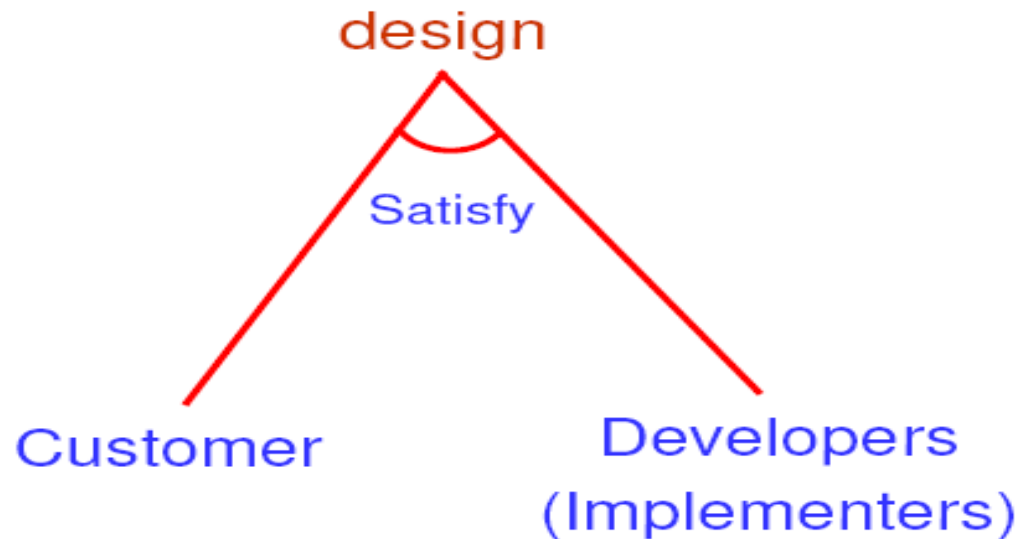
- In this process(phase) designer plans “How” a s/w system should be produced as per customer requirements.
- SRS tell us “What” a system does. Design Process tell us “How” a s/w system work.
- Designing of a s/w system means determining how requirements are realized and result in a s/w design document(SDD).
- Framework of design is given below:



- s/w design process involves the transformation of ideas into details implementation description, with good satisfying the s/w requirement.
- First we produce conceptual design that tells customer exactly what the system will do.
- Once conceptual design is approved by customer it translate into details design.

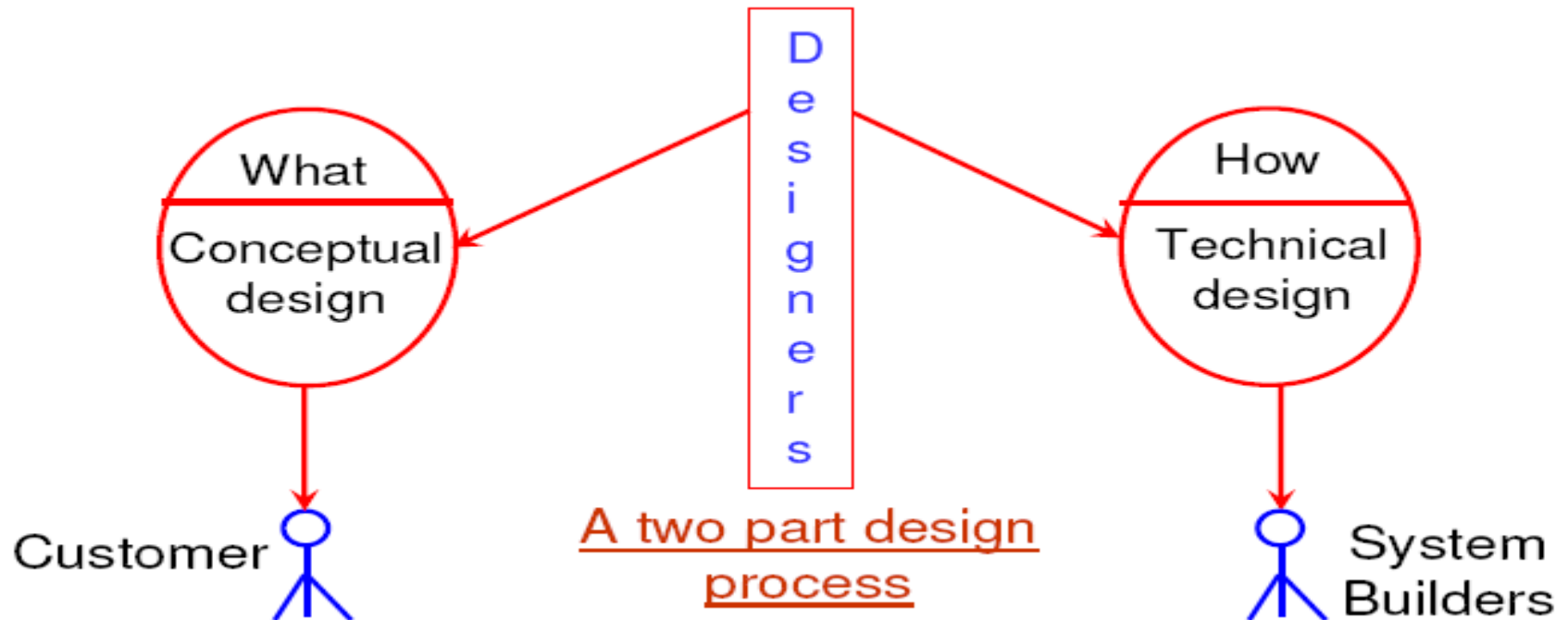
# Software Design

- It start with initial requirement and ends with final design.
- Data is gathered on user requirement and analyze accordingly.
- A High level design is prepared after answering question requirements.
- Design is validated against requirements on regular basis.
- Design is refined in every cycle and finally it is documented to produce SDD(S/w design document)



# Conceptual design and Technical design (CO3)

## Conceptual Design and Technical Design



A two part design process

## Conceptual design and Technical design

- To transform req. into working system, designer must satisfy both customer and system(S/W) builder.
- A design is a two part iterative process
  - 1. Conceptual design or preliminary design or high level design:** it the identification of different modules and control relationship among them and definition of the interface among these module. It is also called program structure or S/W architecture.
  - 2. Technical design or detailed design or low level design:** it describe h/w configuration, s/w needs, communication interface, I/O of the system, data structure and algorithms of different modules are designed

# Conceptual design and Technical design

## Conceptual design

- ✓ Where will the data come from ?
- ✓ What will happen to data in the system?
- ✓ How will the system look to users?
- ✓ What choices will be offered to users?
- ✓ What is the timings of events?
- ✓ How will the reports & screens look like?

## Technical design

- ✓ Hardware configuration
- ✓ Software needs
- ✓ Communication interfaces
- ✓ I/O of the system
- ✓ Software architecture
- ✓ Network architecture
- ✓ Any other thing that translates the requirements in to a solution to the customer's problem.

## Outcome of a design process

- Out come of a design process:
  - i. Different modules required.
  - ii. Control relationship among modules.
  - iii. Interfaces among different modules.
  - iv. Data structure of individual modules.
  - v. Algorithm required to implement module.
- Characteristic of a good design:
  - i. Correctness: implement all functionality identified in SRS doc.
  - ii. Understandability: interpret by coder and tester.
  - iii. Efficiency: should be efficient.
  - iv. Maintainability: easy amenable to change

# Design Transformation

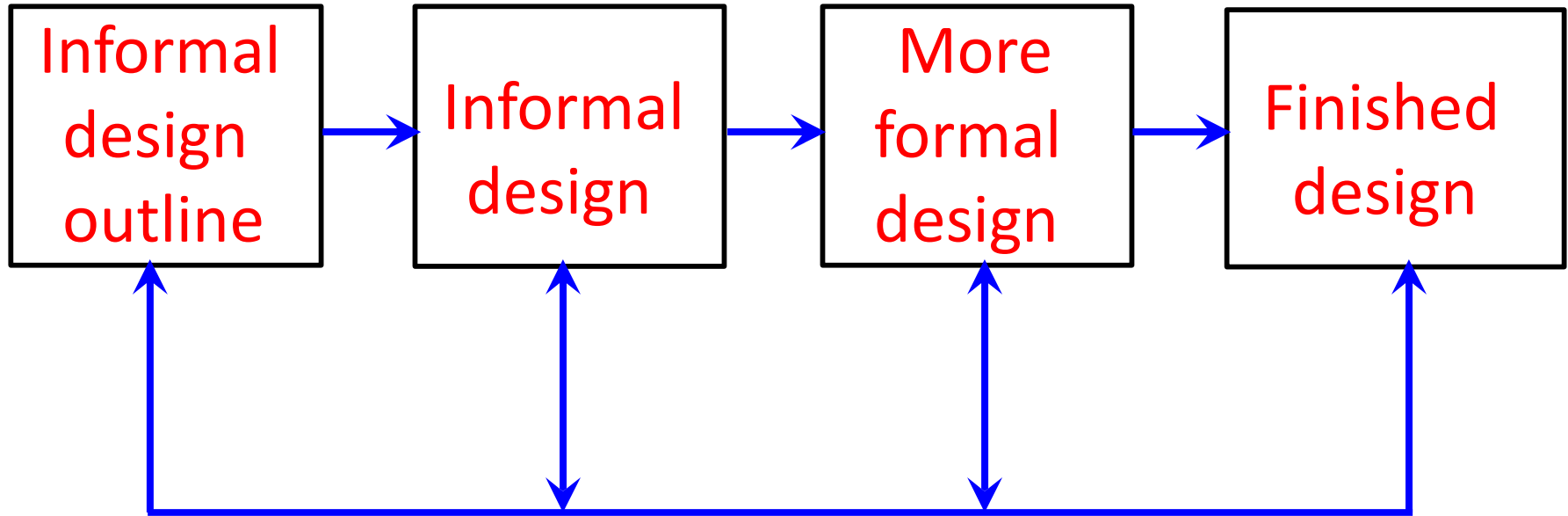


Fig: The transformation of an informal design to a detailed design.

## Architectural Design (CO3)

- Outcome of AD/high level design is called **program structure** or **s/w architecture**.
- Problem is decomposed into a set of modules and manage them with cohesive and low coupling.
- Control relationship and interfaces among various modules are identified.
- Many notations such as structure chart, UML etc. are used in high level design.

# Architectural Design

- AD methods have various alternative arch. Style of designing a system. these are:
  1. **Data flow architecture** : flow of data in the system or sub systems.
  2. **Object oriented architecture** : it involves class and objects
  3. **Layered architecture**: define no. of layered. Outer layered handle functionality of user interface and inner most layer handle interaction with the H/W.
  4. **Data centric architecture**: it involves the use of a central database operation such as inserting, updating, etc. in the form of a table.

## Low level Design (CO3)

- **Technical design or detailed design or low level design**
  - **Modularization**
  - **Coupling**
  - **Cohesion**
  - **Flow chart**
  - **Pseudo codes**

- **Modularization:**

- Complex system may be divided into simpler pieces called modules.
- A system is composed of modules is called modular.
- Module is treated separately If different modules have either no or little interactions with each other.
- Cohesion and coupling are decide the degree of modularity.
- A design solution is considered to be highly modular if different modules in the solution have high cohesion and their inter module coupling are low.

## Properties of a module

- Well defined subsystem
- Well defined purpose
- Can be separately compiled and stored in a library.
- Module can use other modules
- Module should be easier to use than to build
- Simpler from outside than from the inside.

**Modularity** is the single attribute of software that allows a program to be intellectually manageable. It enhances design clarity, which in turn eases implementation, debugging, testing, documenting, and maintenance of software product.

# Modularity

- A system considered modular if it consists of discrete component so that each component can be implemented separately and a change to one component has minimal impact on other component.
- No of module grow, the effort associated with integration the module also grows.
- So Under modularity and over modularity in a software should be avoided.

