



## Software engineering unit 3

Software Engineering (Dr. A.P.J. Abdul Kalam Technical University)



Scan to open on Studocu

# **Software Engineering(Unit-3)**

# **What is Software Design?**

- Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. The output of the design phase is Software Design Document (SDD).

## **•Advantages of software design:-**

- 1) **Flexibility:-** We can easily change the software.
- 2) **Reusability:-** Modules are reusable.
- 3) Easy to understand the system.
- 4) Cost-efficiency is increased.

- The software design process can be divided into the following three levels of phases of design:

- 1) Architectural Design
- 2) High level Design/ Macro level Design
- 3) Detailed Design/ Low level Design

- 1) **Architectural Design** - The architectural design is the highest abstract version of the system.

- It identifies the software as a system with many components interacting with each other.
- At this level, the designers get the idea of proposed solution domain.
- The software needs the architectural design to represent the design of software.
- IEEE defines architectural design as “the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

2) **High-level Design-** The high-level design breaks the concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other.

- High-level design focuses on how the system along with all of its components can be implemented in forms of modules

3) **Detailed Design/ Low-level Design-** Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs.

- It is more detailed towards modules and their implementations.
- It defines logical structure of each module and their interfaces to communicate with other modules.

## **Objectives of Software Design**



## **Difference between functional oriented design and object oriented design**

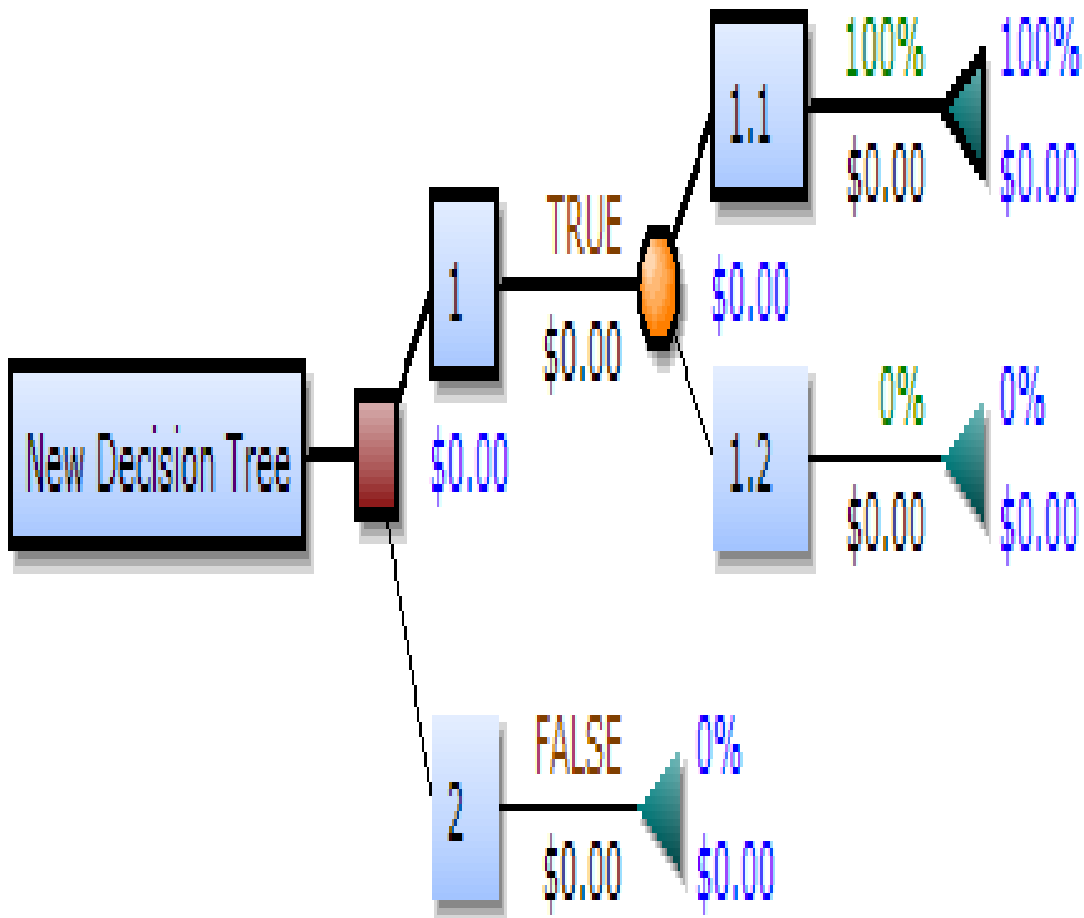
- **Function Oriented Design** is a software design method in which the model is broken into a series of interacting pieces or modules, each with a distinct function. As a result, the system is functionally designed.
- **Object oriented design** is where you break up the problem (or problem space) into objects, or perhaps it could be thought of as building up your solution conceptually into objects. The function-oriented design relies on identifying functions that transform their inputs to create outputs, whereas OOD (Design) is where you break up the problem (or problem space) into objects.

