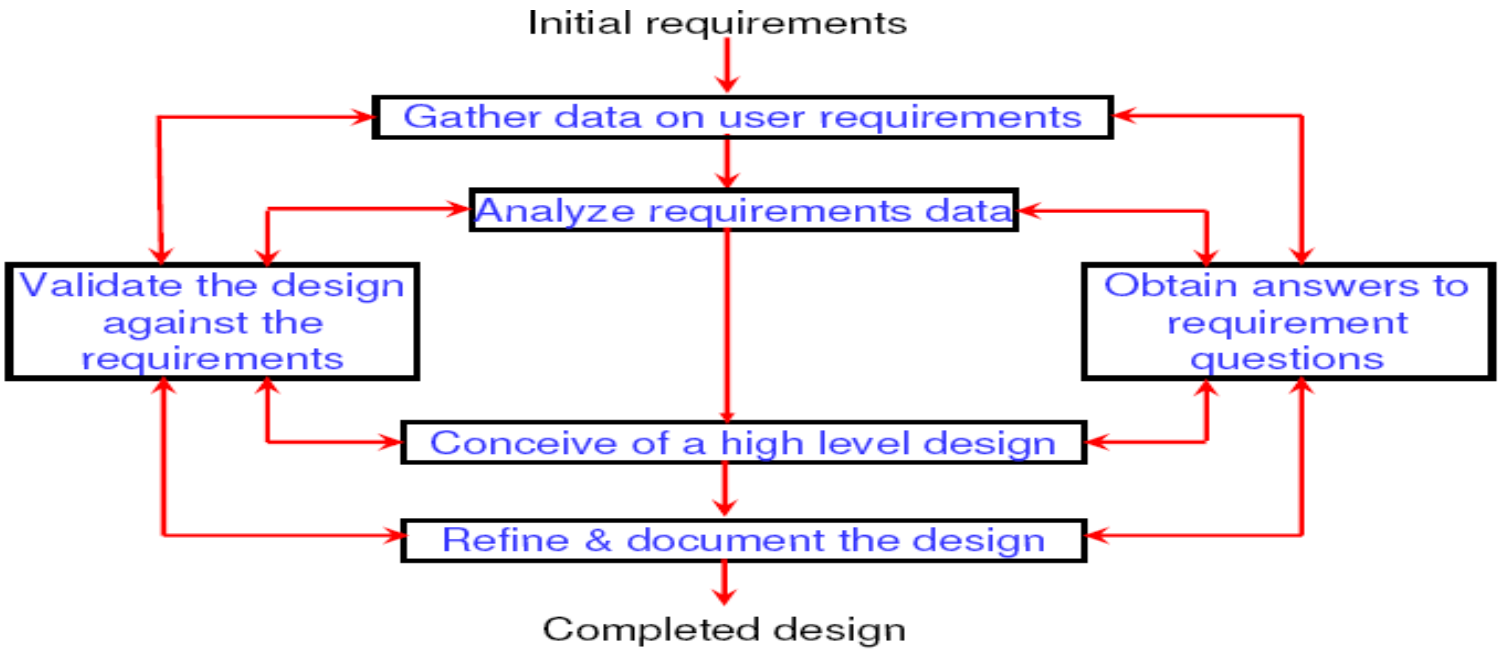
Software Design (CO3)

- 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:

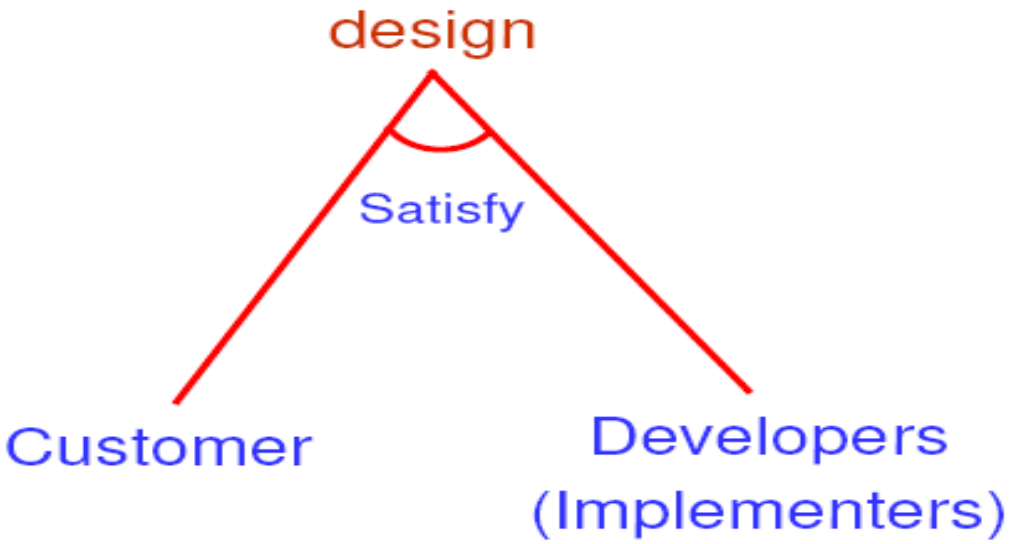


Software Design

- 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.