# Modularity

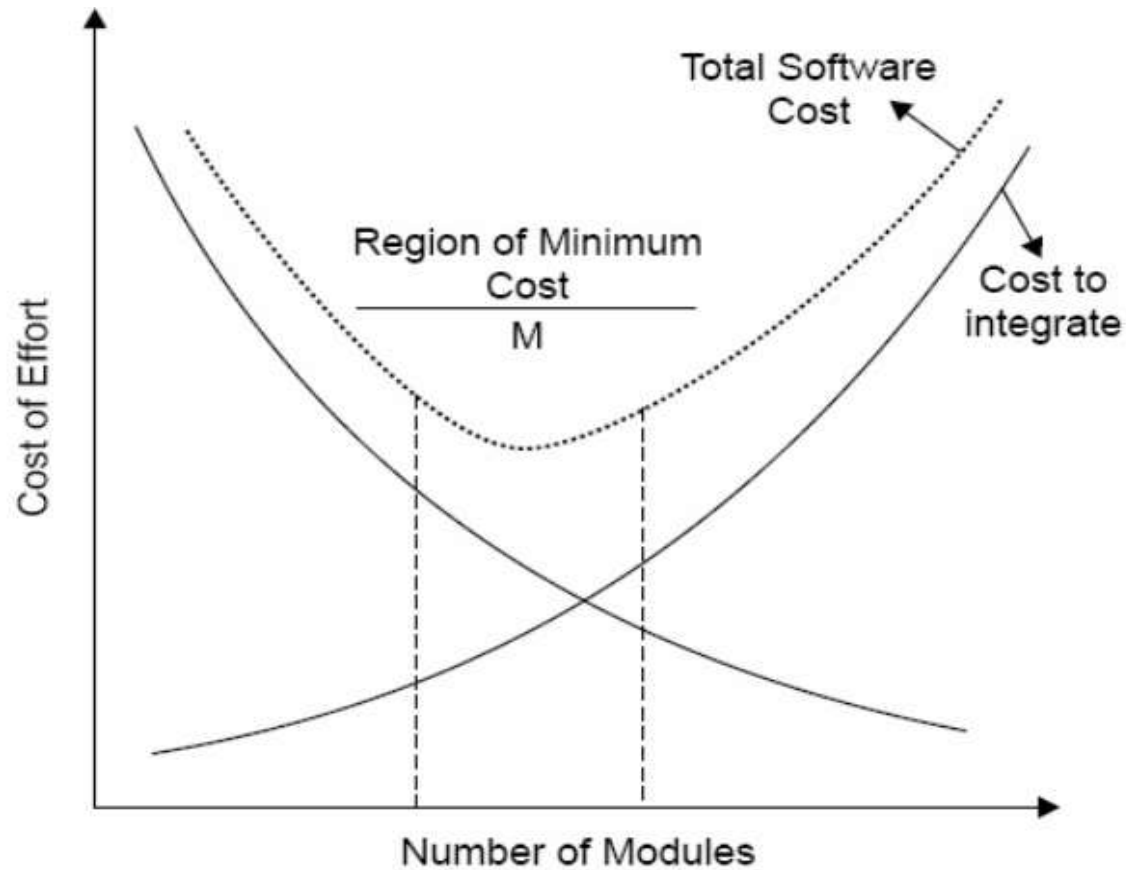
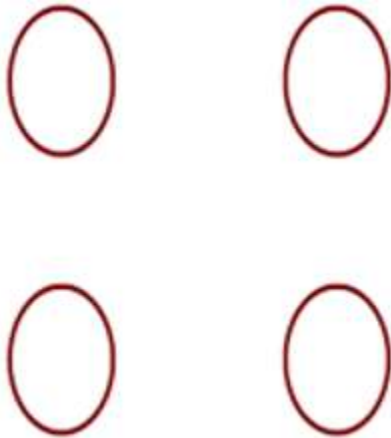


Fig. : Modularity and software cost

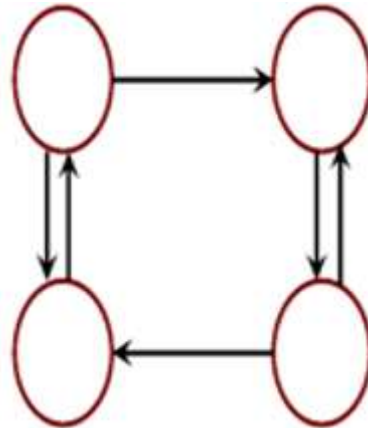
## Module Coupling (CO3)

- Coupling is the measure of the degree of interdependence(or no of interconnection) between modules.
- Two modules with high coupling are strongly interconnected i.e. dependent on each other.
- Two modules with low coupling are not dependent on one another.
- A good design will have low coupling.
- Design with High coupling will have more error.
- Loose coupling minimize the interdependence amongst modules.
- **Low coupling can be achieve by:**
  - eliminating unnecessary relationships
  - reducing the number of necessary relationships
  - easing the 'tightness' of necessary relationships

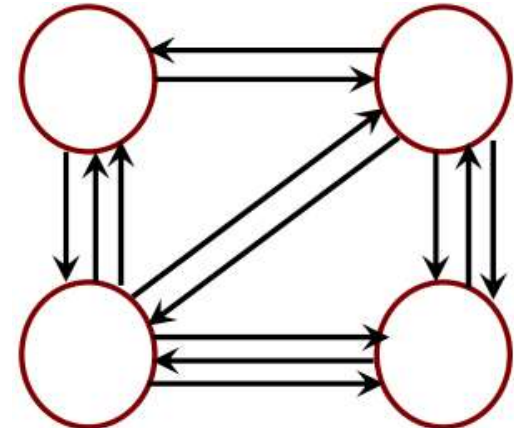
# Module Coupling



Uncoupled : no dependencies



Loosely coupled:  
some dependencies



Highly coupled:  
many dependencies

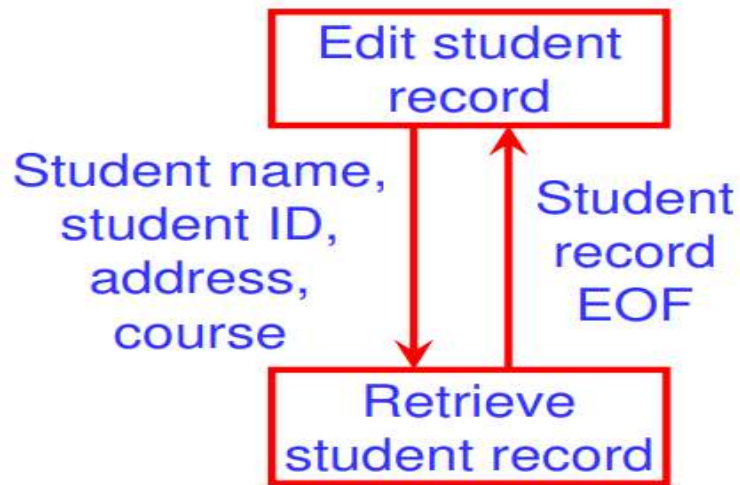
# Module Coupling

Loose coupling can be achieved as:

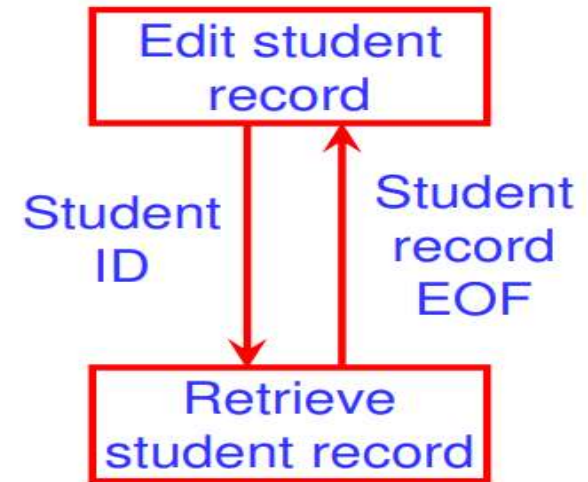
- Controlling the no of parameter passed amongst modules.
- Avoid passing undesired data to calling module.
- Maintain parent/child relationship between calling and called modules.
- Pass data, not the control information.

## Example of coupling

Consider the example of editing a student record in a 'student Information system'.




Poor design: Tight Coupling



Good design: Loose Coupling

## Types of Module Coupling

Data coupling	<b>Best</b>
Stamp coupling	
Control coupling	
External coupling	
Common coupling	
Content coupling	<b>Worst</b>

Strength of coupling from lowest coupling(best) to highest coupling(worst).

# Types of Module Coupling

## Data coupling

- Modules communicate by parameters Each parameter is an elementary piece of data Each parameter is necessary to the communication. Nothing extra is needed

- **Data coupling problems**

Too many parameters - makes the interface difficult to understand and possible error to occur

**Tramp data** - data 'traveling' across modules before being used

## Stamp coupling

Occurs when A composite data(data structure) is passed between modules

### **problem in stamp coupling**

Internal structure contains data not used Bundling - grouping of unrelated data into an artificial structure

# Types of Module Coupling

## Control coupling

- A module controls the logic of another module through the control information(flag)
- Controlling module needs to know how the other module works – not

## External Coupling

- Occurs when another module is external to the s/w being developed or to a particular type of hardware.
- It is based on communication to external tools and devices.

# Types of Module Coupling

## Common coupling

Use of global data as communication between modules

Making a change to the common data means tracing back to all the modules which access that data to evaluate the effect of changes.

### problem in Common coupling

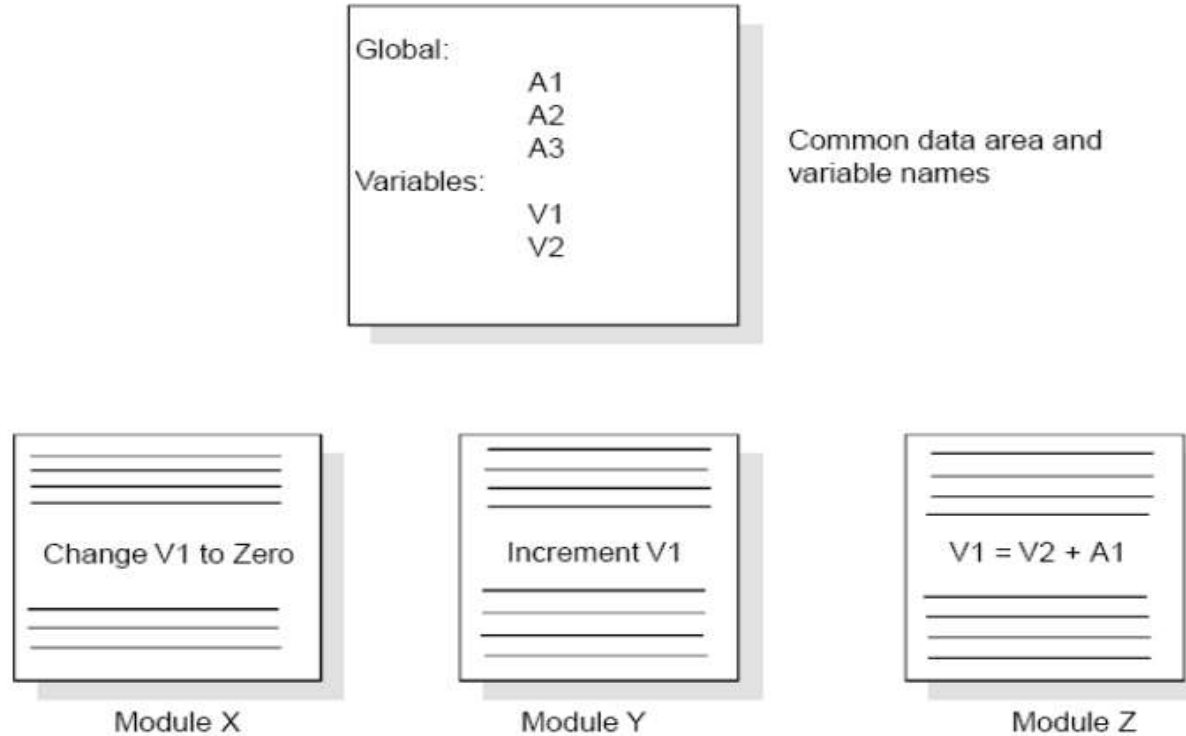
- ripple effect

- inflexibility

- difficult to understand the use of data

It can difficult to determine which value is responsible for having set a variable to a particular values