## **Function Oriented Design Strategies**

Function Oriented Design Strategies are as follows:

- **Entity Relationship Diagram**
  - **Decision Tree & Decision Table**
  - **Data Flow Diagrams (DFD)**
  - **Data Dictionary**
  - **Pseudo - Code**
- 
- A Decision Tree provides a visual representation of the processing logic involved in higher cognitive activity, allowing for the appropriate actions. The perimeters of a choice tree represent conditions, and the leaf nodes reflect the actions to be taken in response to the condition's test result.
  - A decision table is a valuable tool for dealing with various combinations of inputs and results. It's a black box test design technique for determining complex business logic test scenarios.

The below Image shows an example of a Decision Tree:



## Pseudocodes

- Pseudo code is a term used to describe a made-up programming language. It aids in the development of algorithms.
- It does not employ appropriate syntax or code. It is a combination of regulation and the English language.
- The basics for generating pseudocode are the programme description and function.
- Pseudo-code enables exact specification without worrying programming language syntax

- Pseudocodes already include conceptual material and reduce programming time.
- By becoming sufficiently precise in the DFD, developers and designers can write pseudocode, a hybrid of English and coding.

## **How to Write Pseudocode**

- Always capitalize the initial word (often one of the main six constructs).
- Make only one statement per line.
- Indent to show hierarchy, improve readability, and show nested constructs.
- Always end multi-line sections using any of the END keywords (ENDIF, ENDWHILE, etc.).
- Keep your statements programming language independent.
- Use the naming domain of the problem, not that of the implementation.

For instance: “Append the last name to the first name” instead of “name = first+ last.”

Object-oriented Design	Functional-oriented design
The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.	The basic abstractions, which are given to the user, are real world functions.
Functions are grouped together on the basis of the data they operate since the classes are associated with their methods.	Functions are grouped together by which a higher level function is obtained. an eg of this technique is SA/SD
In this approach the state information is not represented in a centralized memory but is implemented or distributed among the objects of the system.	In this approach the state information is often represented in a centralized shared memory.
OOD approach is mainly used for evolving systems which mimics a business process or business case.	FOD approach is mainly used for computation sensitive application.
we decompose in class level	we decompose in function/procedure level
Bottom up approach	Top down Approach
Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis.	It views the system as Black Box that performs high level function and later decomposes it into a detailed function so to be mapped to modules.
Begins by identifying objects and classes.	Begins by considering the use case diagram and Scenarios.



# Software Design Principles

1)**Modularity (divide and conquer ):-** Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently.

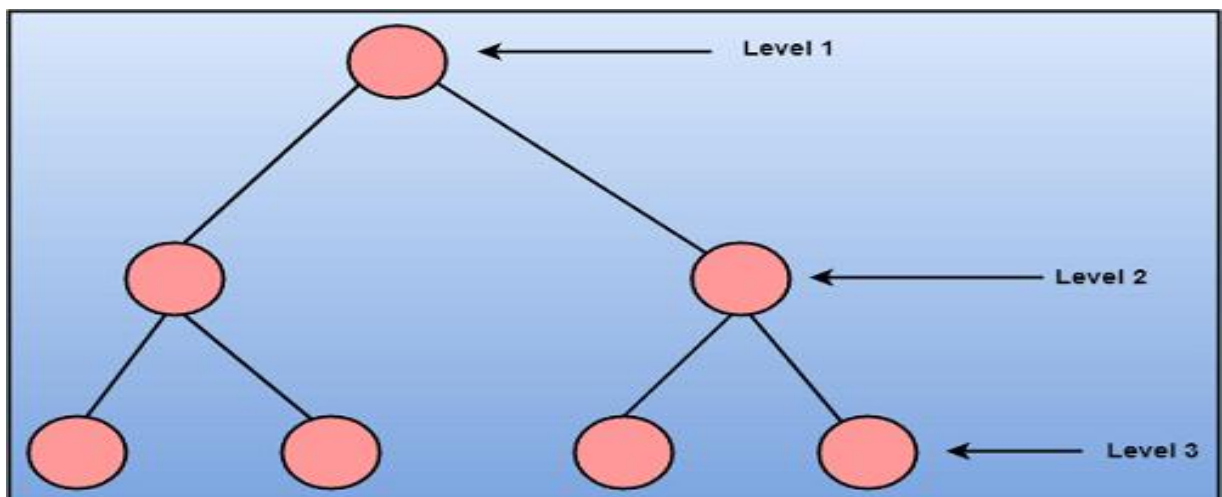
2)**Abstraction:-** An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.