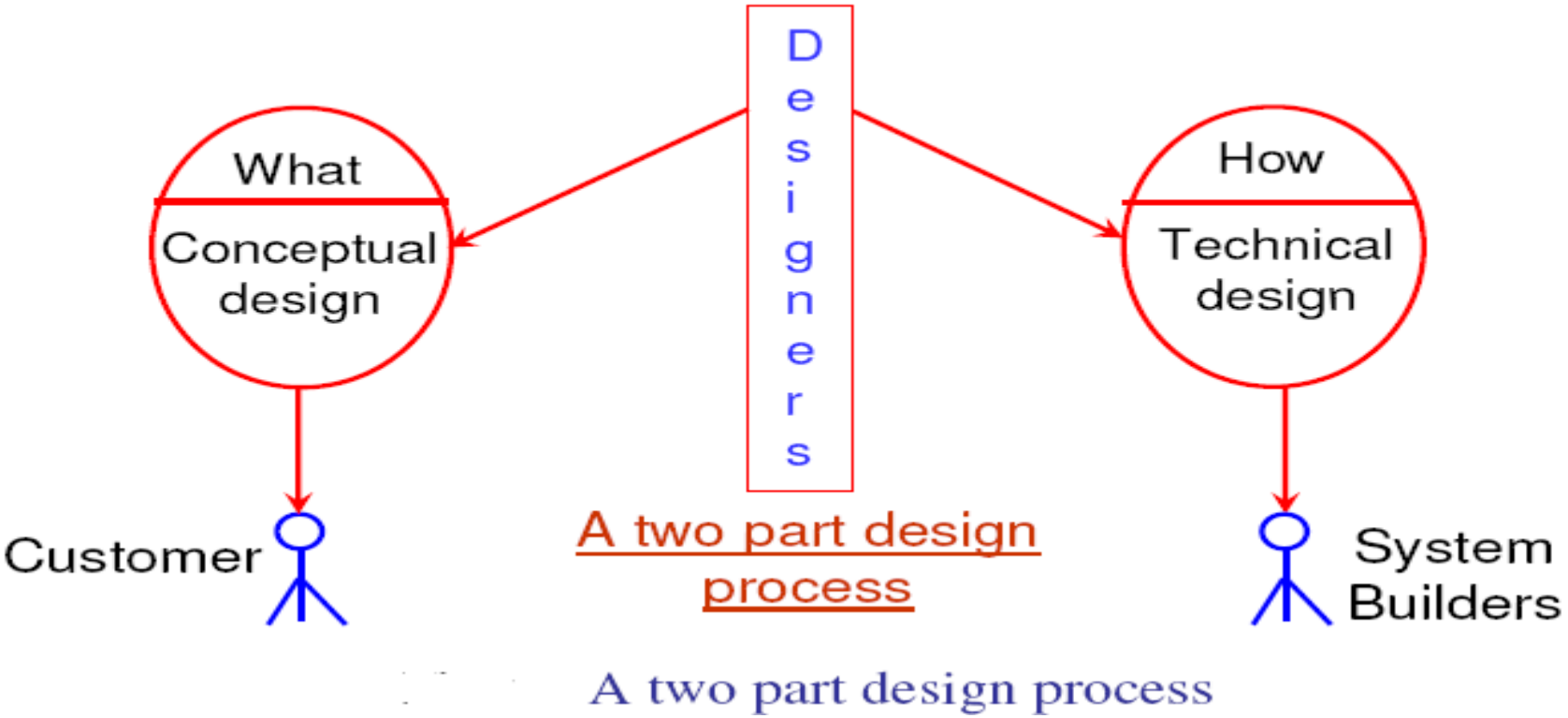
Software Design

- 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

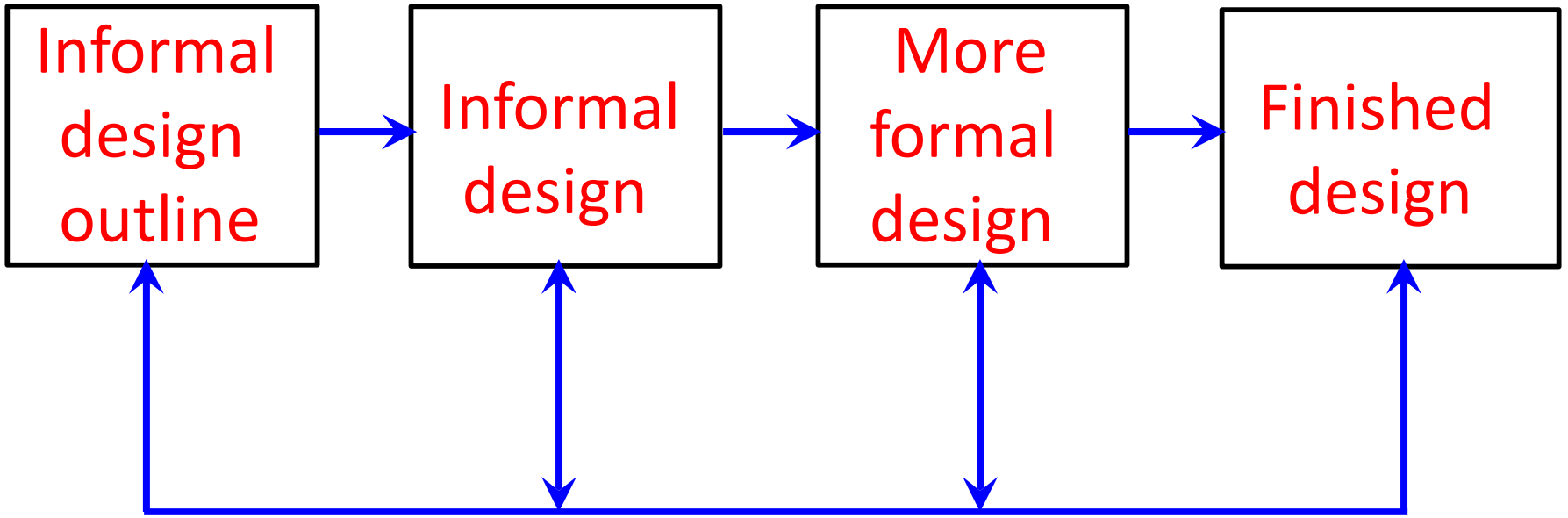


Fig: The transformation of an informal design to a detailed design.

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 (CO3)

- **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.

Modularity

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 discreet 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

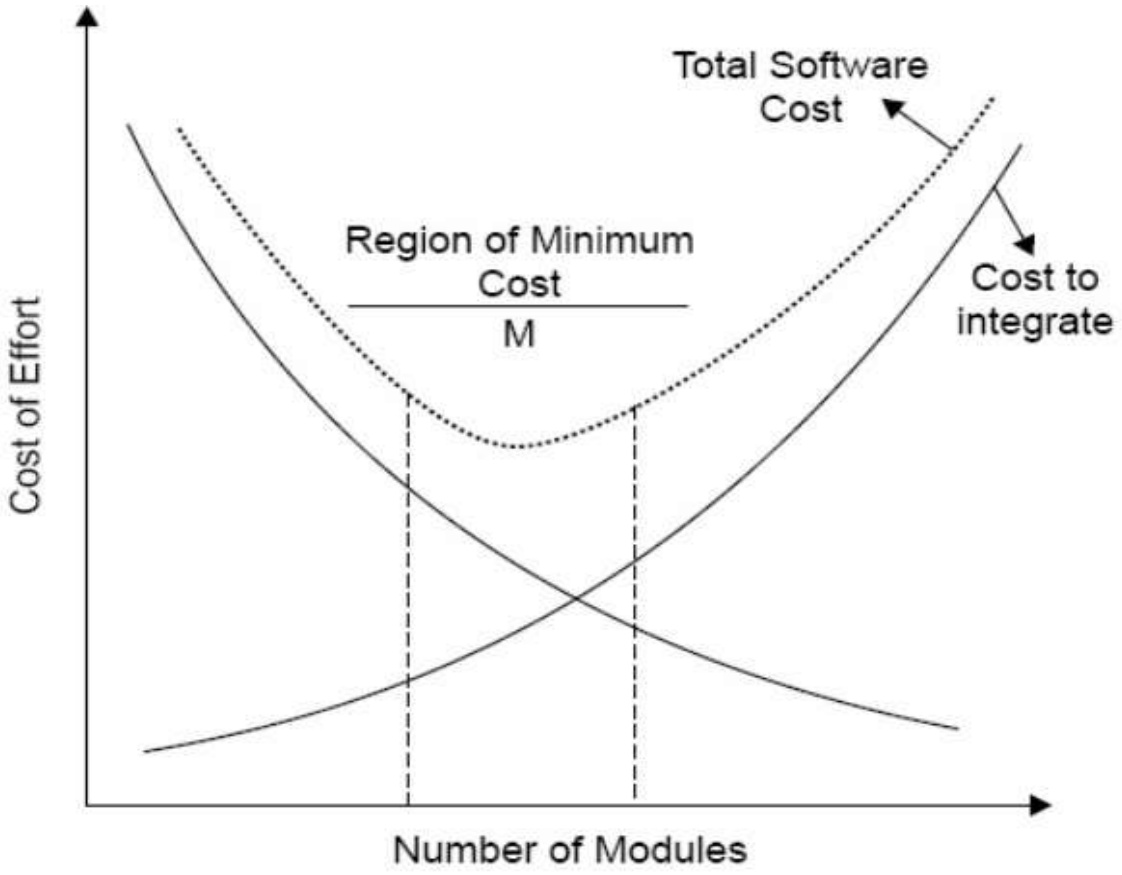
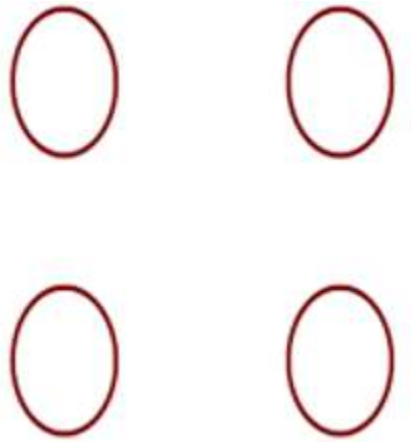


Fig. : Modularity and software cost

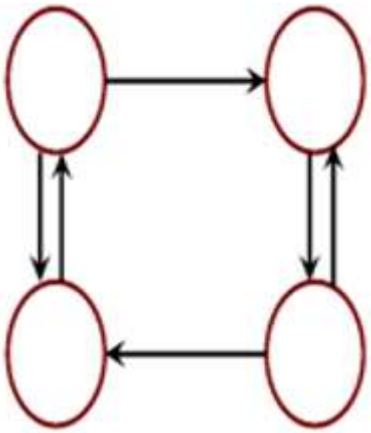
Module Coupling (CO3)

- 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

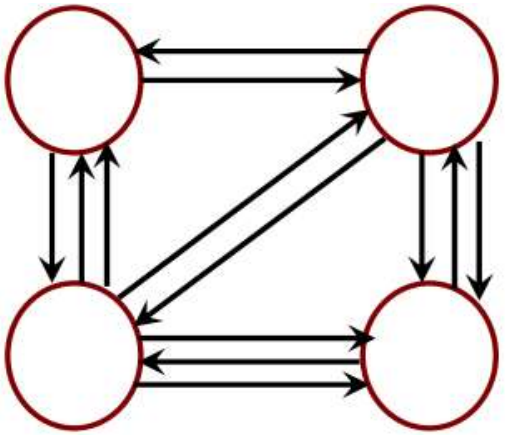
Module Coupling



Uncoupled : no dependencies



Loosely coupled: some dependencies



Highly coupled: many dependencies

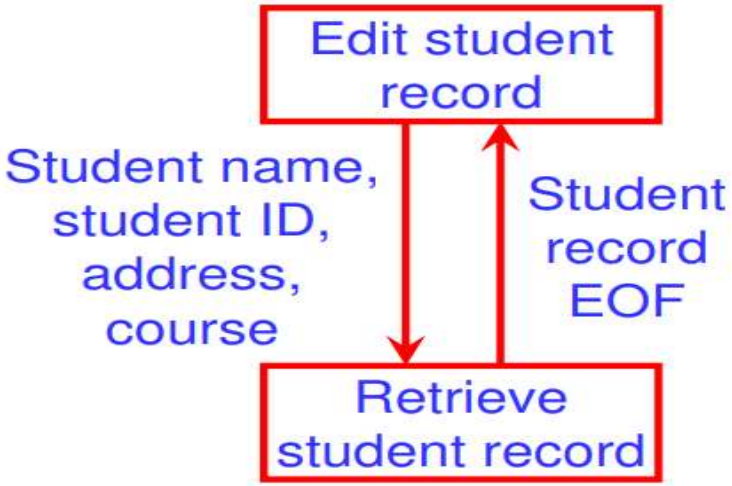
Module Coupling

Loose coupling can be achieved as:

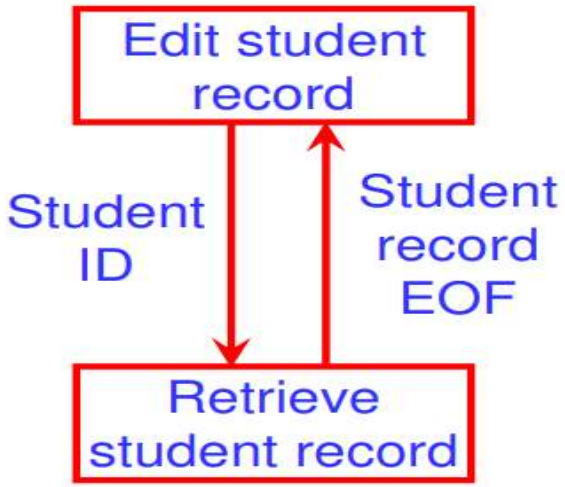
- Controlling the no of parameter passed amongst modules.
- Avoid passing undesired data to calling module.
- Maintain parent/child relationship between calling and called modules.
- Pass data, not the control information.