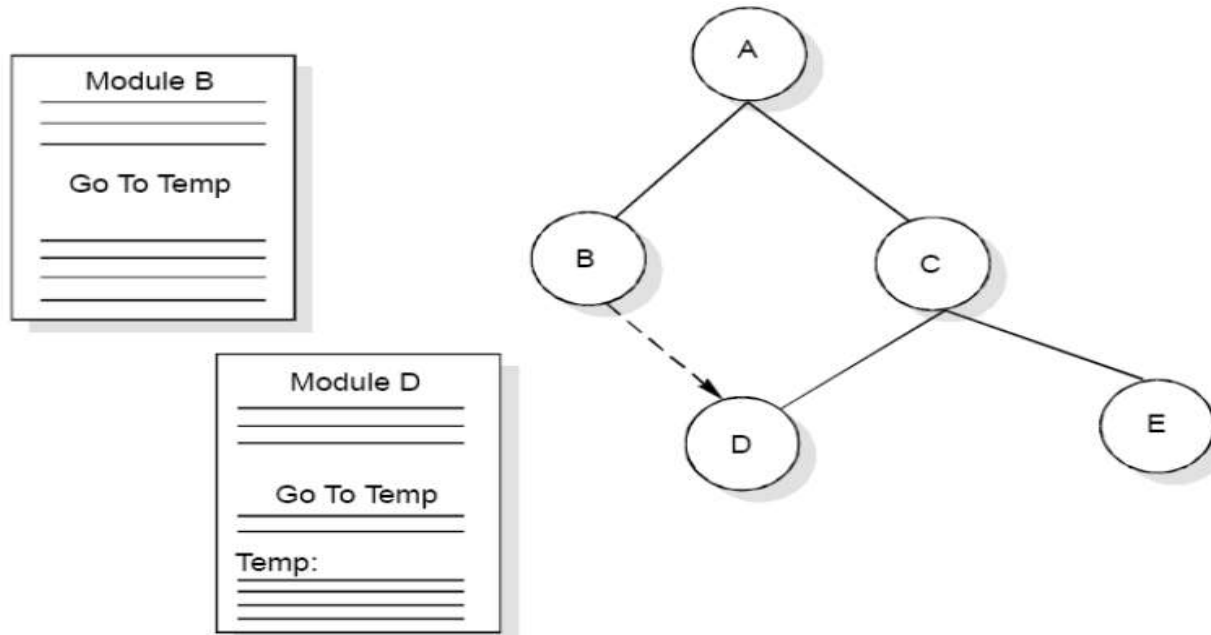
## Example of common coupling



# Types of Module Coupling

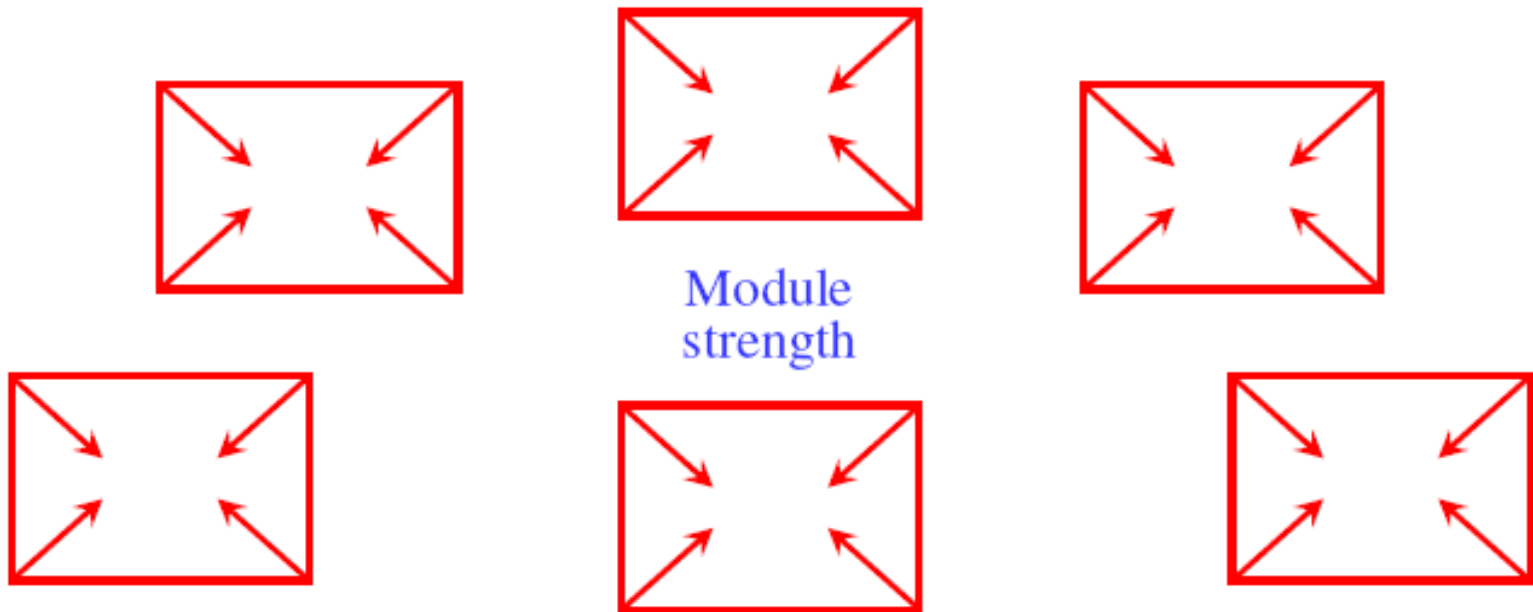
## Content coupling

Content coupling occurs when module A changes data of module B or when control is passed from one module to the middle of another. In Fig. , module B branches into D, even though D is supposed to be under the control of C. A module refers to the inside of another module. Branch into another module Refers to data within another module Changes the internal workings of another module Mostly by low-level languages



## Module Cohesion (CO3)

- Cohesion is a measure of the degree(strength) to which the elements of a module are functionally related.



Module  
strength

Cohesion=Strength of relations within modules


# Cohesion

- Definition
  - The degree to which all elements of a module are directed towards a single task.
  - The degree to which all elements directed towards a task are contained in a single component.
  - The degree to which all responsibilities of a single class are related.
- Internal glue with which Module is constructed
- All elements of module are directed toward and essential for performing the same task.
- **Elements**: instructions, groups of instructions, data definition, call of another module.
- Strong cohesion will reduce relations between modules - minimize coupling

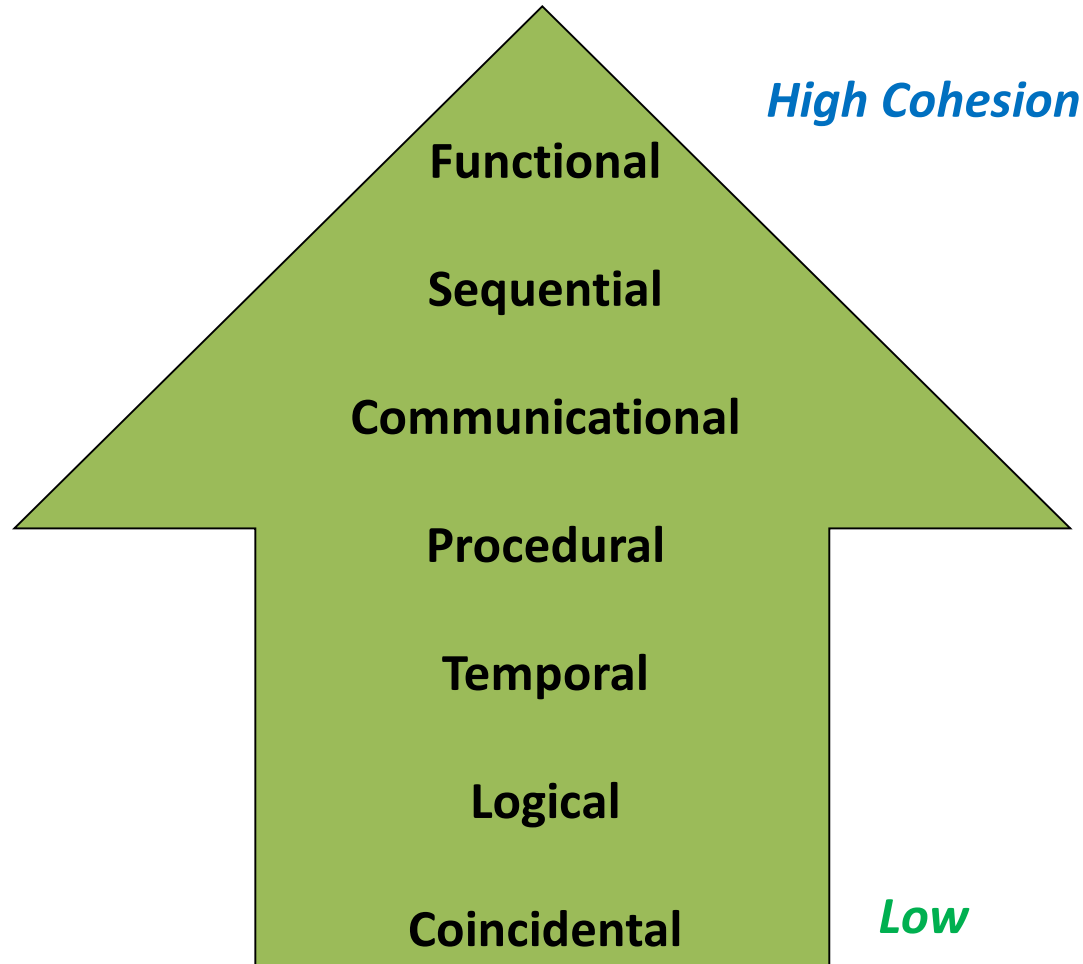
# Type of Cohesion

- ❖ Functional cohesion
- ❖ Sequential cohesion
- ❖ Procedural cohesion
- ❖ Temporal cohesion
- ❖ Logical cohesion
- ❖ Coincident cohesion

# Type of Module Cohesion

Functional Cohesion	Best (high)
Sequential Cohesion	
Communicational Cohesion	
Procedural Cohesion	
Temporal Cohesion	
Logical Cohesion	
Coincidental Cohesion	Worst (low)

# Type of Cohesion



# Coincidental Cohesion

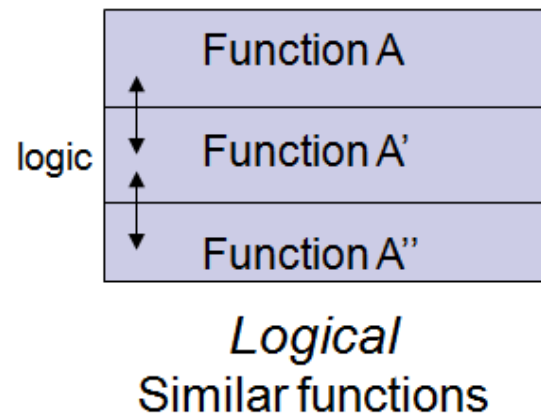
- Parts of the module are unrelated (unrelated functions, processes, or data)
- Parts of the module are only related by their location in source code.
- Elements needed to achieve some functionality are scattered throughout the system.
- EX.
  1. Print next line
  2. Reverse string of characters in second argument
  3. Add 7 to 5<sup>th</sup> argument
  4. Convert 4<sup>th</sup> argument to float

Function A	
Function B	Function C
Function D	Function E

*Coincidental*  
Parts unrelated

# Logical Cohesion

- Elements of module are related logically and not functionally.
- Several logically related elements are in the same module and one of the elements is selected by the client module. Ex.
  - A module reads inputs from tape, disk, and network.
  - All the code for these functions are in the same module.
  - Operations are related, but the functions are significantly different.



# Temporal Cohesion

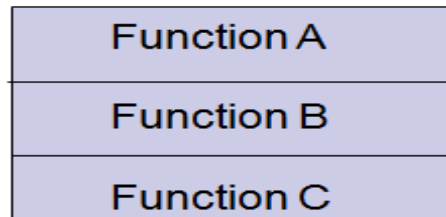
- Elements are related by timing involved
- Elements are grouped by when they are processed.
- Example: An exception handler that
  - Closes all open files
  - Creates an error log
  - Notifies user
  - Lots of different activities occur, all at same time

Time $t_0$
Time $t_0 + X$
Time $t_0 + 2X$

*Temporal*  
Related by time

# Procedural Cohesion

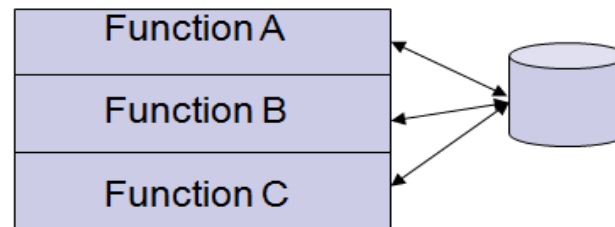
- Elements of a module are related only to ensure a particular order of execution.
- Actions are still weakly connected and unlikely to be reusable.
- Example:
  - ...
  - Write output record
  - Read new input record
  - Pad input with spaces
  - Return new record
  - ...



*Procedural*  
Related by order of functions

# Communicational Cohesion

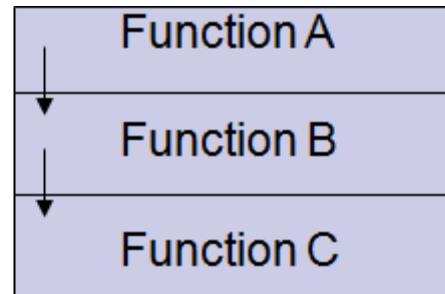
- Functions performed on the same data or to produce the same data.
- Examples:
  - Update record in data base and send it to the printer
    - Update a record on a database
    - Print the record



*Communicational*  
Access same data

## Sequential Cohesion

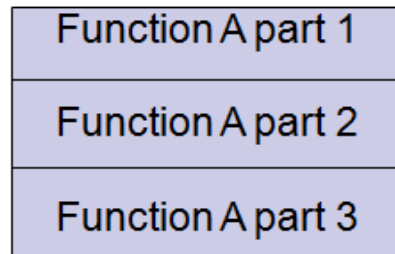
- Def: The output of one part is the input to another.
- *Data flows* between parts (different from procedural cohesion)
- Occurs naturally in functional programming languages



*Sequential*  
Output of one is input to another

# Functional Cohesion

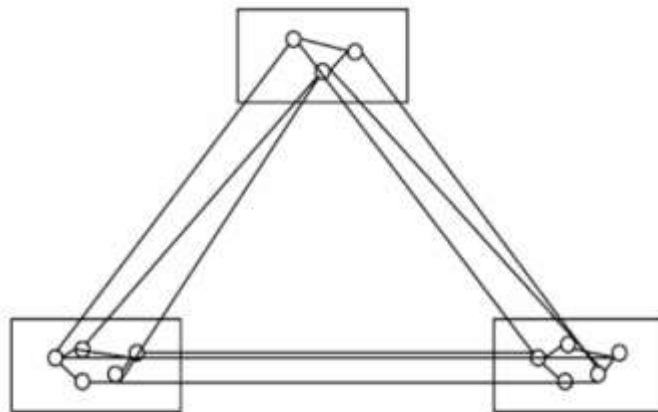
- Every essential element to a single computation is contained in the module.
- Every element in the module is essential to the computation.
- What is a functionally cohesive module?
  - One that not only performs the task for which it was designed but
  - it performs only that function and nothing else.