3)**Problem Partitioning:-** For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

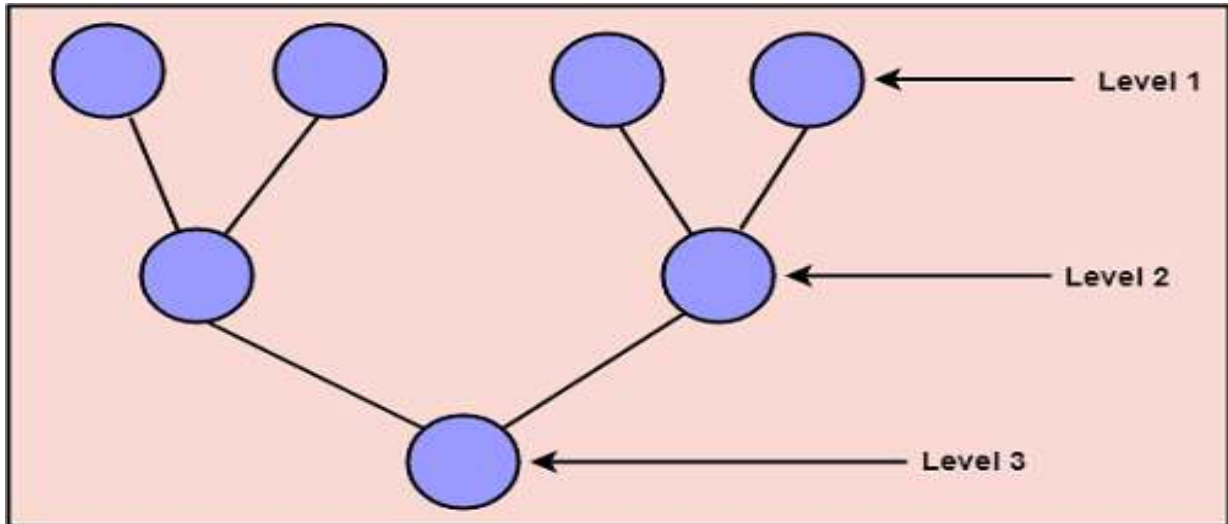
To design a system, there are two possible approaches:

- Top-down Approach
- Bottom-up Approach

1. **Top-down Approach:** This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.



**2. Bottom-up Approach:** A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.



## Coupling and Cohesion

Coupling and Cohesion are two key concepts in software engineering that are used to measure the quality of a software system's design.

There are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion.

1) **Cohesion**:-Cohesion is a measure that defines the degree of intra-dependability

- There are seven types of cohesion-

1) **Co-incident cohesion** - It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.

2) **Logical cohesion** - When logically categorized elements are put together into a module, it is called logical cohesion.

3) **Temporal Cohesion** - When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.

4) **Procedural cohesion** - When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.

5) **Communicational cohesion**- When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.

6) **Sequential cohesion** - When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.

7) **Functional cohesion** - It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

## **Coupling**

Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

- There are five levels of coupling-

1) **Data coupling**- Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

2) **Stamp coupling**- When multiple modules share common data structure and work on different part of it, it is called stamp coupling.

3) **Control coupling**- Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.

4) **Common coupling**- When multiple modules have read and write access to some global data, it is called common or global coupling.







## Design Verification

- The output of software design process is design documentation, pseudo codes, detailed logic diagrams, process diagrams, and detailed description of all functional and non-functional requirements.

## Structure Chart

- A structure chart in software engineering and organizational theory is a chart which shows the breakdown of a system to its lowest manageable levels (Black boxes-Functionality is known to user but inner details are not known).
- They are used in structured programming to arrange program modules into a tree.
- Modules at top level call modules at lower level.
- Each module is represented by a box, which contains the module's name. Structured Chart is a graphical representation which shows:
  1. System partitions into modules
  2. Hierarchy of component modules
  3. The relation between processing modules
  4. Interaction between modules
  5. Information passed between modules

**The following notations are used in structured chart:**

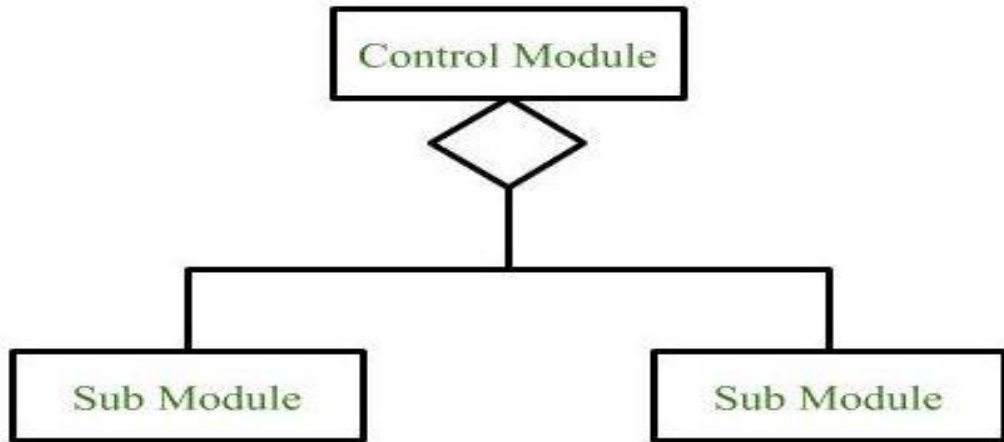
SYMBOL	DESCRIPTION
	Module
	Arrow
	Data couple
	Control Flag
	Loop
	Decision

- **Module**

It represents the process or task of the system.