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	**Best**
Stamp coupling	
Control coupling	
External coupling	
Common coupling	
Content coupling	**Worst**

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

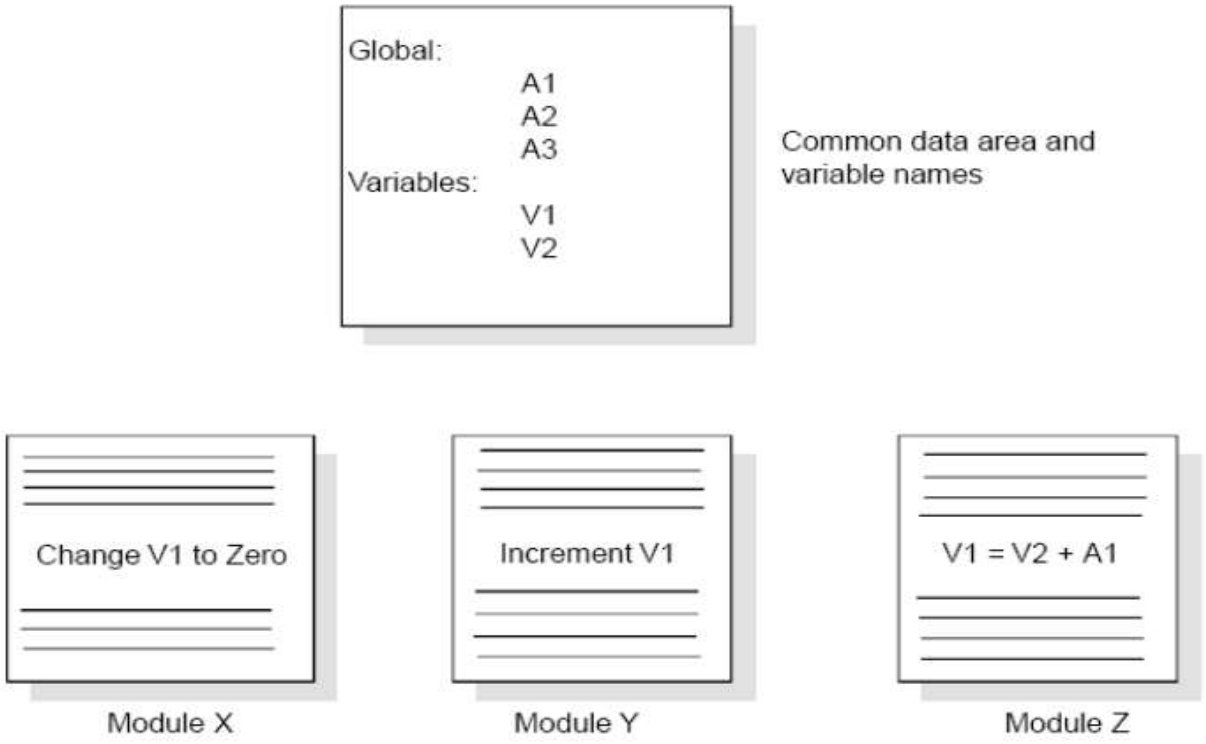
ripple effect

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

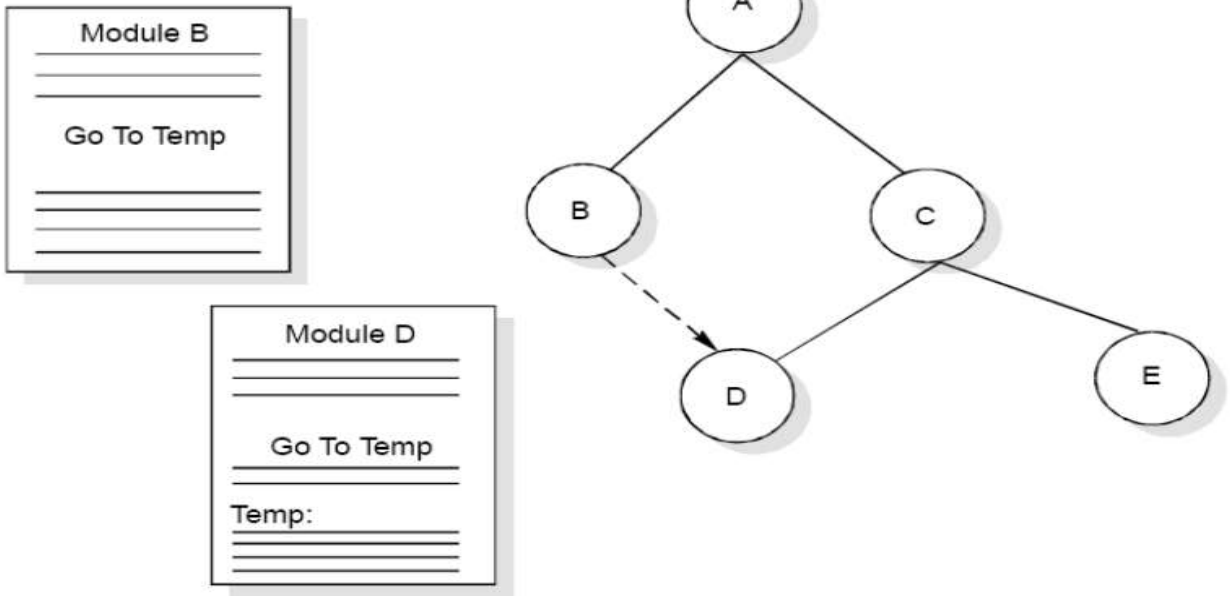
Example of common coupling



Types of Module Coupling

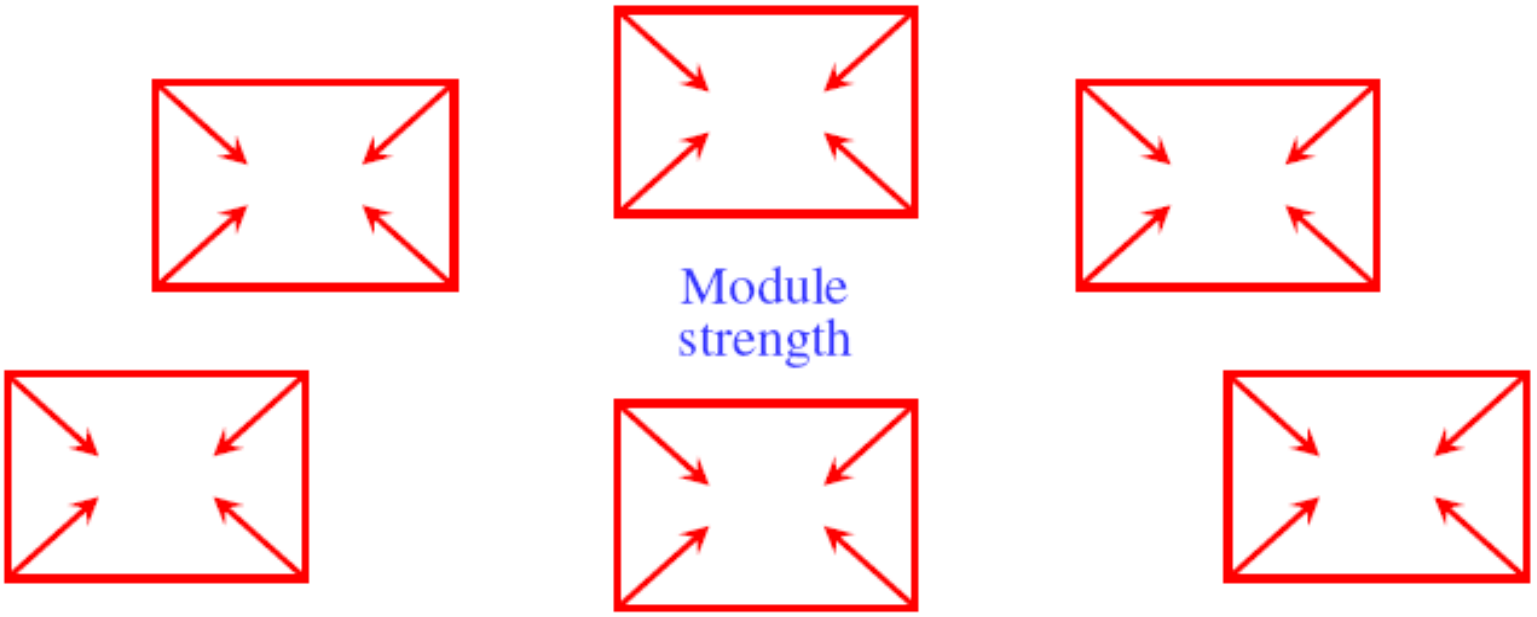
Content coupling

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.