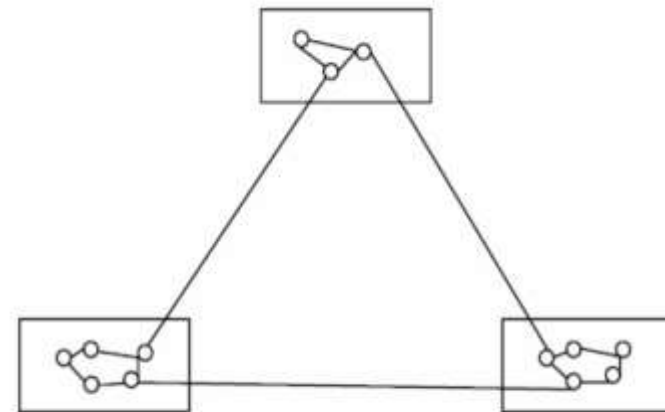
*Functional*  
Sequential with complete, related functions

## Relationship between cohesion & coupling (CO3)

If the software is not properly modularized, a host of seemingly trivial enhancement or changes will result into death of the project. Therefore, a software engineer must design the modules with goal of high cohesion and low coupling.



High Coupling



Low Coupling

## Flow Chart (CO3)




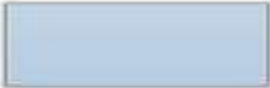

- It is convenient technique to represent the flow of control in a program.
- It is a graphical representation of an algorithms.
- It also help during testing and modifications in the programs.
- Advantage of Flow chart:
  - It help in following ways:
    - Synthesis(Systematic combination of different elements.)
    - Coding
    - Debugging
    - Communication
    - Testing

# Flow Chart

## Limitation:

- No standard way that should be included in flow chart.
- Difficult to draw, if algorithms has complex branches and loops.
- Time consuming process for large complex problems
- Difficult to include any new step in existing flow chart.

# Basic Component of Flow Chart

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

## Example 1

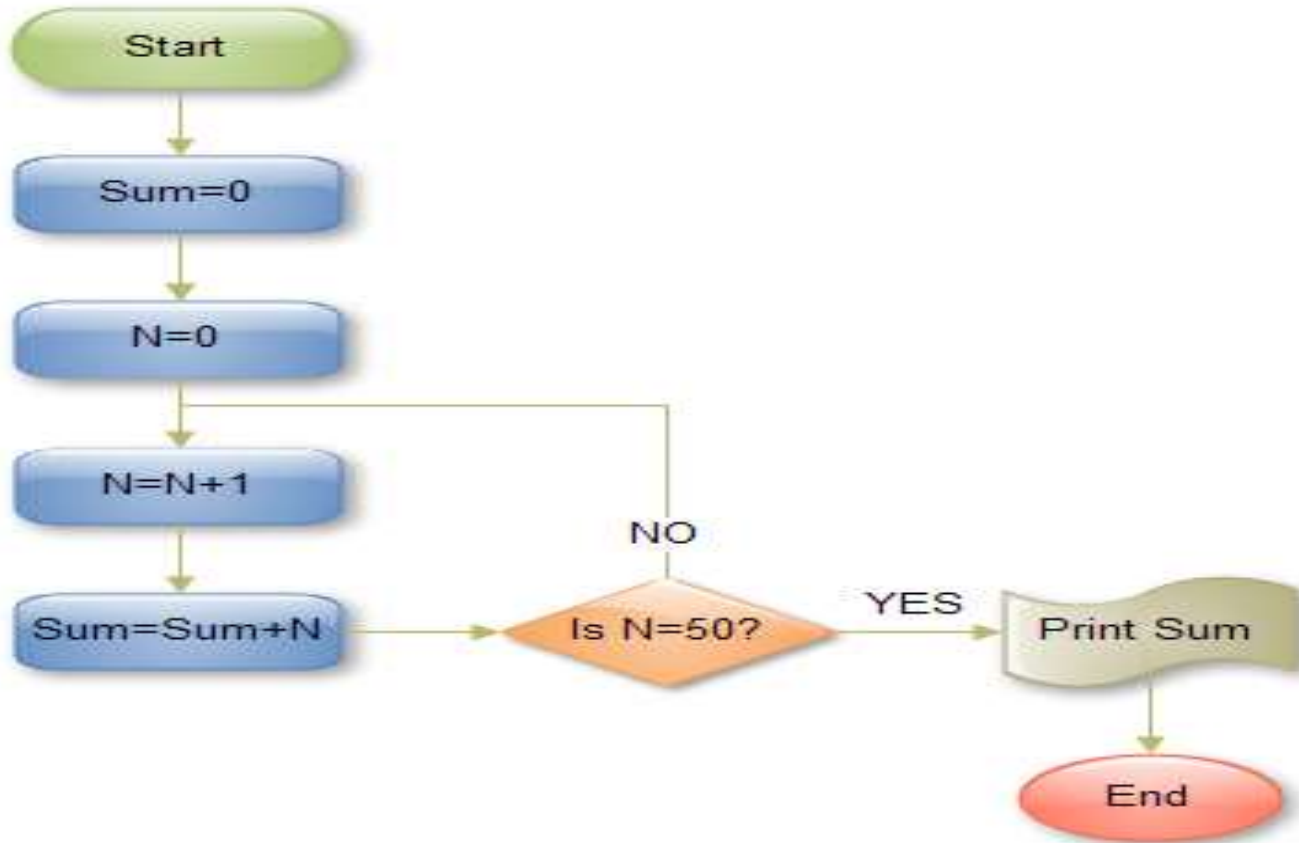


Fig. 1 Flowchart for the sum of the first 50 natural numbers

## Example 2

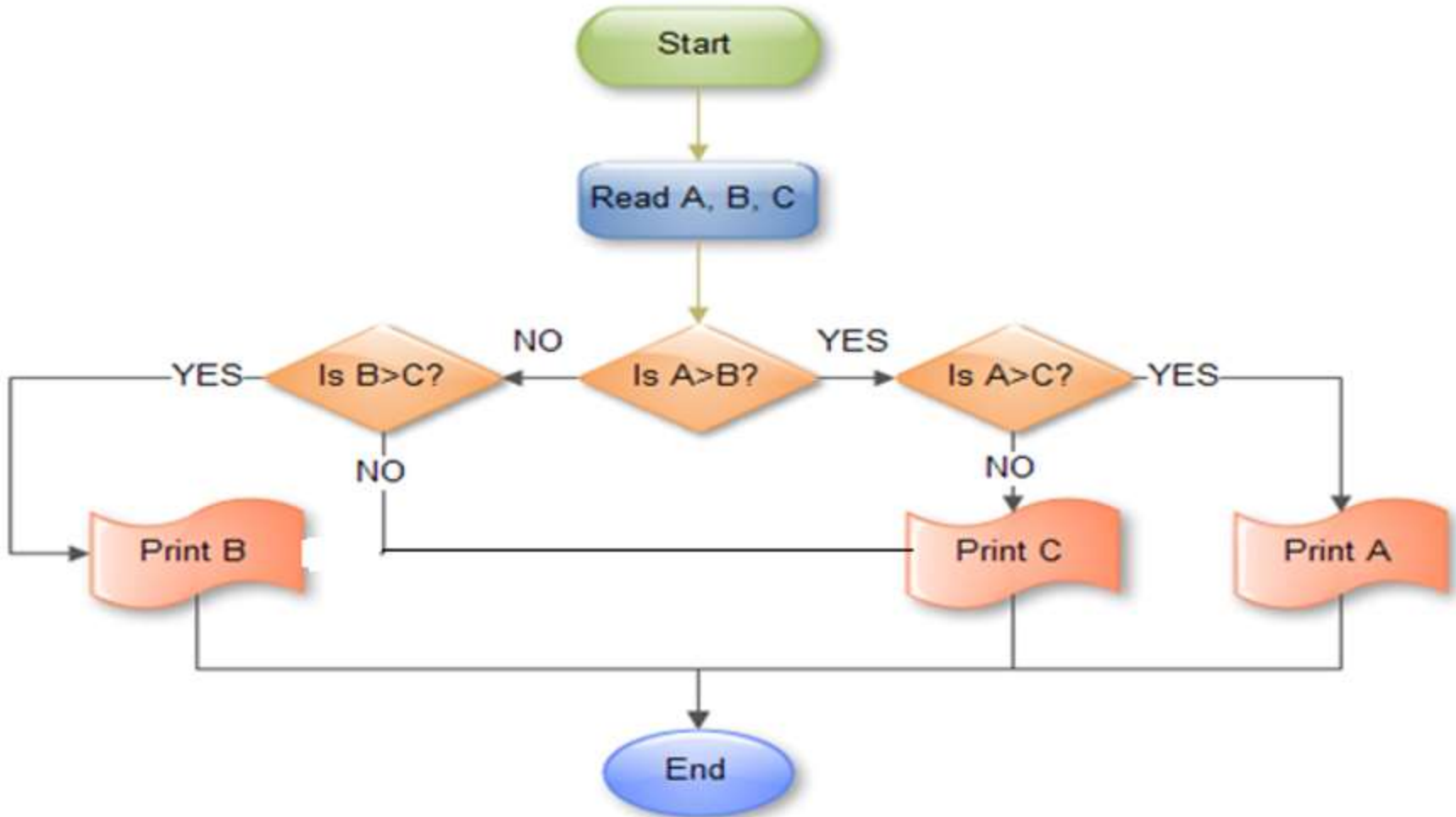


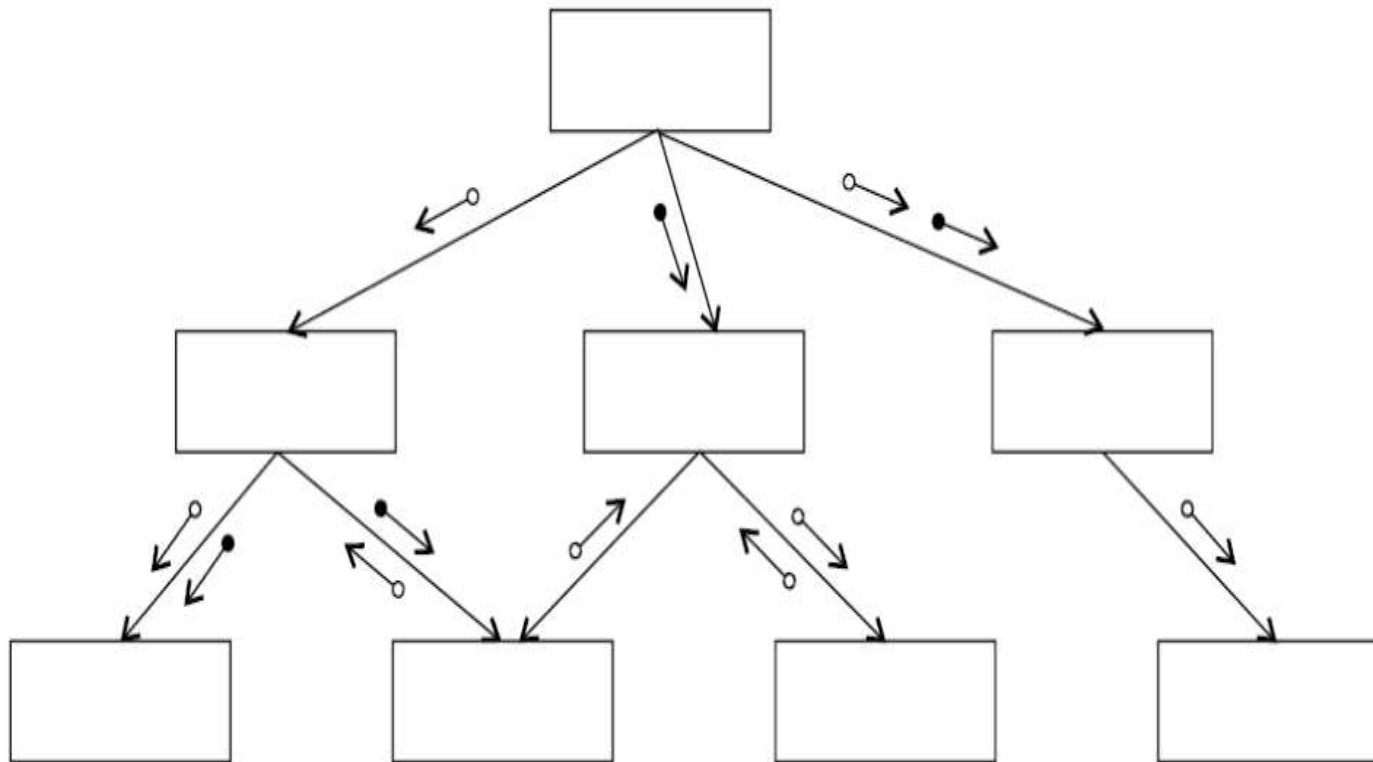
Fig 2 Flowchart for finding out the largest of three numbers

## Structure Chart (CO3)

- It represent the s/w architecture that can be easily implemented using some prog. language.
- It partitions a system into black box(input/output)
- Connection between modules are represented by lines between rectangular boxes.
- Component are generally read from top to bottom and left to right.
- Top level modules called lower level modules.
- It has only one module on the top called root.

# Structure Chart

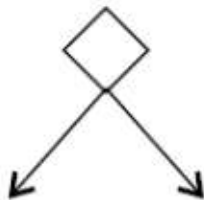
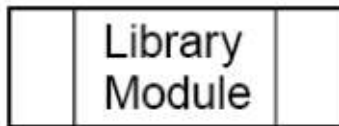
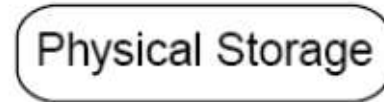
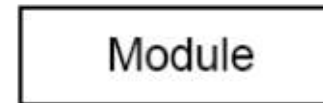
It partitions a system into block boxes. A black box means that functionality is known to the user without the knowledge of internal design.