- **Conditional Call**

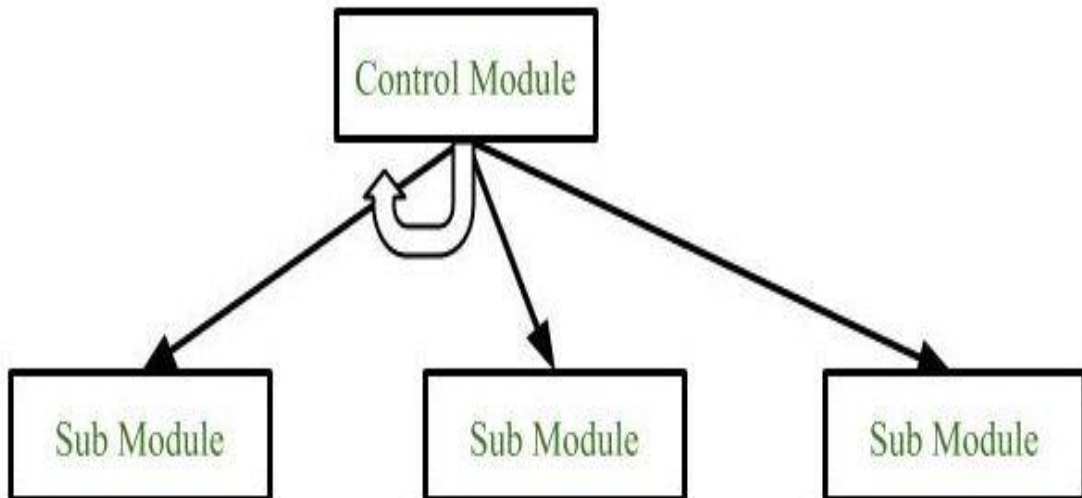
It represents that control module can select any of the sub module on the basis of some condition.



- **Loop (Repetitive call of module)**

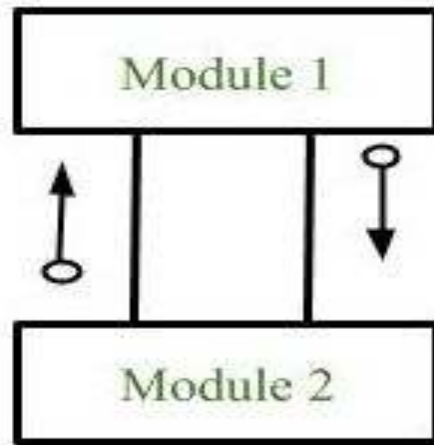
It represents the repetitive execution of module by the sub module.

A curved arrow represents loop in the module.



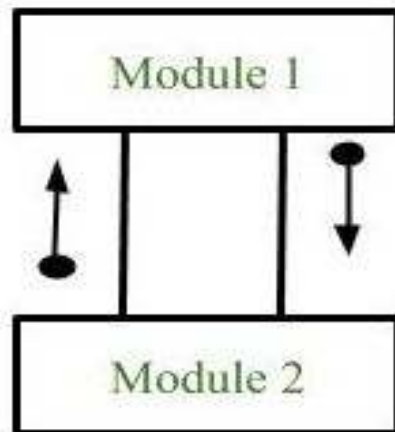
- **Data Flow**

It represents the flow of data between the modules. It is represented by directed arrow with empty circle at the end.