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

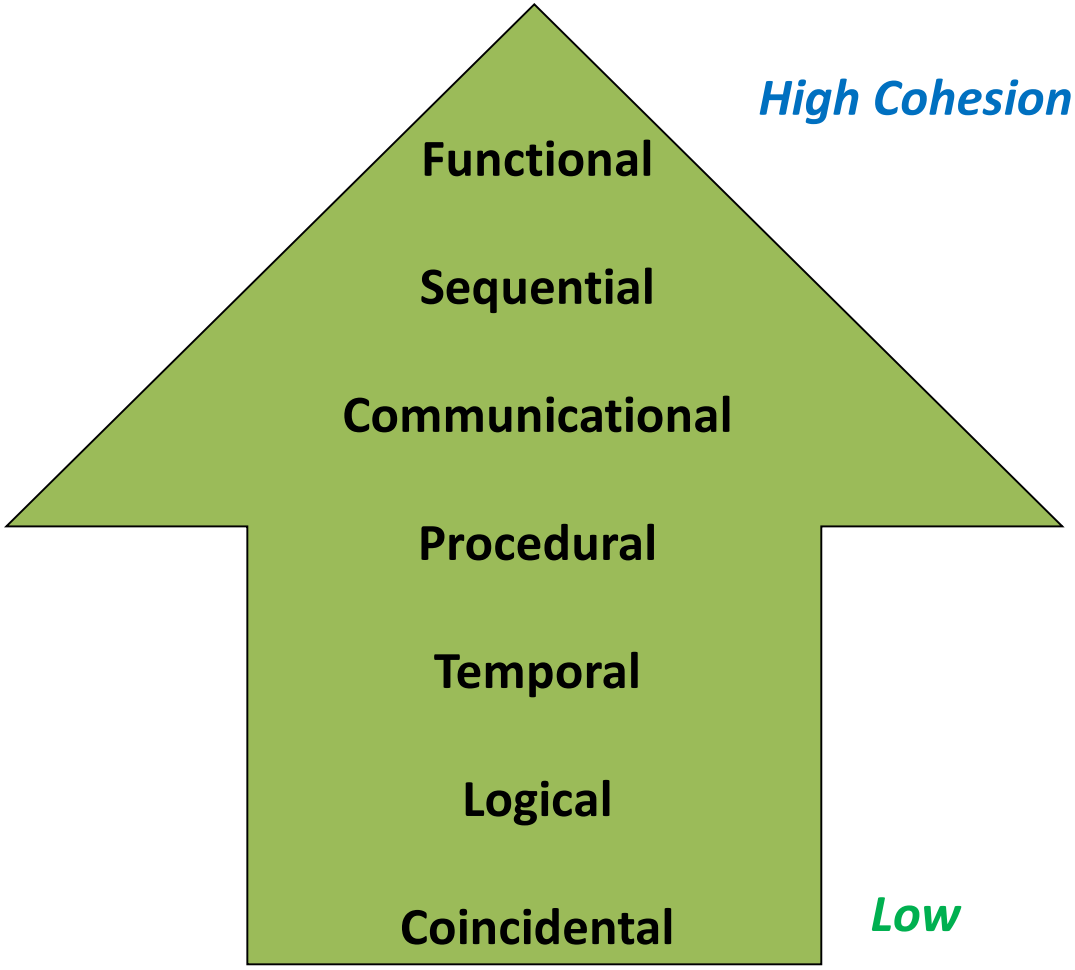
Type of Cohesion

- 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)

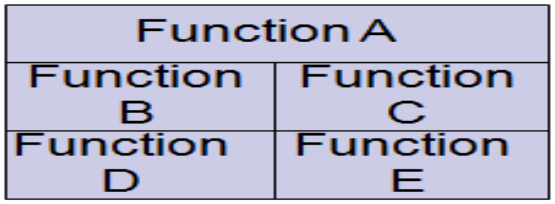
Type of Cohesion



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 5th argument
4. Convert 4th argument to float

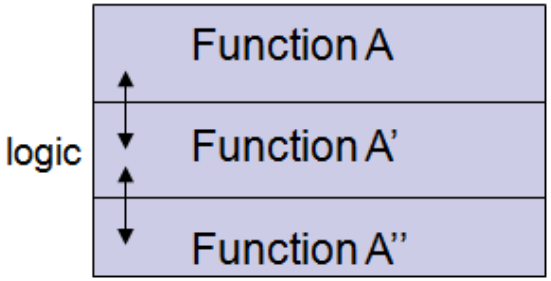


Coincidental
Parts unrelated

Logical Cohesion

- Elements of module are related logically and not functionally.

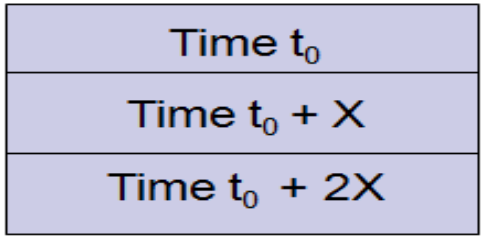
- Several logically related elements are in the same module and one of the elements is selected by the client module. Ex.
  - A module reads inputs from tape, disk, and network.
  - All the code for these functions are in the same module.
  - Operations are related, but the functions are significantly different.



Logical
Similar functions

Temporal Cohesion

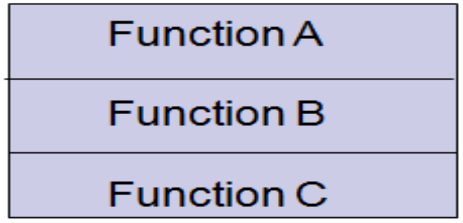
- Elements are related by timing involved
- Elements are grouped by when they are processed.
- Example: An exception handler that
  - Closes all open files
  - Creates an error log
  - Notifies user
  - Lots of different activities occur, all at same time



Temporal
Related by time

Procedural Cohesion

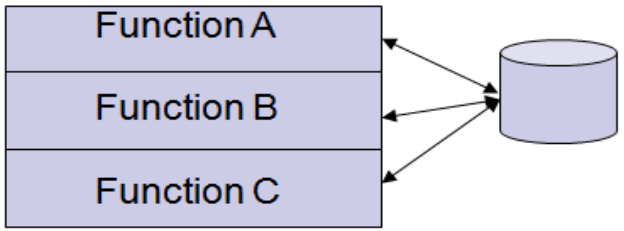
- Elements of a module are related only to ensure a particular order of execution.
- Actions are still weakly connected and unlikely to be reusable.
- Example:
  - ...
  - Write output record
  - Read new input record
  - Pad input with spaces
  - Return new record
  - ...



Procedural
Related by order of functions

Communicational Cohesion

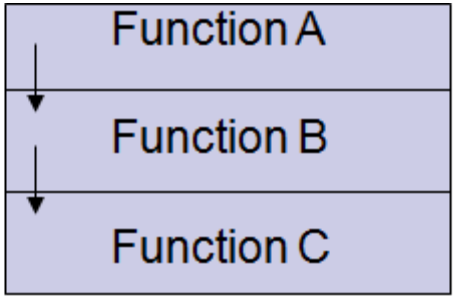
- Functions performed on the same data or to produce the same data.
- Examples:
  - Update record in data base and send it to the printer
    - Update a record on a database
    - Print the record



Communicational
Access same data

Sequential Cohesion

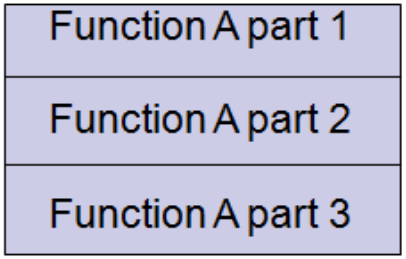
- Def: The output of one part is the input to another.
- *Data flows* between parts (different from procedural cohesion)
- Occurs naturally in functional programming languages



Sequential
Output of one is input to another

Functional Cohesion

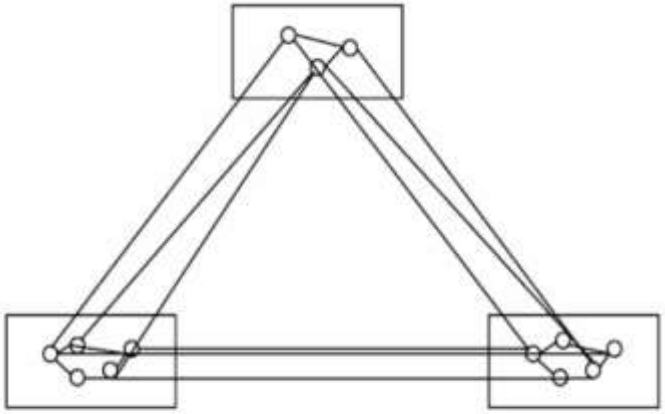
- Every essential element to a single computation is contained in the module.
- Every element in the module is essential to the computation.
- What is a functionally cohesive module?
  - One that not only performs the task for which it was designed but
  - it performs only that function and nothing else.