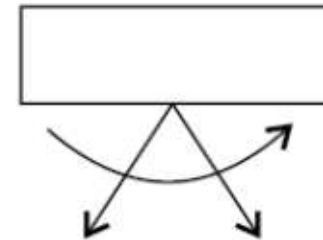
# Notation of Structure Chart

○ → Data

● → Control



Diamond symbol  
for conditional call  
of module

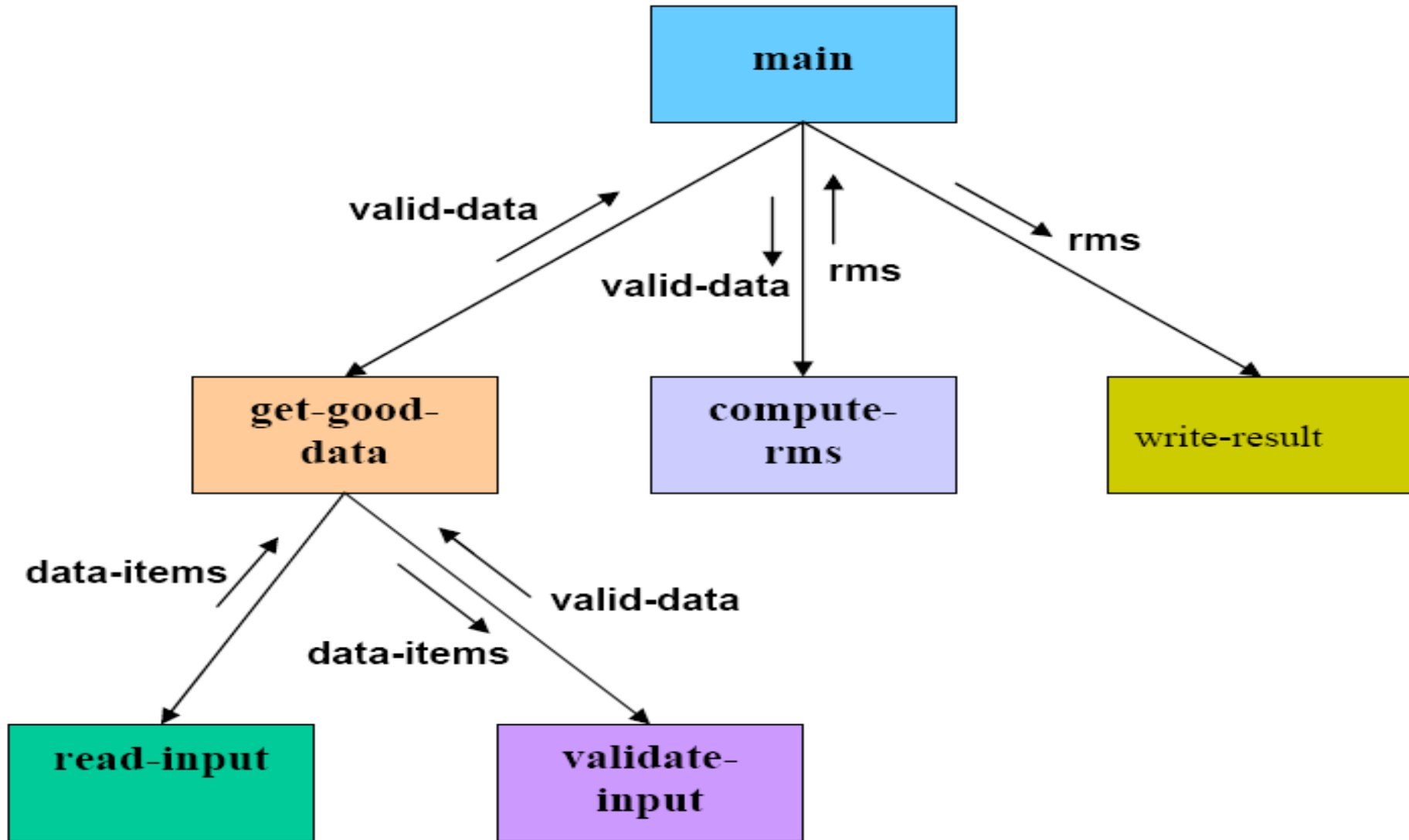


Repetitive call  
of module

## Notation of Structure Chart

- **Rectangular boxes:** Represents a module.
- **Module invocation arrows:** Control is passed from one module to another module in the direction of the connecting arrow.
- **Data flow arrows:** Arrows are annotated with data name; named data passes from one module to another module in the direction of the arrow.
- **Library modules:** Represented by a rectangle with double edges.
- **Selection:** Represented by a diamond symbol.
- **Repetition:** Represented by a loop around the control flow arrow.

## Structure Chart Ex.1: rms calculator



## Pseudo Code (CO3)

- Pseudo code notation can be used in both the preliminary (high level) and detailed design (low level) phases.
- Code are effective and building block for actual program.
- Using pseudo code, the designer describes system characteristics using short, concise, English language phrases that are structured by key words such as **If-Then-Else**, **While-Do**, and **End**.
- **Advantage of pseudo code (compare to flow chart)**
  - Easy to convert in programming language.
  - Easy to modify.
  - Require less time and effort to write it.
  - Easy to write than writing a program in programming language.
- **Disadvantage of pseudo code:**
  - No graphical representation of program logic.
  - No standard rule are follows to writing pseudo code

# Pseudo Code

- Problem: find smallest number among three variables
  1. read values of a, b and c variables
  2. if (a < b)
    - {
      - if(a < c)
      - print "a is small"
      - else print "c is small"
  - else if(b < c)
    - print "b is small"
    - else print "c is small"
  3. end.

## STRATEGY OF DESIGN (CO3)

A good system design strategy is to organize the program modules in such a way that are easy to develop and latter to, change. Structured design techniques help developers to deal with the size and complexity of programs. Analysts create instructions for the developers about how code should be written and how pieces of code should fit together to form a program. It is important for two reasons:

- ❖ First, even pre-existing code, if any, needs to be understood, organized and pieced together.
- ❖ Second, it is still common for the project team to have to write some code and produce original programs that support the application logic of the system

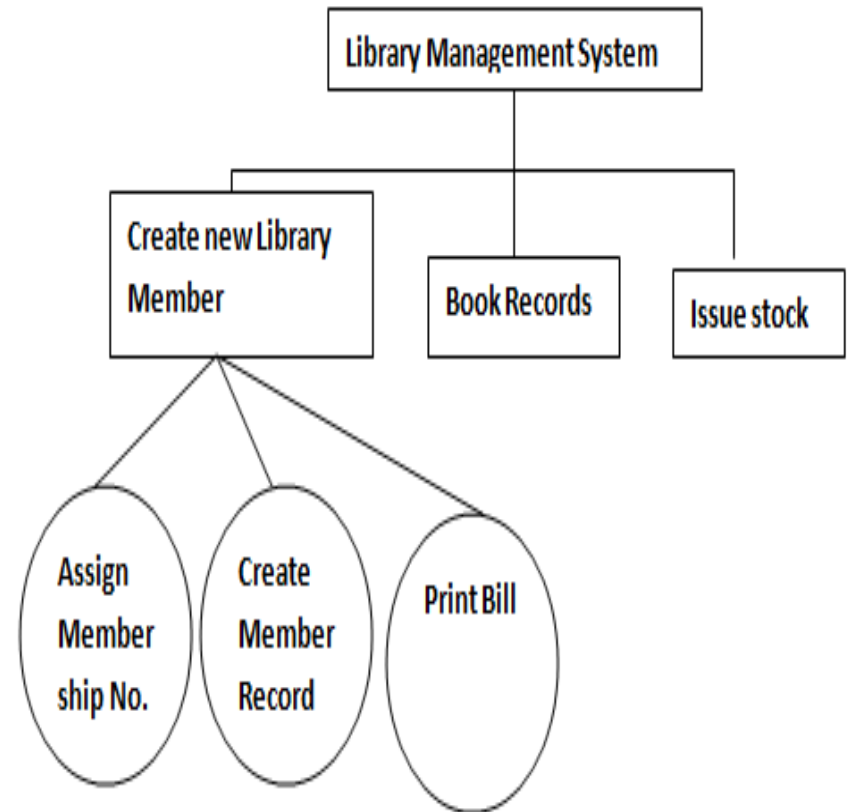
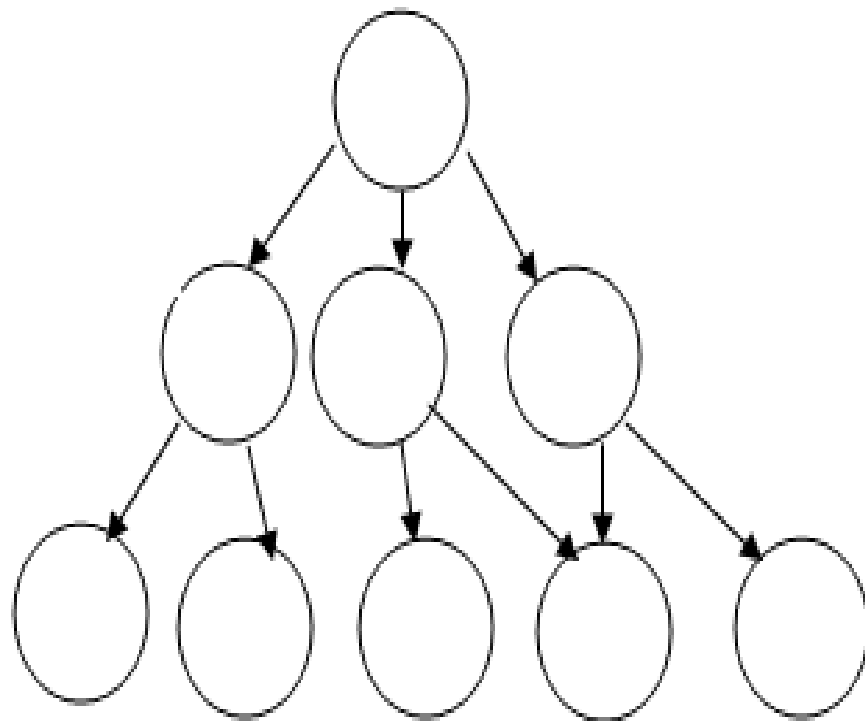
# STRATEGY OF DESIGN

- ❖ Top down design
- ❖ Bottom up design.
- ❖ Function oriented design.
- ❖ Object oriented design.

## Top Down Design (CO3)

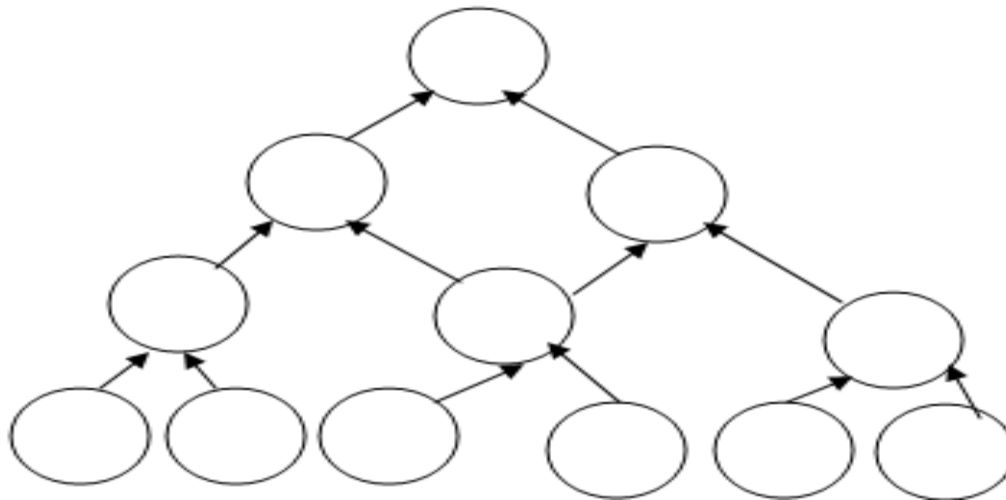
- Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics.
- Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy is achieved.
- Top-down design starts with a generalized model of system and keeps on defining the more specific part of it. When all components are composed the whole system comes into existence.
- Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown.

# Top Down Design



## Bottom up Design (C03)

- The bottom up design model starts with most specific and basic components.
- It proceeds with composing higher level of components by using basic or lower level components. It keeps creating higher level components until the desired system is not evolved as one single component.



## Bottom-up Design

- **Bottom-up strategy** is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.
- **Hybrid Design** :Both, top-down and bottom-up approaches are not practical individually. Instead, a good combination of both is used.

## Function oriented Design (CO3)

Function Oriented design is an approach to software design where the design is decomposed into a set of interacting units where each unit has a clearly defined function. Thus, system is designed from a functional viewpoint

# Function Oriented Design

Consider the example of scheme interpreter. Top-level function may look like:

While (not finished)

```
{  
    Read an expression from the terminal;  
    Evaluate the expression;  
    Print the value;  
}
```

We thus get a fairly natural division of our interpreter into a “read” module, an “evaluate” module and a “print” module. Now we consider the “print” module and is given below:

Print (expression exp)

```
{  
    Switch (exp → type)  
    Case integer: /*print an integer*/  
    Case real:   /*print a real*/  
    Case list:   /*print a list*/  
    :::  
}
```

# Function oriented Design

- In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.
- Function oriented design inherits some properties of structured design where **divide and conquer methodology** is used.
- This design mechanism divides the whole system into smaller functions. These functional modules can share information among themselves by means of information passing and using information available globally.
- Another characteristic of functions is that when a program calls a function, the function changes the state of the program, which sometimes is not acceptable by other modules. Function oriented design works well where the system state does not matter and program/functions work on input rather than on a state.