- **Control Flow**

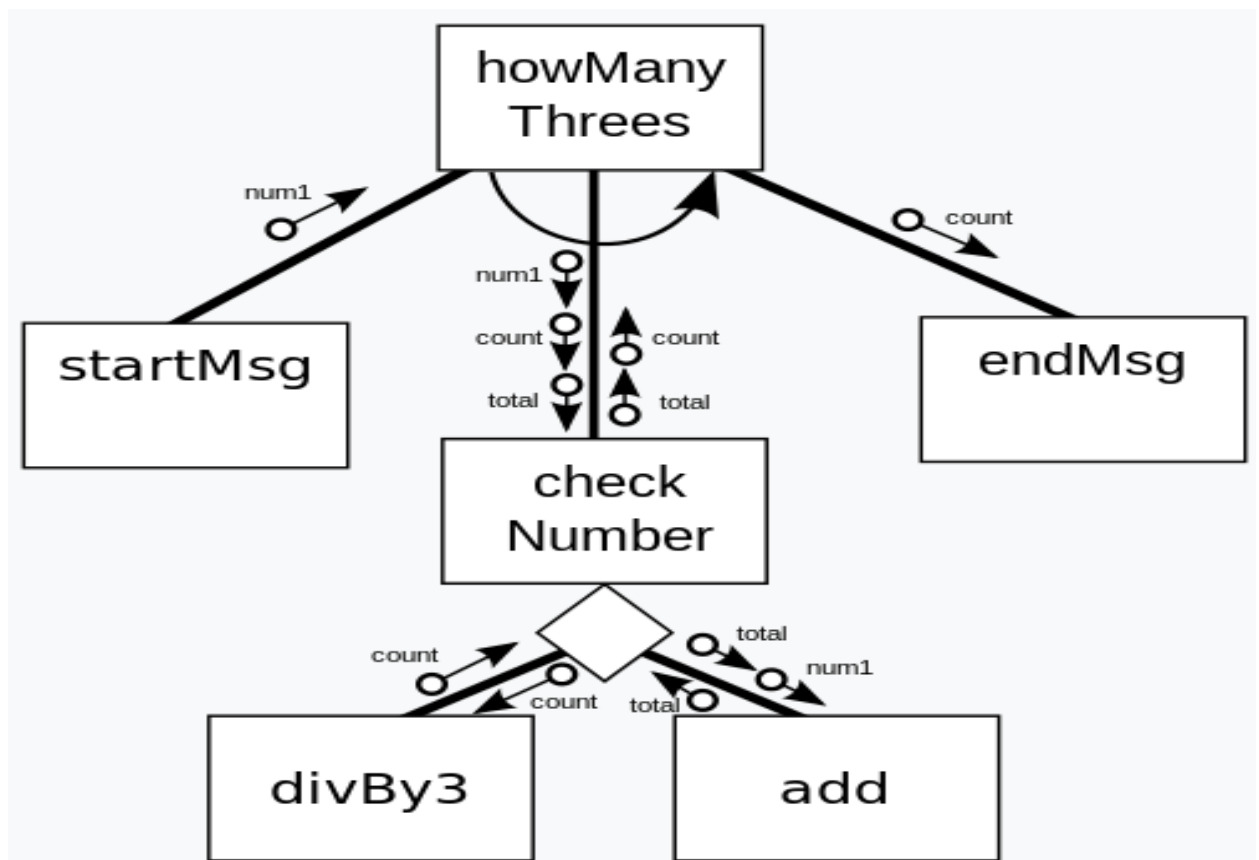
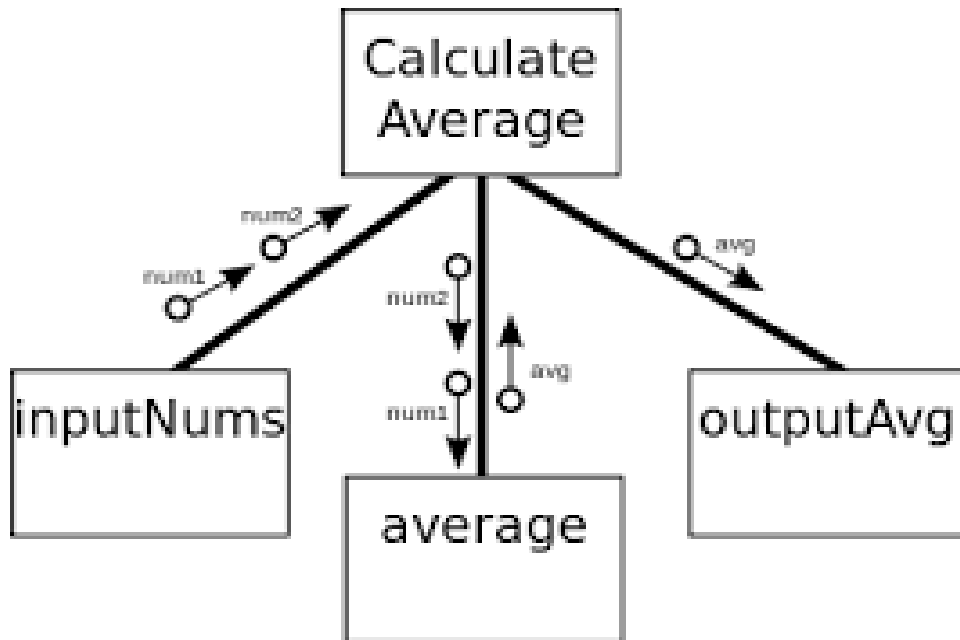
It represents the flow of control between the modules. It is represented by directed arrow with filled circle at the end.