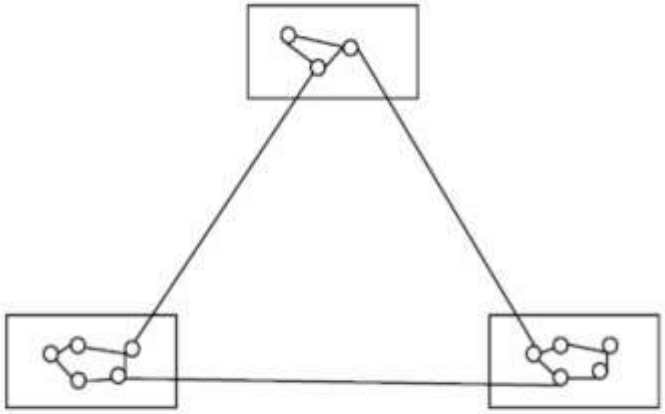
Functional
Sequential with complete, related functions

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

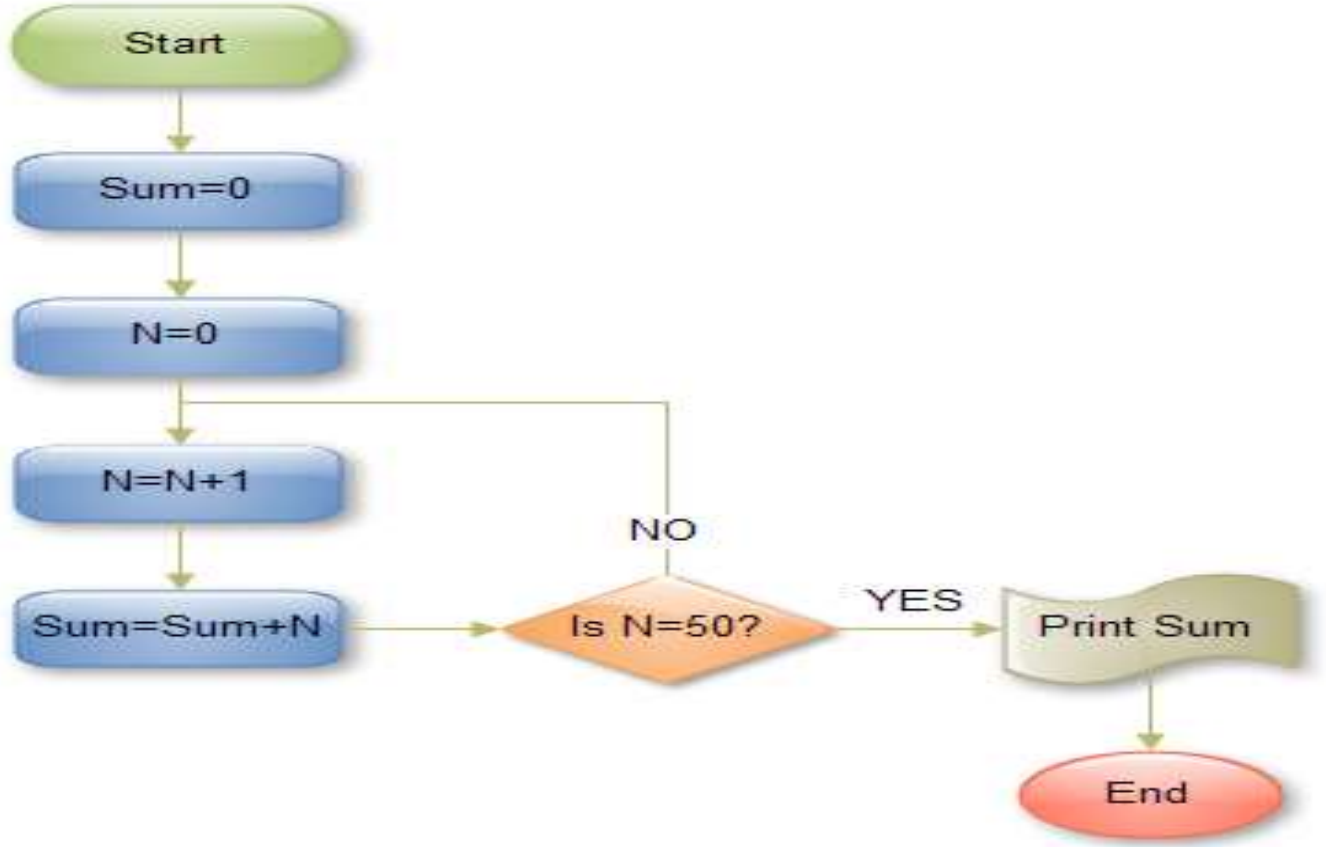


Fig. 1Flowchart for the sum of the first 50 natural numbers

Example 2

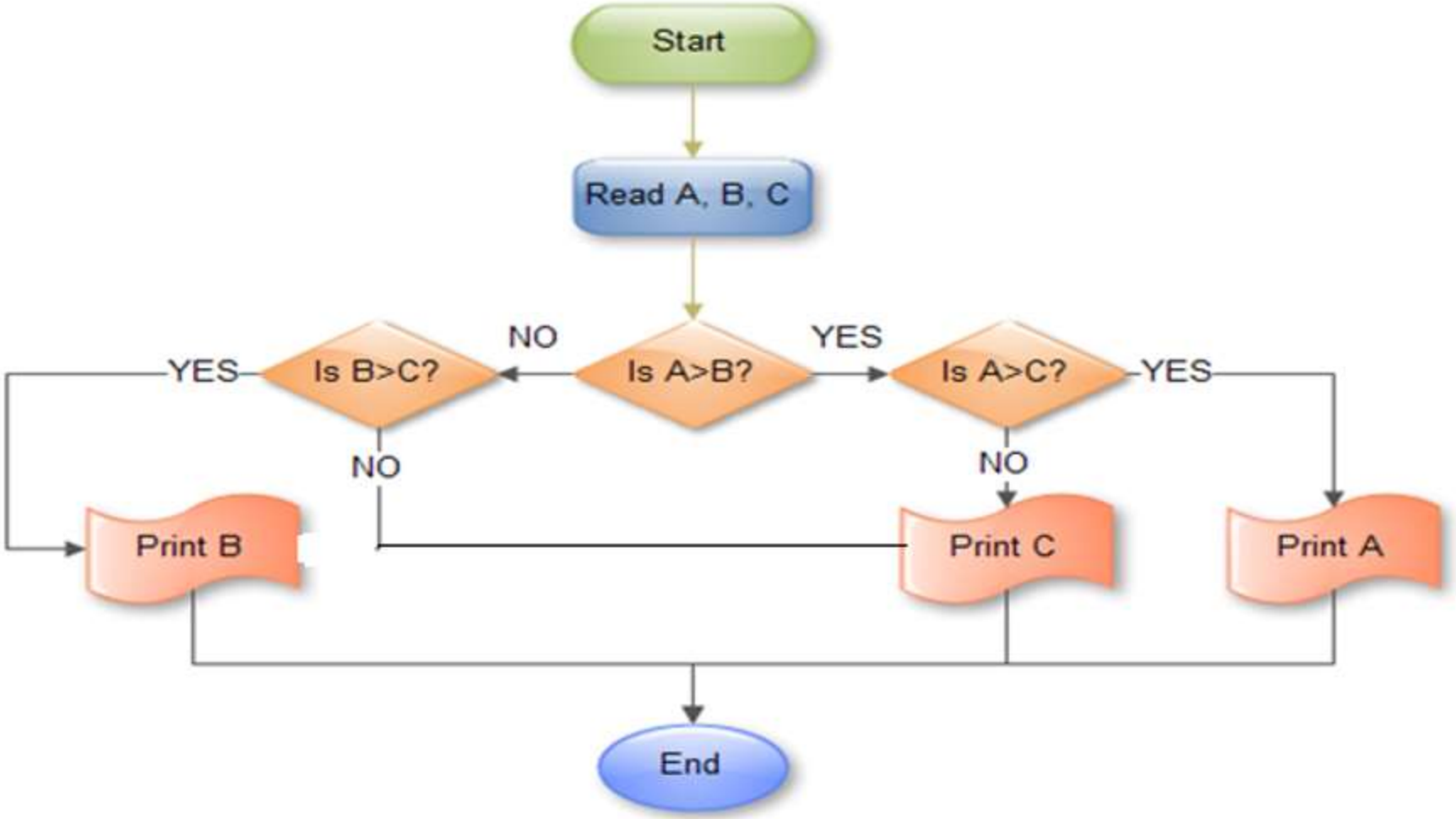


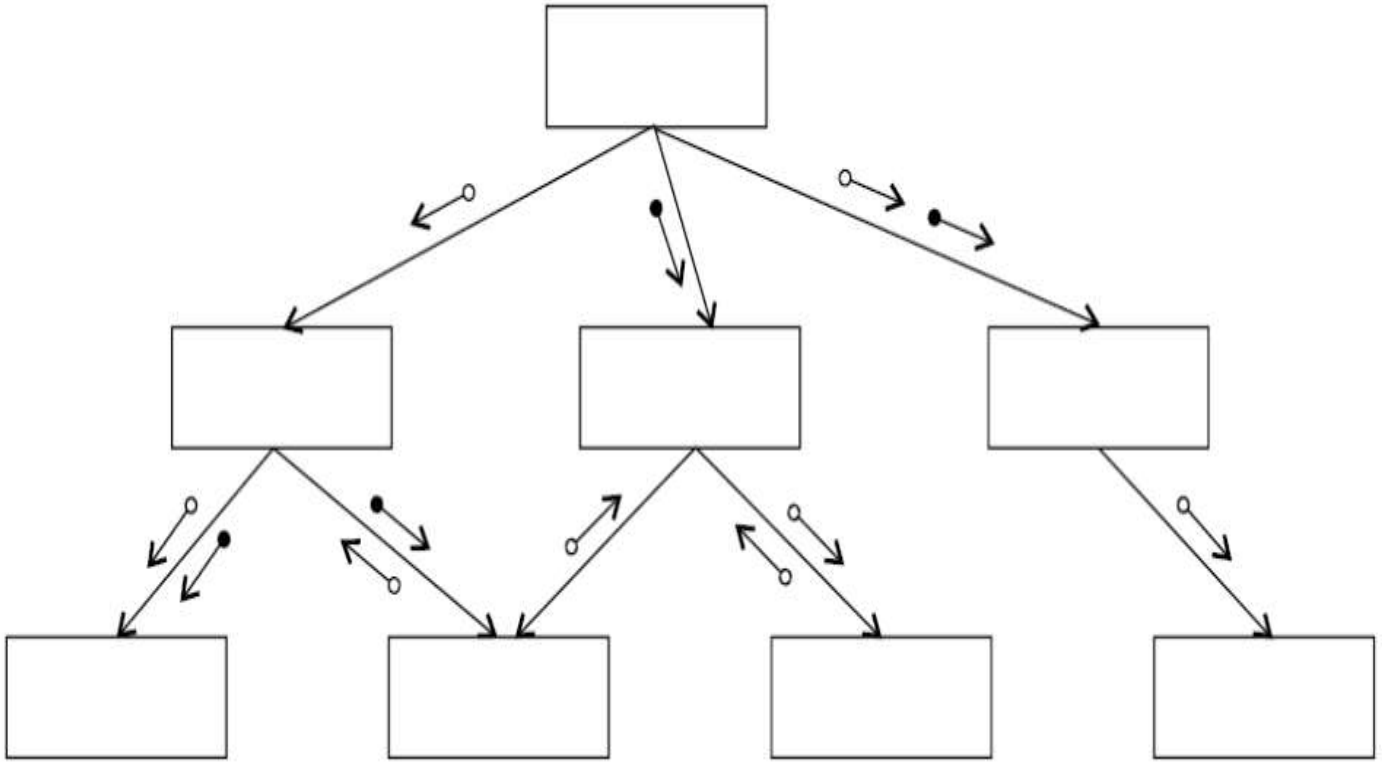
Fig 2 Flowchart for finding out the largest of three numbers

Structure Chart (CO3)

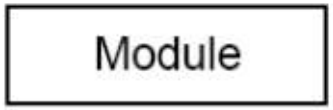
- It represent the s/w architecture that can be easily implemented using some prog. language.
- It partitions a system into black box(input/output)
- Connection between modules are represented by lines between rectangular boxes.
- Component are generally read from top to bottom and left to right.
- Top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

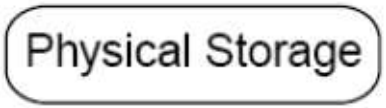
Structure Chart

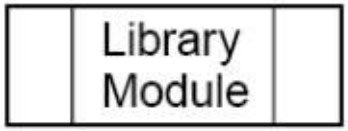
It partition a system into block boxes. A black box means that functionality is known to the user without the knowledge of internal design.