- The whole system is seen as how data flows in the system by means of data flow diagram.
- DFD depicts how functions changes data and state of entire system.
- The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.
- Each function is then described at large.

## FOD notations

- For FOD, the design can be represented graphically and mathematically by following:
  1. Data flow diagram.
  2. Data dictionary.
  3. Structure chart.
  4. Pseudo code.

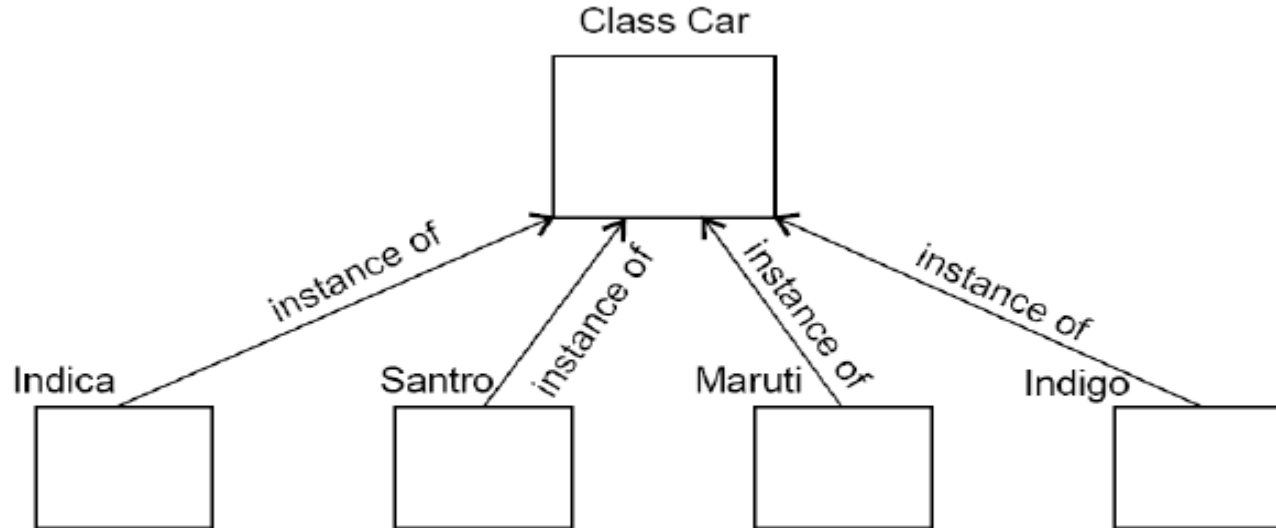
## Object Oriented concepts (CO3)

- Object oriented design is the result of focusing attention not on the function performed by the program, but instead on the data that are to be manipulated by the program.

### Important concepts of Object Oriented Design:

- **Objects** - All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes and methods to perform operations on the attributes.
- **Classes** - A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and Operations(methods), which defines the functionality of the object.

# Object Oriented Design



Indica, Santro, Maruti, Indigo are all instances of the class “car”

Class Square

Square	Name
Colour	} Attributes
Point[4]	
Set Colour()	} Operations
Draw()	

**The square class**

# Object Oriented concepts

## Attribute

An attributes is a data value held by the objects in a class.

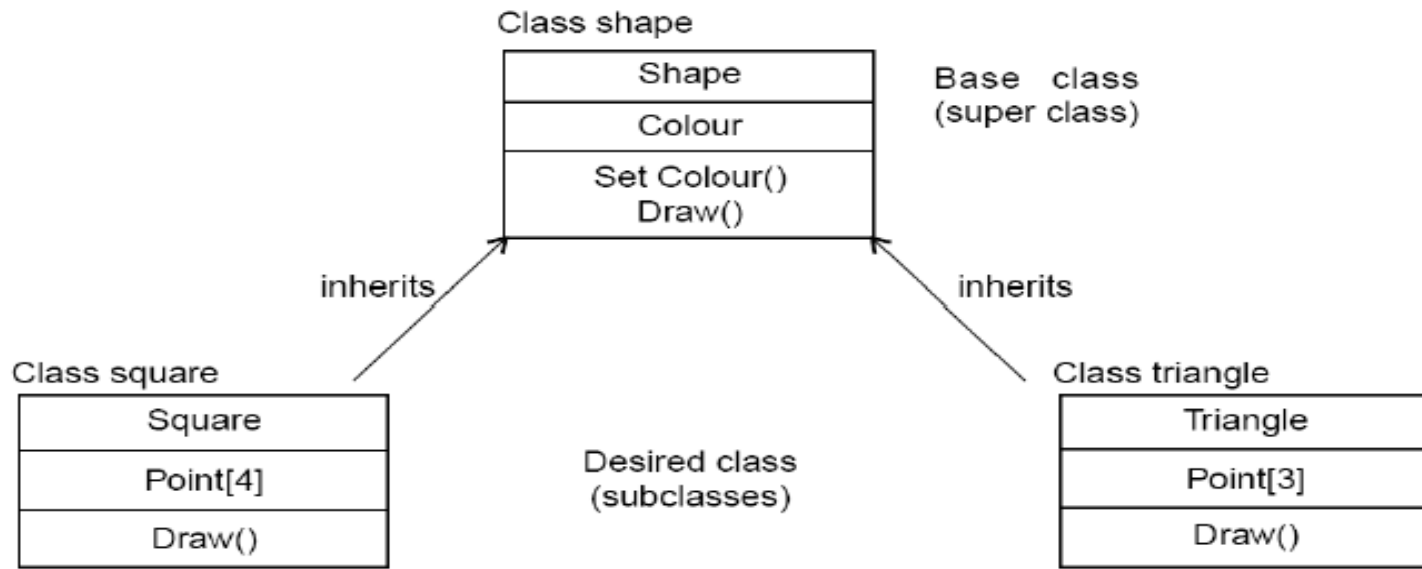
## Operation

An operation is a function or transformation that may be applied to or by objects in a class

## Inheritance -

OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance.

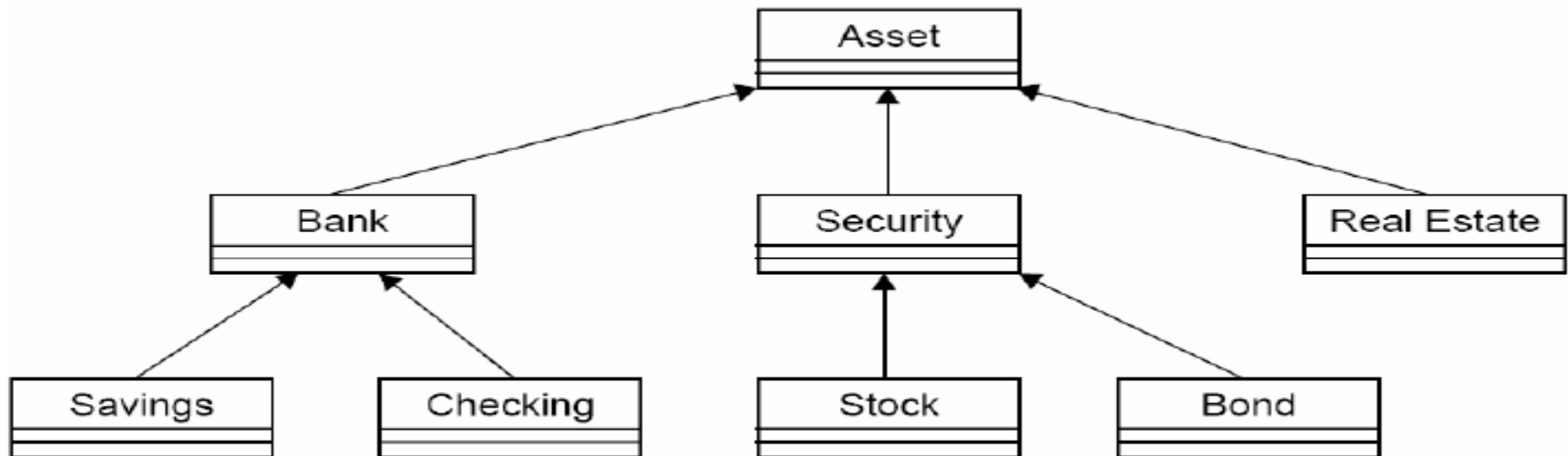
# Object Oriented concepts



Abstracting common features in a new class

# Object Oriented concepts

- **Encapsulation** - In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation restricts access of the data and methods from the outside world. This is called information hiding.
- **Polymorphism** - OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism.



## Hierarchy

# Design Process

- It may have the following steps involved:
- A solution design is created from requirement or previous used system and/or system sequence diagram.
- Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.
- Class hierarchy and relation among them is defined.
- Application framework is defined.

## Software Matrices and Measurement (CO3)

- It is the estimation of s/w project parameters such as:
  - Effort
  - Time duration completing the project
  - Total cost for developing the s/w project
- Project size is a measure of the problem complexity in term of effort and time require to develop the product.
- Two matrices are popularly used to estimate size:
  1. Lines of Code(LOC)
  2. Function Point(FP)
- 1. Lines of code:
  - it is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line.
  - This specifically includes all lines containing program header, declaration, and executable and non-executable statements

# Software Matrices and Measurement

## Size Estimation

### Lines of Code (LOC)

If LOC is simply a count of the number of lines then figure shown below contains 18 LOC .

When comments and blank lines are ignored, the program shown contains 17 LOC.

### Function for sorting an array

1.	int. sort (int x[], int n)
2.	{
3.	int i, j, save, im1;
4.	/*This function sorts array x in ascending order */
5.	If (n<2) return 1;
6.	for (i=2; i<=n; i++)
7.	{
8.	im1=i-1;
9.	for (j=1; j<=im; j++)
10.	if (x[i] < x[j])
11.	{
12.	Save = x[i];
13.	x[i] = x[j];
14.	x[j] = save;
15.	}
16.	}
17.	return 0;
18.	}

## Function Point Metric (CO3)

- It is a size measurement technique of a problem developed by **Alan Albrecht** in 1970.
- Conceptual idea behind it is that size of a s/w product is directly dependent on the no. of different function.
- Each function when invoked reads input data and transform it to the corresponding output data.
- s/w size is also dependent on :
  - no of files
  - No of interfaces.
- Interfaces refer to different mechanisms that need to support for data transfer with other external systems.

## Function Point Analysis (CO3)

The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units.

Inputs : information entering the system

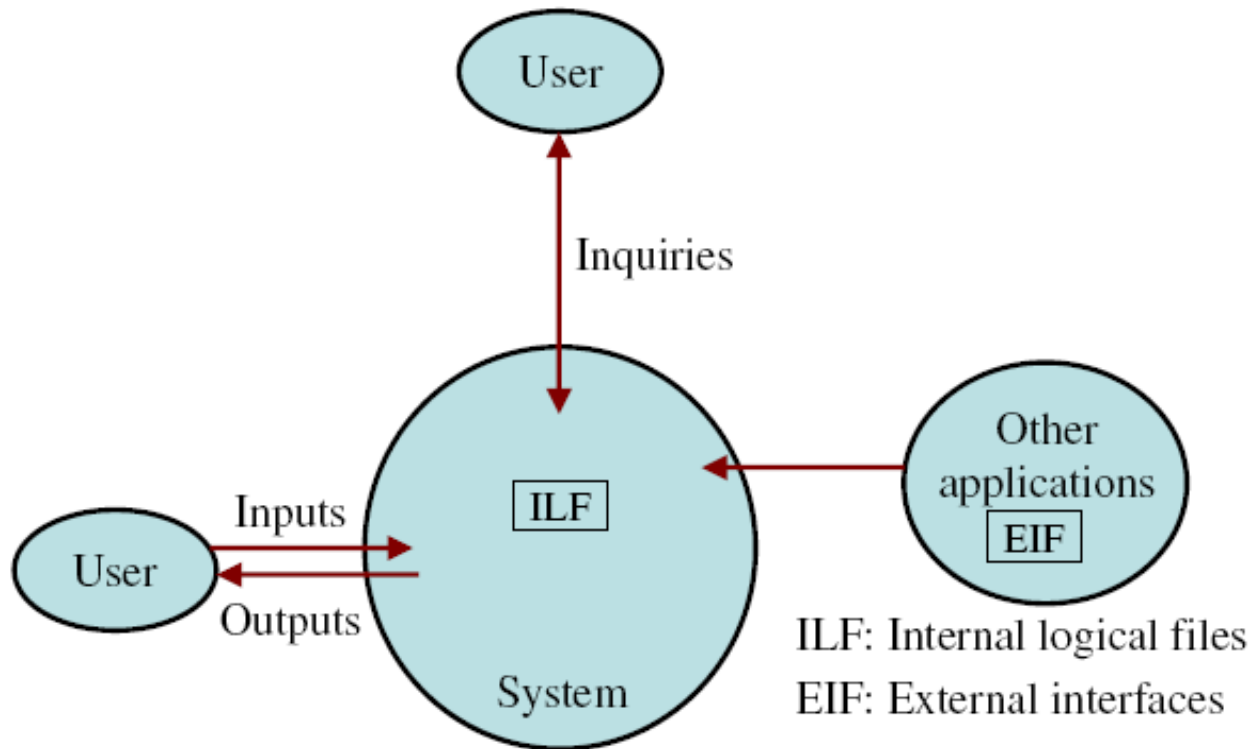
Outputs : information leaving the system

Enquiries : requests for instant access to information

Internal logical files : information held within the system

External interface files : information held by other system that is used by the system being analyzed

The FPA functional units are shown in figure given below:



FPA's functional units System

The five functional units are divided in two categories:

## **(i) Data function types**

**Internal Logical Files (ILF):** A user identifiable group of logical related data or control information maintained within the system.