- **Physical storage**



Examples-



# **Software Measurement and Metrics**

- A software measurement deals with the measuring of the size, quantity, amount, or dimension of a particular attribute of a product or process.
- Software Metrics provides the functions by which we can measure the attribute of software system, product or process.
- Software metrics is a standard of measure that contains many activities which involve some degree of measurement. It can be classified into three categories:
  - **Product metrics-** Describe the characteristics of the product such as size, complexity, design features, performance, and quality level.
  - **Process metrics-** Can be used to improve software development and maintenance. Examples include the effectiveness of defect removal during development, the pattern of testing defect arrival, and the response time of the fix process.
  - **Project metrics-** describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

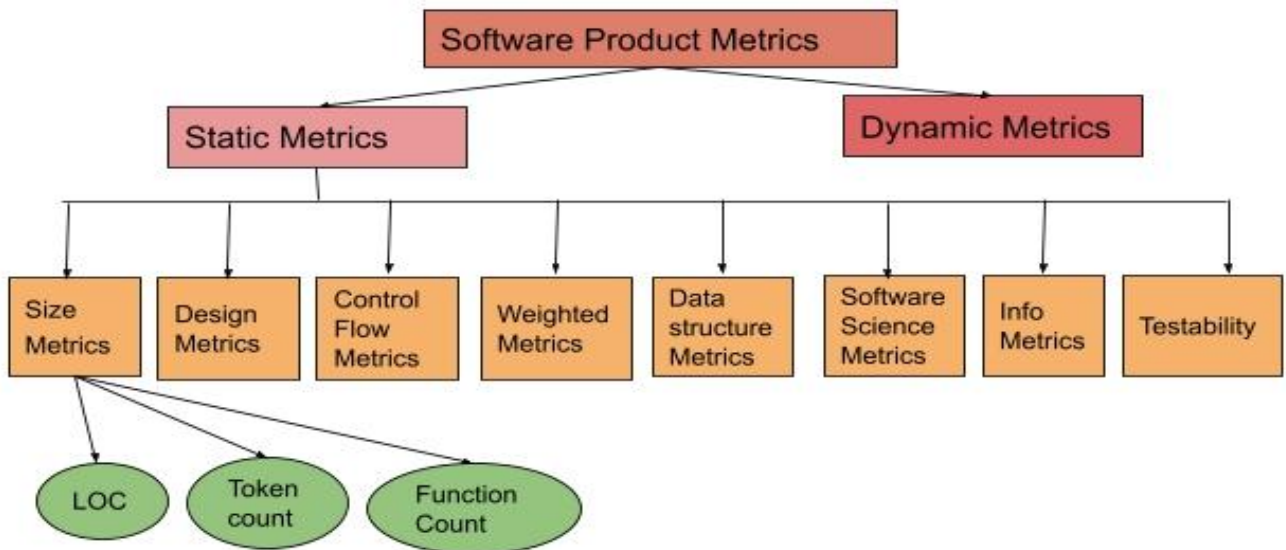
## **Scope of Software Metrics**

- Software metrics contains many activities which include the following –
- Cost and effort estimation
- Productivity measures and model
- Data collection
- Quantity models and measures
- Reliability models
- Performance and evaluation models
- Structural and complexity metrics
- Capability – maturity assessment
- Management by metrics



## Size-Oriented Metrics

•Size-Oriented Metrics concentrates on the size of the programme created. Size-oriented metrics are used to collect direct measures of **software output and quality**. We can calculate Size-Oriented Metrics by averaging out quality and productivity.



•Size-Oriented Metrics can also measure and compare programmers' productivity. It is a direct evaluation of a piece of software. The size estimation is based on the computation of **lines of code**. A line of code is defined as one line of text in a source file.

### Set of Size Measures

- The following is a simple set of size measures that can be developed:
- $\text{Quality} = \text{Errors per KLOC}$
- $\text{Cost} = \$ \text{ per KLOC}$
- $\text{Documentation} = \text{Pages of documentation per KLOC}$
- $\text{Errors} = \text{Errors per person-month}$
- $\text{Productivity} = \text{LOC per person-month}$

- Size-oriented metrics are a direct measure of software and the development process. Effort (time), money spent, KLOC (thousands of lines of code), pages of documentation written, errors, and individuals on the project are examples of these measures.

- The task's output is quantified in terms of LOCs, Function Points, and other metrics.

- **Lines of Code (LOC) :**

The **line of code** (LOC) metric is any line of text in a code that is not a comment or blank line, in any case of the number of statements or fragments of statements on the line. LOC clearly consists of all lines containing program header files, declaration of any variable, and executable and non-executable statements

- **Function Point (FP) :**

In the **function point** metric, the number and type of functions hold up by the software are used to find FPC (function point count).

- A larger unit called a module can estimate the size of a primary software product rather than the LOC measure. A module is a reusable code piece that may be compiled independently. The number of modules in an extensive software system is more accessible to anticipate than the number of lines of code.

### **Function Point (FP)**

1. Function Point metric is specification-based.
2. Function Point metric is language independent.
3. Function Point metric is user-oriented.
4. Function Point metric is extendible to Line of Code.
5. Function Point is used for data processing systems.
6. Function Point can be used to portray the project time.

### **Line of Code (LOC)**

1. LOC metric is based on analogy.
2. LOC metric is dependent on language.
3. LOC metric is design-oriented.
4. It is changeable to FP (i.e., backfiring)
5. LOC is used for calculating the size of the Computer program.
6. LOC is used for calculating and comparing the productivity of programmers.