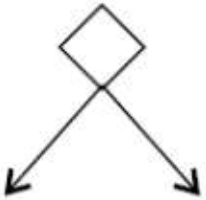


Notation of Structure Chart

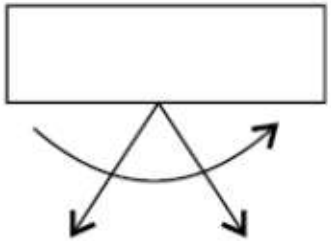








Diamond symbol for conditional call of module

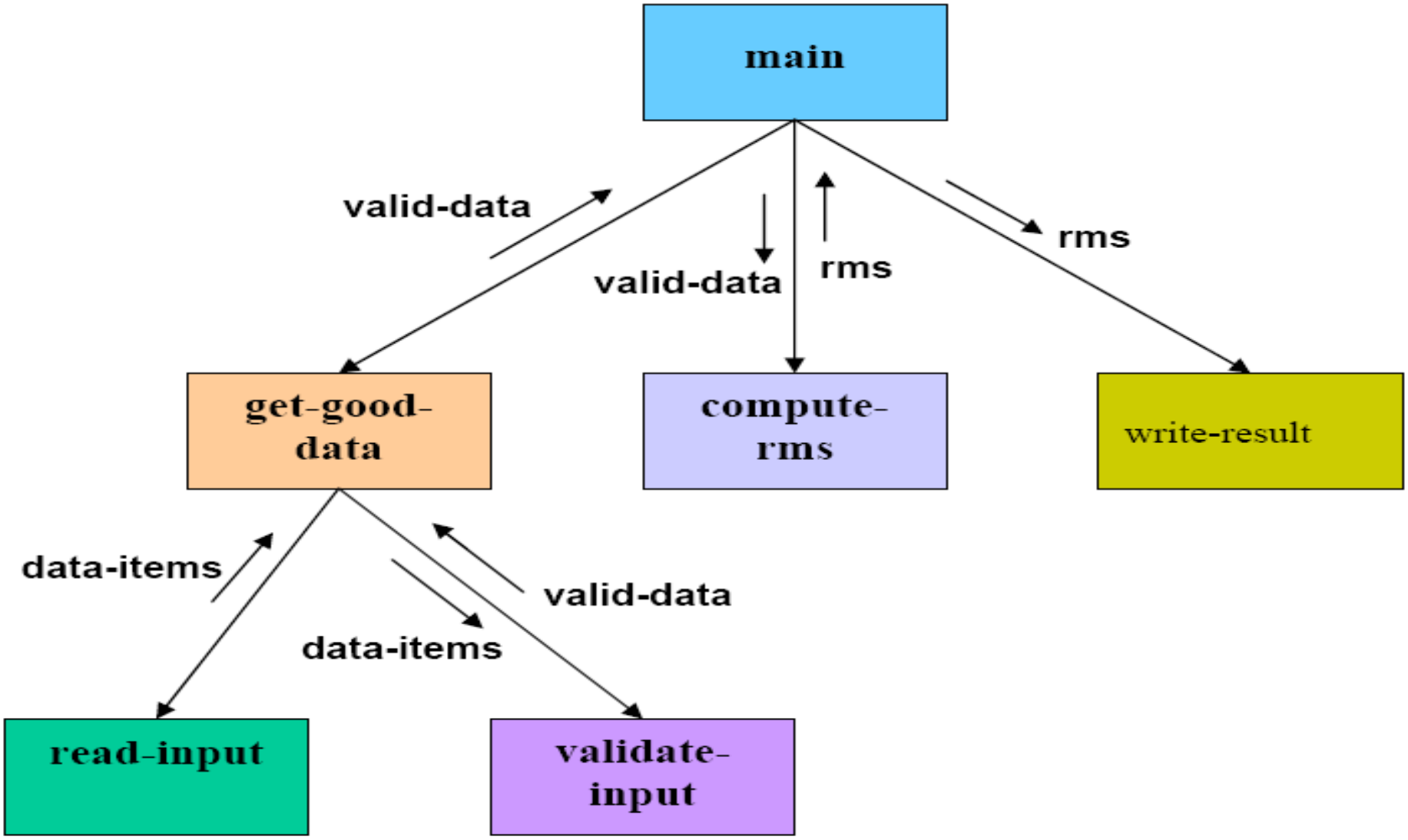


Repititive call of module

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

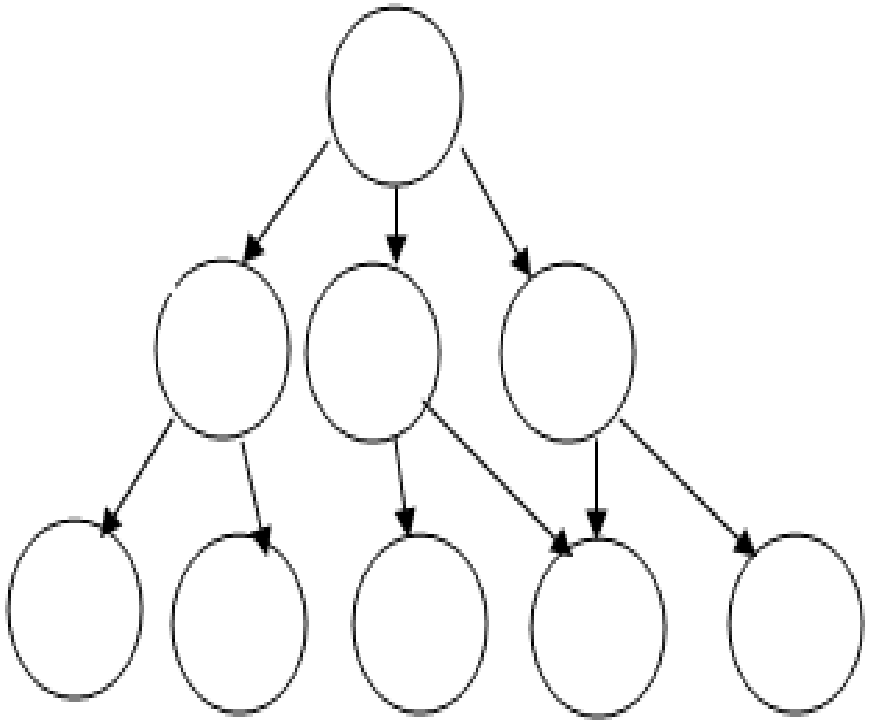
STRATEGY OF DESIGN

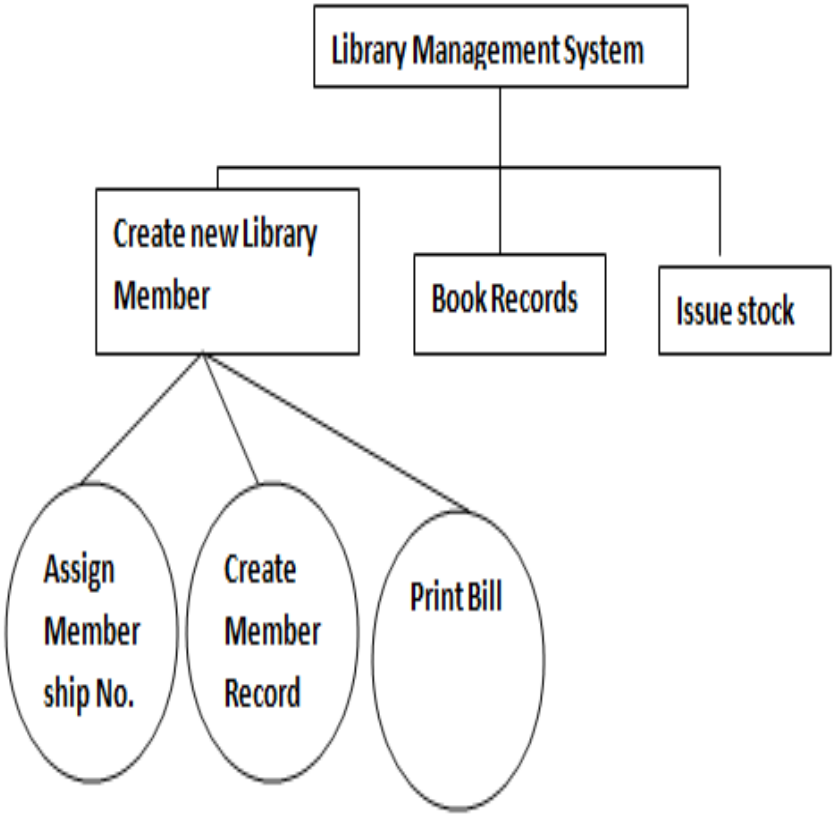
- Top down design
- Bottom up design.
- Function oriented design.
- Object oriented design.

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.

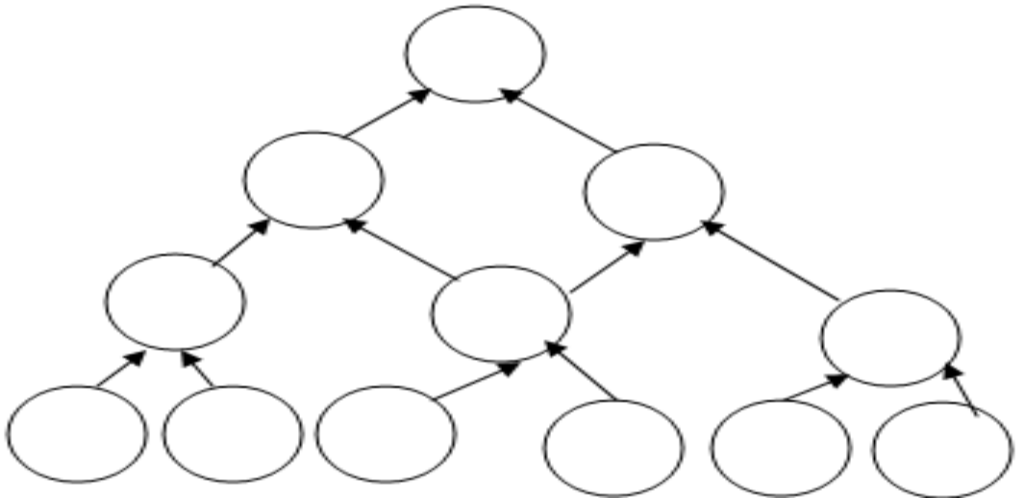
Top Down Design





Bottom up Design (CO3)

- 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.

Design Process