**External Interface files (EIF):** A user identifiable group of logically related data or control information referenced by the system, but maintained within another system. This means that EIF counted for one system, may be an ILF in another system.

## (ii) Transactional function types

- **External Input (EI):** An EI processes data or control information that comes from outside the system. The EI is an elementary process, which is the smallest unit of activity that is meaningful to the end user in the business.
- **External Output (EO):** An EO is an elementary process that generate data or control information to be sent outside the system.
- **External Inquiry (EQ):**
  - An EQ is an elementary process that is made up to an input-output combination that results in data retrieval.

## Special Features of Function Point

- it is independent of the language(PL), tools, or methodologies used for implementation, data base management systems, processing hardware or any other data base technology.
- it can be estimated from requirement specification or design specification, thus making it possible to estimate development efforts in early phases of development.
- It is directly linked to the statement of requirements; any change of requirements can easily be followed by a re-estimate.
- it is based on the system user's external view of the system, non-technical users of the software system have a better understanding of what function points are measuring.

# Counting Function Point

## Counting function points

Functional Units	Weighting factors		
	Low	Average	High
External Inputs (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiries (EQ)	3	4	6
External logical files (ILF)	7	10	15
External Interface files (EIF)	5	7	10

# UFP table

UFP calculation table

Functional Units	Count Complexity			Complexity Totals	Functional Unit Totals
External Inputs (EIs)	<input type="text"/>	Low x 3	=	<input type="text"/>	<input type="text"/>
	<input type="text"/>	Average x 4	=	<input type="text"/>	
	<input type="text"/>	High x 6	=	<input type="text"/>	
External Outputs (EOs)	<input type="text"/>	Low x 4	=	<input type="text"/>	<input type="text"/>
	<input type="text"/>	Average x 5	=	<input type="text"/>	
	<input type="text"/>	High x 7	=	<input type="text"/>	
External Inquiries (EQs)	<input type="text"/>	Low x 3	=	<input type="text"/>	<input type="text"/>
	<input type="text"/>	Average x 4	=	<input type="text"/>	
	<input type="text"/>	High x 6	=	<input type="text"/>	
External logical Files (ILFs)	<input type="text"/>	Low x 7	=	<input type="text"/>	<input type="text"/>
	<input type="text"/>	Average x 10	=	<input type="text"/>	
	<input type="text"/>	High x 15	=	<input type="text"/>	
External Interface Files (EIFs)	<input type="text"/>	Low x 5	=	<input type="text"/>	<input type="text"/>
	<input type="text"/>	Average x 7	=	<input type="text"/>	
	<input type="text"/>	High x 10	=	<input type="text"/>	
Total Unadjusted Function Point Count					<input type="text"/>

$$UFP = \sum_{i=1}^5 \sum_{j=1}^3 Z_{ij} w_{ij}$$

Where  $i$  indicate the row and  $j$  indicates the column of Table 1

$w_{ij}$  : It is the entry of the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the table 1

$Z_{ij}$  : It is the count of the number of functional units of Type  $i$  that have been classified as having the complexity corresponding to column  $j$ .

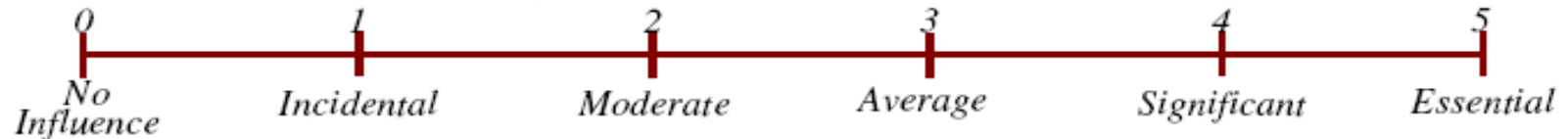
$$FP = UFP * CAF$$

Where CAF is complexity adjustment factor and is equal to  $[0.65 + 0.01 \times \sum F_i]$ . The  $F_i$  ( $i=1$  to 14) are the degree of influence and are based on responses to questions noted in table 3.

# Function Point

Computing function points.

Rate each factor on a scale of 0 to 5.



Number of factors considered ( $F_i$ )

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

# Function Point

Functions points may compute the following important metrics:

Productivity = FP / persons-months

Quality = Defects / FP

Cost = Rupees / FP

Documentation = Pages of documentation per FP

## Halestead's Software Science (CO3)

- Introduced by Maurice Howard Halstead in 1977.
- Halstead's "software science", essentially counting **operators** and **operands**
- Its goal was to identify measurable properties of software, and the relations between them. This is similar to the identification of measurable properties of matter (like the volume, mass, and pressure of a gas) and the relationships between them.

# Halestead's Software Science

- $\eta_1$  = the number of distinct operators
- $\eta_2$  = the number of distinct operands
- $N_1$  = the total number of operators
- $N_2$  = the total number of operands

From these numbers, several measures can be calculated:

- Program vocabulary:  $\eta = \eta_1 + \eta_2$
- Program length:  $N = N_1 + N_2$
- Calculated program length:  $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Program Volume:  $V = N \times \log_2 \eta$
- Difficulty : |

$$D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$$

- Effort:  $E = D \times V$

The difficulty measure is related to the difficulty of the program to write or understand.

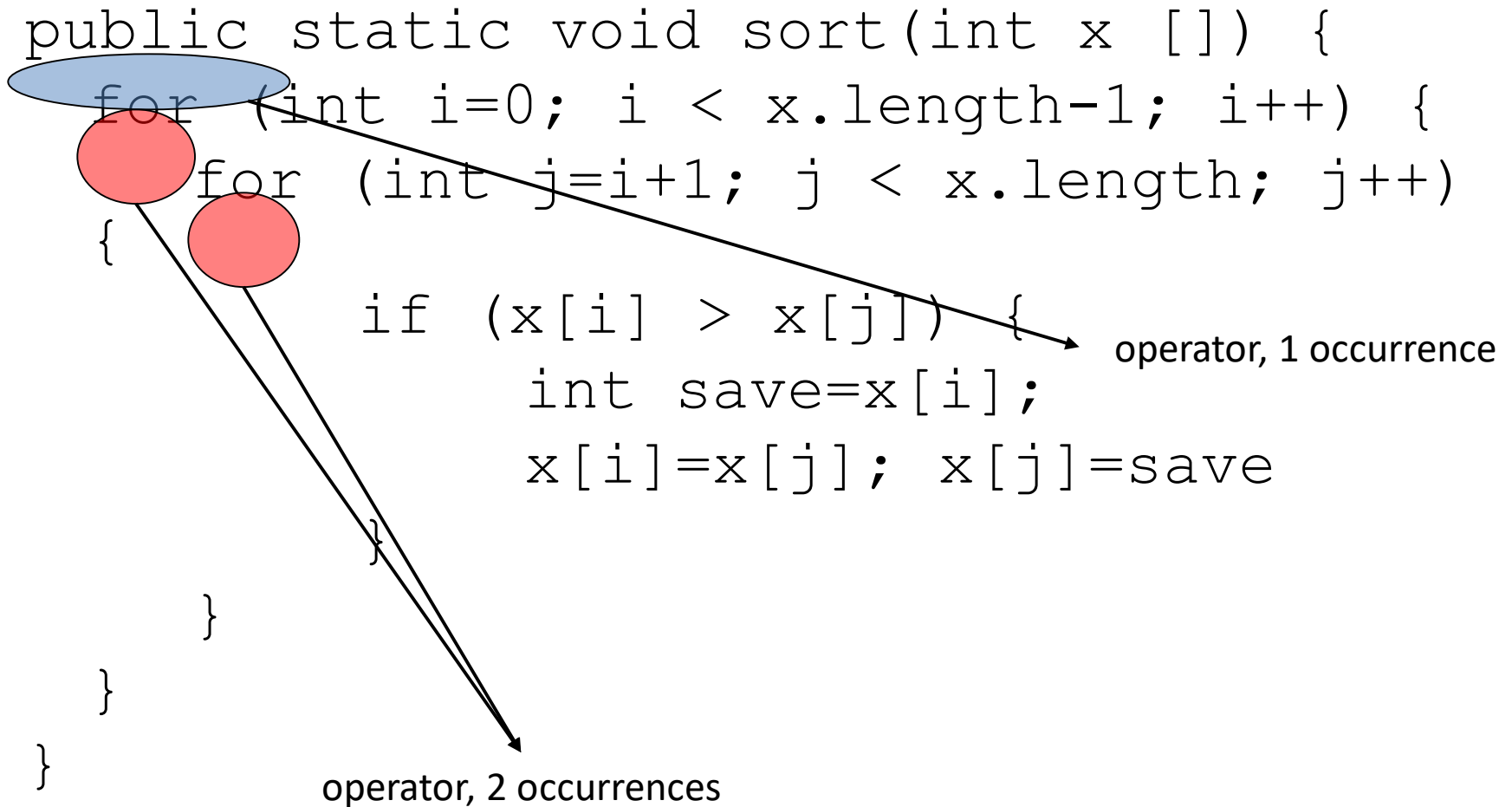
The effort measure translates into actual coding time using the following relation,

## Example program

```
public static void sort(int x []) {
    for (int i=0; i < x.length-1; i++) {
        for (int j=i+1; j < x.length; j++)
        {
            if (x[i] > x[j]) {
                int save=x[i];
                x[i]=x[j]; x[j]=save
            }
        }
    }
}
```

operator, 1 occurrence

operator, 2 occurrences



## Example program

operator	# of occurrences
public	1
sort()	1
int	4
[]	7
{}	4
for {;;}	2
if ()	1
=	5
<	2
...	...
$n_1 = 17$	$N_1 = 39$

## Example

$i = i+1$	Total	Unique
Operators	$N1 = 2$	$n1 = 2$
Operands	$N2 = 3$	$n2 = 2$

$i++$	Total	Unique
Operators	$N1 = 1$	$n1 = 1$
Operands	$N2 = 1$	$n2 = 1$

```
void sort ( int *a, int n ) {
    int i, j, t;
```

```
    if ( n < 2 ) return;
    for ( i=0 ; i < n-1; i++ ) {
        for ( j=i+1 ; j < n ; j++ ) {
            if ( a[i] > a[j] ) {
                t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        }
    }
}
```

$V = 80 \log_2(24) \approx 392$

- Ignore the function definition
- Count operators and operands

```
3 < 3 {
5 = 3 }
1 > 1 +
1 - 2 ++
2 , 2 for
9 ; 2 if
4 ( 1 int
4 ) 1 return
6 []
```

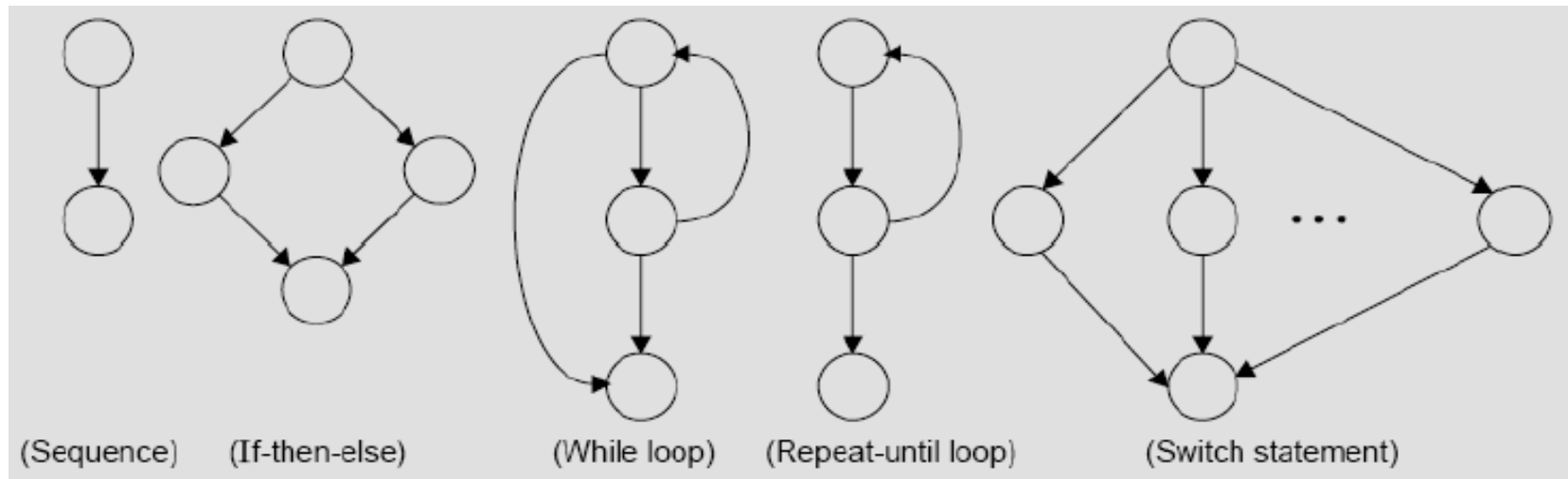
```
1 0
2 1
1 2
6 a
8 i
7 j
3 n
3 t
```

	Total	Unique
Operators	$N1 = 50$	$n1 = 17$
Operands	$N2 = 30$	$n2 = 7$

# Cyclomatic Complexity Measure (CO3)

## Control Flow Graph

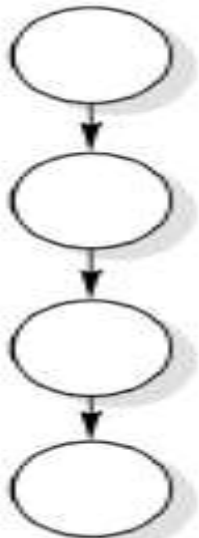
- The control flow of a program can be analyzed using a graphical representation known as control flow graph.
- The flow graph is a directed graph in which nodes are either entire statements or fragments of a statement, and edges represents flow of control