# Halstead's Software Metrics

- Halstead's software metrics is a set of measures proposed by Maurice Halstead to evaluate the complexity of a software program. These metrics are based on the number of distinct operators and operands in the program, and are used to estimate the effort required to develop and maintain the program.
- In these metrics, a computer program is considered to be a collection of tokens, which may be classified as either operators or operands. All software science metrics can be defined in terms of these basic symbols. These symbols are called as a token.

## Token count

- **Operator** –Any symbol or keyword in a program that specifies an algorithm is considered an operator. For example, arithmetic symbols like (+, -, /, \*), punctuation marks, common names (while, for, print f), special symbols (=, :) and function names.
- **Operand**– In an algorithm or program, a symbol is used to represent data, constants, variables, labels, etc. is considered as an operand.

## The Halstead metrics include the following:

$n_1$  = Number of distinct operators.

$n_2$  = Number of distinct operands.

$N_1$  = Total number of occurrences of operators.

$N_2$  = Total number of occurrences of operands.

- **Program length (N):** This is the total number of operator and operand occurrences in the program. ( $N = N_1 + N_2$ )
- **Vocabulary size (n):** This is the total number of distinct operators and operands in the program. ( $n = n_1 + n_2$ )
- **Program difficulty (D):** This is the ratio of the number of unique operators to the total number of operators in the program, i.e.,  **$D = (n_1/2) * (N_2/n_2)$** ,  **$D = 1 / L$**
- **Program effort (E):** This is the product of program volume (V) and program difficulty (D), i.e.,  **$E = V * D$** .
- **Time to implement (T):** This is the estimated time required to implement the program, based on the program effort (E) and a constant value that depends on the programming language and development environment.

- **Program level (L):** This is the ratio of the number of operator occurrences to the number of operand occurrences in the program, i.e.,  $L = n1/n2$ , where  $n1$  is the number of operator occurrences and  $n2$  is the number of operand occurrences.
- **Programming Time** = Shows time (in minutes) needed to translate the existing algorithm into implementation in the specified program language.  $T = E / (f * S)$

## **Rules**

- Comments are not considered.
- The identifier and function declarations are not considered
- All the variables and constants are considered operands.
- Global variables used in different modules of the same program are counted as multiple occurrences of the same variable.
- Local variables with the same name in different functions are counted as unique operands.
- Functions calls are considered as operators.
- All looping statements e.g., do { ... } while ( ), while ( ) { ... }, for ( ) { ... }, all control statements e.g., if ( ) { ... }, if ( ) { ... } else { ... }, etc. are considered as operators.
- In control construct switch ( ) { case: ... }, switch as well as all the case statements are considered as operators.
- The reserve words like return, default, continue, break, sizeof, etc., are considered as operators.
- All the brackets, commas, and terminators are considered as operators.
- GOTO is counted as an operator and the label is counted as an operand.
- The unary and binary occurrence of “+” and “-” are dealt separately. Similarly “\*” (multiplication operator) are dealt separately.
- In the array variables such as “array-name [index]” “array-name” and “index” are considered as operands and [ ] is considered as operator.

• In the structure variables such as “struct-name, member-name” or “struct-name -> member-name”, struct-name, member-name are taken as operands and ‘.’, ‘->’ are taken as operators. Some names of member elements in different structure variables are counted as unique operands. All the hash directive are ignored.

### **Advantages of Halstead Metrics:**

- It is simple to calculate.
- It measures overall quality of the programs.
- It predicts the rate of error.
- It predicts maintenance effort.
- It does not require the full analysis of programming structure.
- It is useful in scheduling and reporting projects.

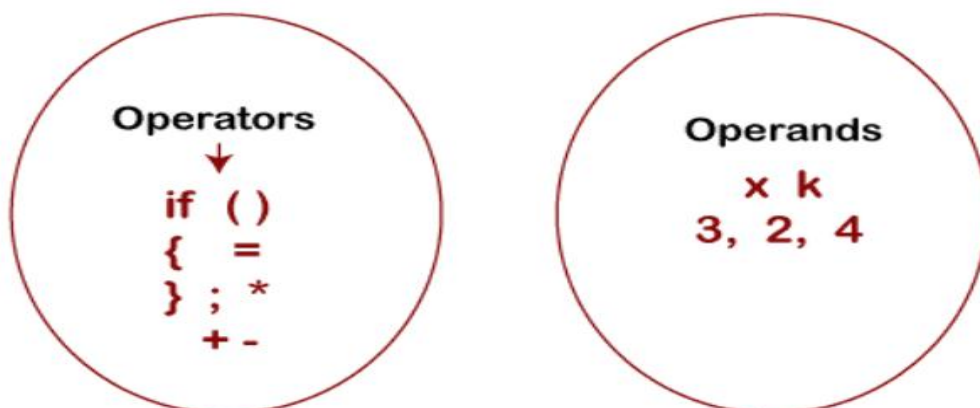
### **Disadvantages of Halstead Metrics:**

- It depends on the complete code.
- It has no use as a predictive estimating model.  
Limited scope
- Limited applicability
- Limited accuracy

• **For**

**example:** •

```
if (k < 2)
{
  k=3;
  x=x*k;
}
```



# Cyclomatic complexity

- Cyclomatic complexity is a measurement developed to determine the stability and level of confidence in a program.
- Programs with lower Cyclomatic complexity are easier to understand and less risky to modify.
- It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.