- 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.

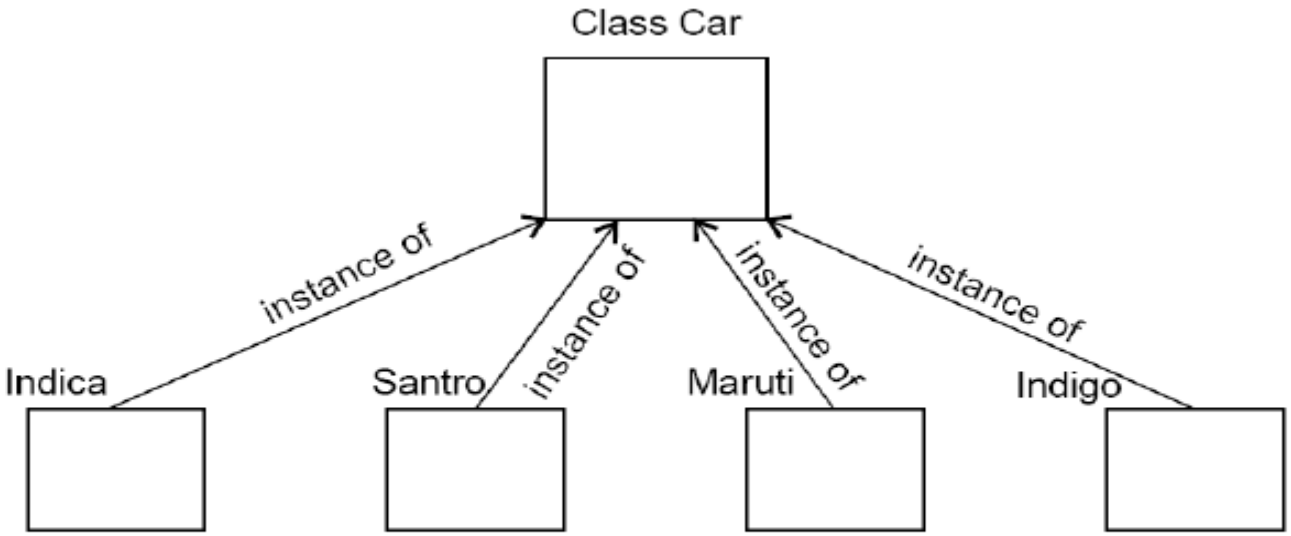
Object Oriented concepts (CO3)

- 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 do 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 methods to perform on the attributes.
- Classes - A class is a generalized description of an object. An object is an instance of a class. 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 Point[4]	Attributes
Set Colour() Draw()	Operations

The square class

Object Oriented concepts

Attribute

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

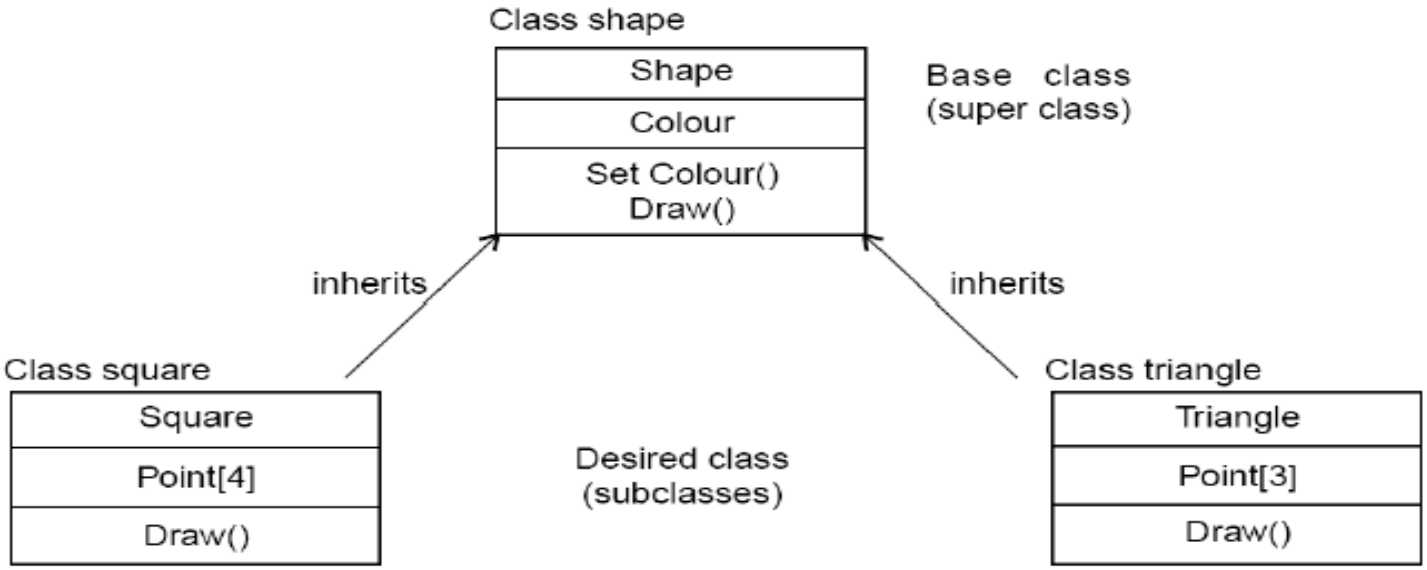
Operation

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.