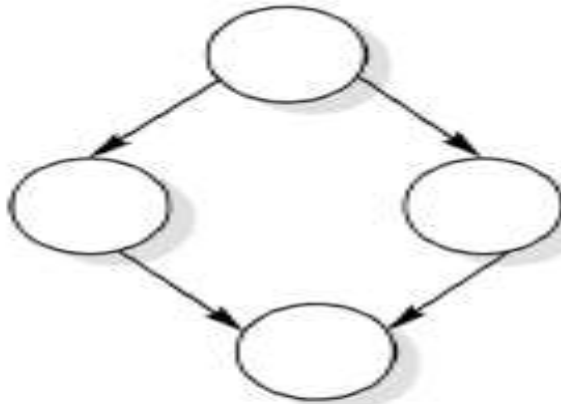
## The basic construct of the flow graph

## Connected component

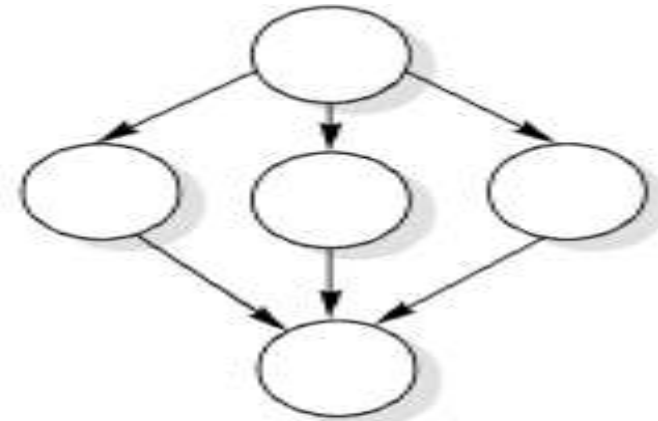
- Control flow graph with unique entry and exit nodes, all nodes reachable from the entry, and exit reachable from all nodes then it has **only one** connected component.
- imagine a main program M and two called subroutines A and B having a control flow graph



M:



A:



B:

# Faculty Video Links, Youtube & NPTEL Video Links and Online Courses Details

- <https://nptel.ac.in/courses/106/105/106105182/>
- [https://www.youtube.com/watch?v=5q\\_KBeNIRFk&list=PLbRMhDVUMngf8oZR3DpKMvYhZKga90JVt&index=19](https://www.youtube.com/watch?v=5q_KBeNIRFk&list=PLbRMhDVUMngf8oZR3DpKMvYhZKga90JVt&index=19)
- <https://www.youtube.com/watch?v=FTyncRpLd5g&list=PLbRMhDVUMngf8oZR3DpKMvYhZKga90JVt&index=20>
- <https://www.youtube.com/watch?v=OFxBjpE8mT0&list=PLbRMhDVUMngf8oZR3DpKMvYhZKga90JVt&index=22>

1) Which one is not a size measure for software

(a) LOC

(b) Function Count

**(c) Cyclomatic Complexity**

(d) Halstead's program length

2) The worst type of coupling is

**(a) Content**

(b) Common

(c) External

(d) Data coupling

3) The most desirable form of cohesion is

(a) Logical cohesion

(b) Procedural cohesion

**(c) Functional cohesion**

(d) Temporal cohesion

4) Which one is not a strategy for design?

(a) Bottom up design

(b) Top down design

(c) Embedded design

**(d) Hybrid design**

5) The term module used during design phase refers to

(a) Function

(b) Procedure

(c) Sub program

**(d) All of the above**

6) Fault is

- (a) Defect in the program
- (b) Mistake in the program
- (c) Error in the program
- (d) All of the above**

7) The extent to which different modules are dependent upon each other is called

- (a) Coupling**
- (b) Cohesion
- (c) Modularity
- (d) Stability

8) A system that does not interact with external environment is called

- (a) Closed system**
- (b) Logical system
- (c) Open system
- (d) Hierarchal system

9) The worst type of cohesion is

- (a) Temporal cohesion
- (b) Coincidental cohesion**
- (c) Logical cohesion
- (d) Sequential cohesion

10) Which one is not a strategy for design?

- (a) Bottom up design
- (c) Embedded design**
- (b) Top down design
- (d) Hybrid design

# Weekly Assignment

1. What is design? Describe the difference between conceptual design and technical design.
2. Discuss the objectives of software design. How do we transform an informal design to a detailed design?
3. Do we design software when we “write” a program? What makes software design different from coding?
4. What is modularity? List the important properties of a modular system.
5. Define module coupling and explain different types of coupling.
6. Define module cohesion and explain different types of cohesion.
7. Discuss the objectives of modular software design. What are the effects of module coupling and cohesion?
8. If a module has logical cohesion, what kind of coupling is this module likely to have with others?
9. What problems are likely to arise if two modules have high coupling

1. The most desirable form of coupling is
  - (a) Control Coupling
  - (b) Data Coupling
  - (c) Common Coupling
  - (d) Content Coupling
2. The worst type of coupling is
  - (a) Content coupling
  - (b) Common coupling
  - (c) External coupling
  - (d) Data coupling
3. The most desirable form of cohesion is
  - (a) Logical cohesion
  - (b) Procedural cohesion
  - (c) Functional cohesion
  - (d) Temporal cohesion
4. The worst type of cohesion is
  - (a) Temporal cohesion
  - (b) Coincidental cohesion
  - (c) Logical cohesion
  - (d) Sequential cohesion
5. Which one is not a strategy for design?
  - (a) Bottom up design
  - (b) Top down design
  - (c) Embedded design
  - (d) Hybrid design

6. Temporal cohesion means
  - (a) Cohesion between temporary variables
  - (b) Cohesion between local variable
  - (c) Cohesion with respect to time
  - (d) Coincidental cohesion
7. Functional cohesion means
  - (a) Operations are part of single functional task and are placed in same procedures
  - (b) Operations are part of single functional task and are placed in multiple procedures
  - (c) Operations are part of multiple tasks
  - (d) None of the above
8. When two modules refer to the same global data area, they are related as
  - (a) External coupled
  - (b) Data coupled
  - (c) Content coupled
  - (d) Common coupled
9. The module in which instructions are related through flow of control is
  - (a) Temporal cohesion
  - (b) Logical cohesion
  - (c) Procedural cohesion
  - (d) Functional cohesion

# Old Question Papers

Printed Pages : 1

Roll No. 

--	--	--	--	--	--	--	--	--	--

ECS602

**B. TECH.**

**THEORY EXAMINATION (SEM–VI) 2016-17**  
**SOFTWARE ENGINEERING**

*Time : 3 Hours*

*Max. Marks : 100*

*Note : Be precise in your answer.*

**SECTION – A**

1. Attempt all parts of the following questions:

**10 x 2 = 20**

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

**SECTION – B**

2. Attempt any five parts of the following questions:

**5 x 10 = 50**

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

**SECTION – C**

Attempt any two parts of the following questions:

**2 x 15 = 30**

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

# Old Question Papers

Printed Pages: 02

Paper Id: 

1	1	0	6	0	2
---	---	---	---	---	---

Sub Code: NCS602/ECS602

Roll No. 

--	--	--	--	--	--	--	--	--	--

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

*Time: 3 Hours*

*Total Marks: 100*

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

**SECTION A**

**1. Attempt all questions in brief. 2 x 10 = 20**

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

**SECTION B**

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

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

**SECTION C**

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

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

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

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

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

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

# Old Question Papers

6. Attempt any *one/two* part of the following: **10 x 1 = 10**
- (a) What is Regression Testing? Explain the process of test case prioritization in regression testing.
  - (b) What is Integration Testing? Explain different approaches used for integration testing.
7. Attempt any *one/two* part of the following: **10 x 1 = 10**
- (a) Explain various software configuration management activities.
  - (b) Explain Software Risks Analysis and Management process.

## Expected Questions for University Exam

1. Explain modularity? Explain Under modularity and over modularity in a software should be avoided.
2. Describe data design at architectural level.
3. What are different techniques to estimate size of the program? Which technique is better and why?
4. Discuss the main advantages of using an object-oriented approach for software design.
5. Discuss the differences between object oriented and function oriented design

## Software Design

- Basic Concept of Software Design
- Architectural Design Low Level Design: Modularization
- Design Structure Charts
- Pseudo Codes
- Flow Charts
- Coupling and Cohesion Measures

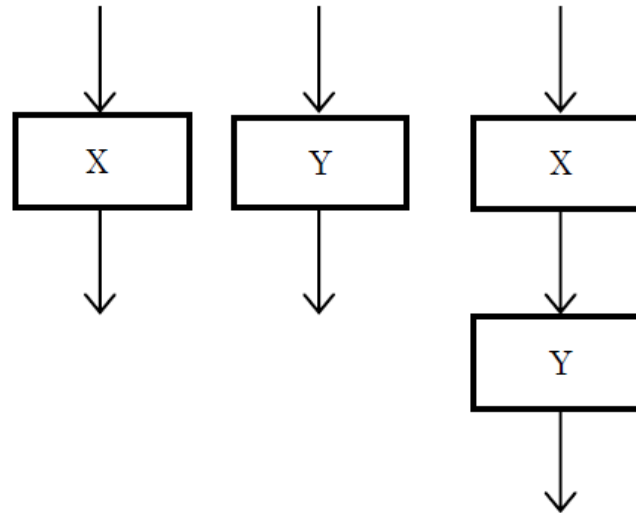
## Design Strategies:

- Function Oriented Design
- Object Oriented Design
- Top-Down and Bottom-Up Design
- Software Measurement and Metrics
- Various Size Oriented Measures
- Halstead's Software Science
- Function Point (FP) Based Measures
- Cyclomatic Complexity Measures, Control Flow Graphs

1. R. S. Pressman, Software Engineering: A Practitioners Approach, McGraw Hill.
2. Rajib Mall, Fundamentals of Software Engineering, PHI Publication.
3. K. K. Aggarwal and Yogesh Singh, Software Engineering, New Age International Publishers.
4. Pankaj Jalote, Software Engineering, Wiley
5. Deepak Jain, " Software Engineering: Principles and Practices", Oxford University Press.
6. Munesh C. Trivedi, Software Engineering, Khanna Publishing House
7. N.S. Gill, Software Engineering, Khanna Publishing House

## Example

a. The cyclomatic complexity of each of the modules X and Y shown below is 10. What is the cyclomatic complexity of the sequential integration shown on the right hand side?



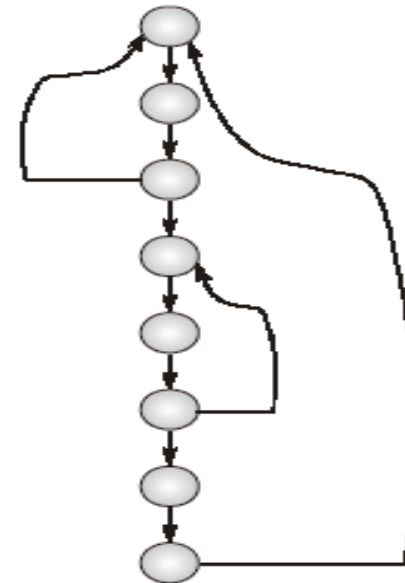
# Example

## Program-X:

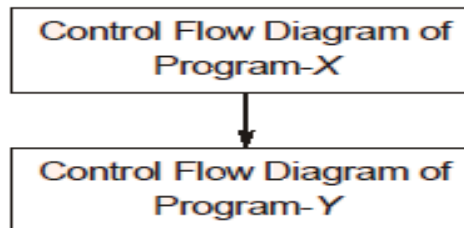
```

sumcal (int maxint, int value)
{
    int result=0, i=0;
    if (value < 0)
    {
        value = -value;
    }
    while ( (i < value) AND (result
    <= maxint) )
    {
        i = i + 1;
        result = result + 1;
    }
    if (result <= maxint)
    {
        print f (result) ;
    }
    else
    {
        print f("large");
        print f ("end of program");
    }
}
    
```

## Control Flow Diagram of Program-Y:



## Control Flow Diagram of Program-Z:



## Example

25. Consider a software project with the following information domain characteristic for calculation of function point metric.

Number of external inputs (I) = 30

Number of external output (O) = 60

Number of external inquiries (E) = 23

Number of files (F) = 08

Number of external interfaces (N) = 02

It is given that the complexity weighting factors for I, O, E, F and N are 4, 5, 4, 10 and 7, respectively. It is also given that, out of fourteen value adjustment factors that influence the development effort, four factors are not applicable, each of the other four factors have value 3, and each of the remaining factors have value 4. The computed value of function point metric is