Cyclomatic complexity =  $E - N + 2 * P$  where,

E = number of edges in the flow graph.

N = number of nodes in the flow graph.

P = number of nodes that have exit points

Steps that should be followed in calculating cyclomatic complexity and test cases design are:

Construction of graph with nodes and edges from code.

- Identification of independent paths.
- Cyclomatic Complexity Calculation
- Design of Test Cases

```
A = 10
```

```
IF B > C THEN
```

```
    A = B
```

```
ELSE
```

```
    A = C
```

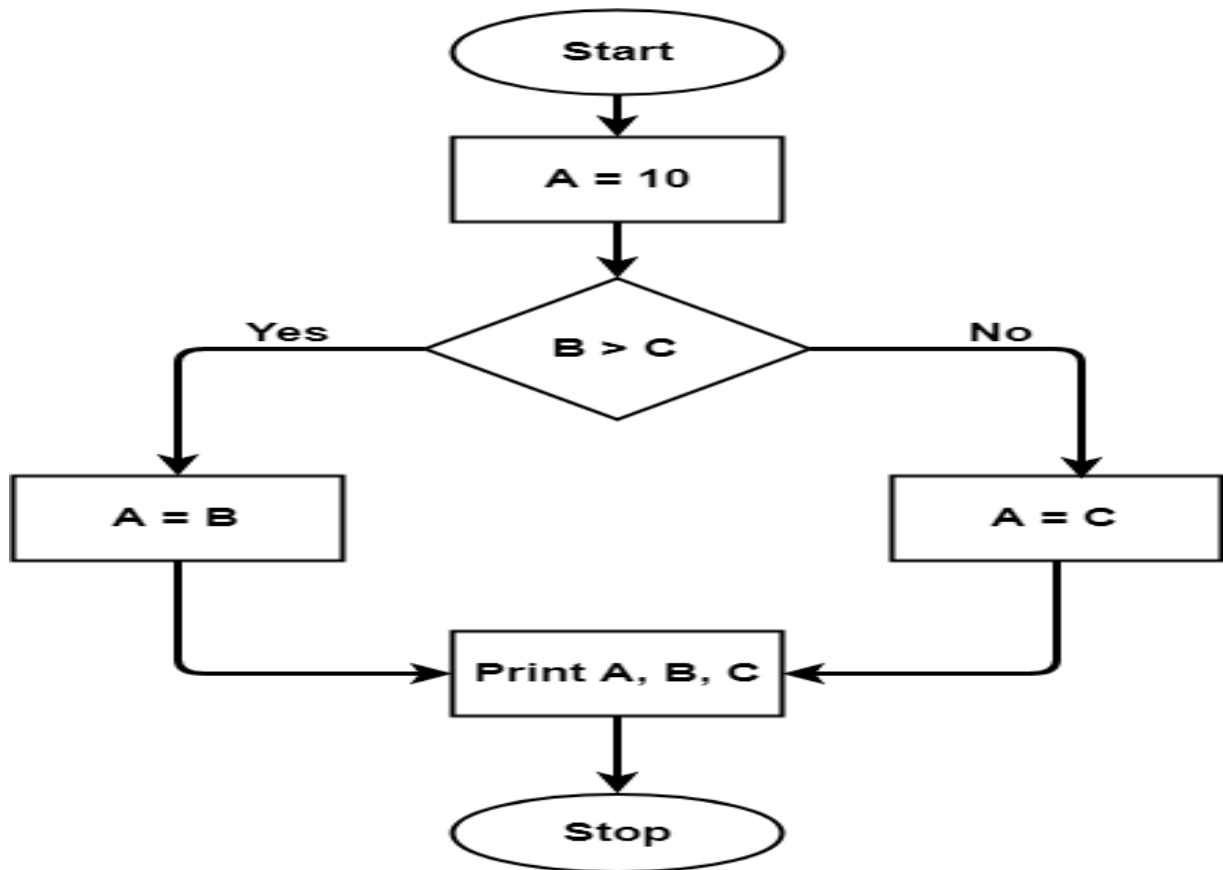
```
ENDIF
```

```
Print A
```

```
Print B
```

```
Print C
```

The graph shows seven shapes(nodes), seven lines(edges), hence cyclomatic complexity is  $7-7+2 = 2$ .



## Use Case Diagram

- A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships.
- It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