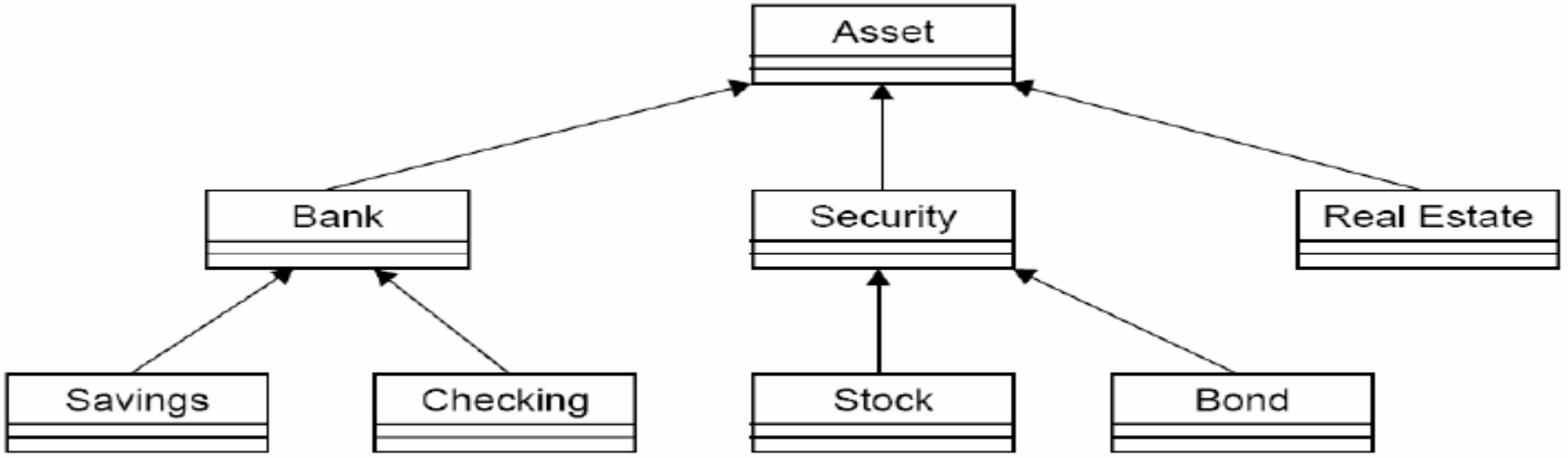
Object Oriented concepts



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)

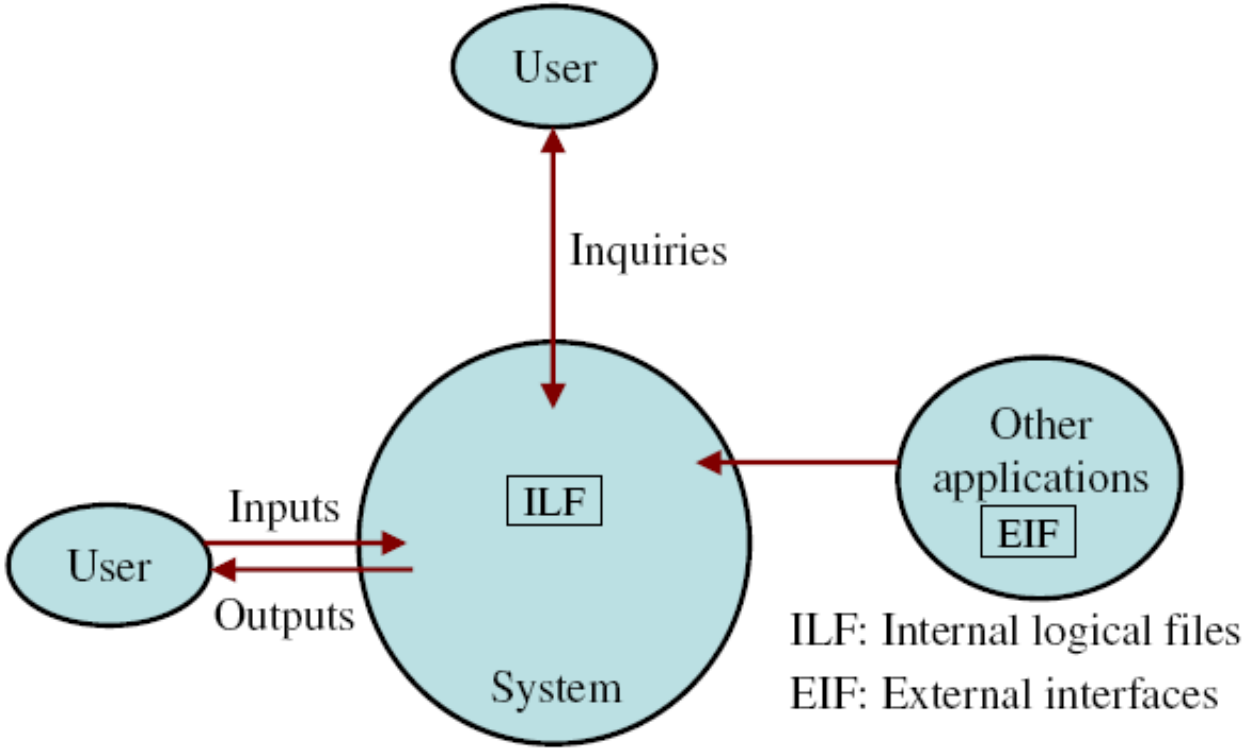
The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units.

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

FPA

The FPA functional units are shown in figure given below:



FPAs functional units System

FPA

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.

FPA

(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)		Low x 3	=		
		Average x 4	=		
		High x 6	=		
External Outputs (EOs)		Low x 4	=		
		Average x 5	=		
		High x 7	=		
External Inquiries (EQs)		Low x 3	=		
		Average x 4	=		
		High x 6	=		
External logical Files (ILFs)		Low x 7	=		
		Average x 10	=		
		High x 15	=		
External Interface Files (EIFs)		Low x 5	=		
		Average x 7	=		
		High x 10	=		
Total Unadjusted Function Point Count					

UFP

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

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

W_{ij} : It is the entry of the i^{th} row and j^{th} column of the table 1

Zij : 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 x ΣF_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

- η_1 = the number of distinct operators
- η_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 o 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

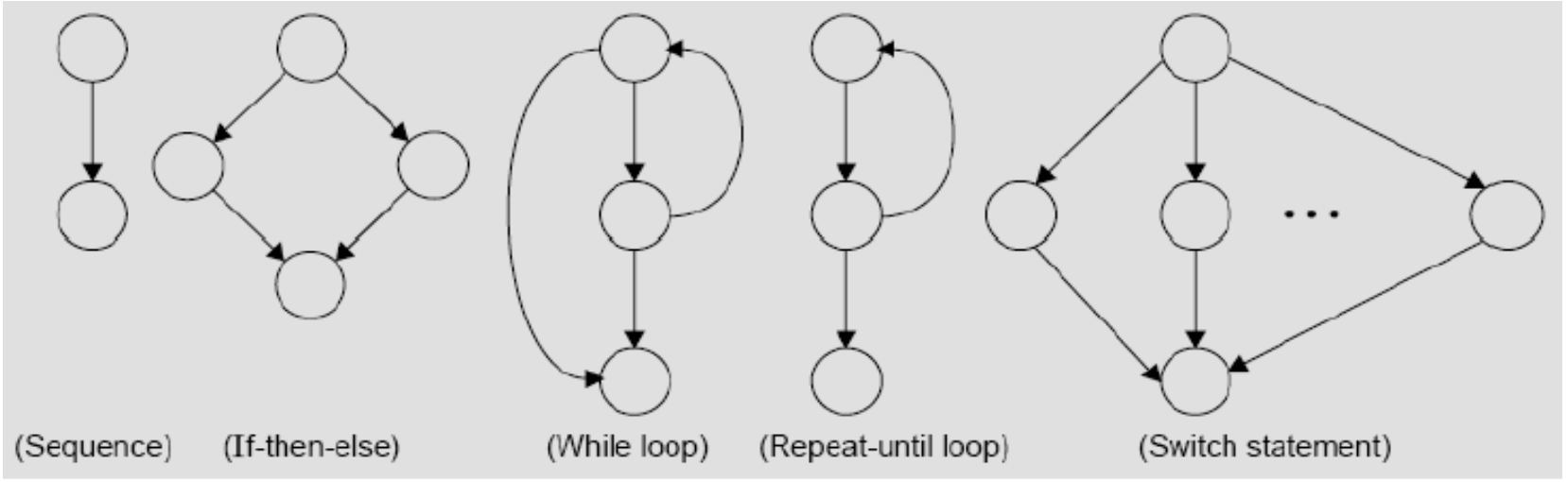
3	<	3	{
5	=	3	}
1	>	1	+
1	-	2	++
2	,	2	for
9	;	2	if
4	(	1	int
4	)	1	return
6	[]		

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