SOFTWARE ENGINEERING

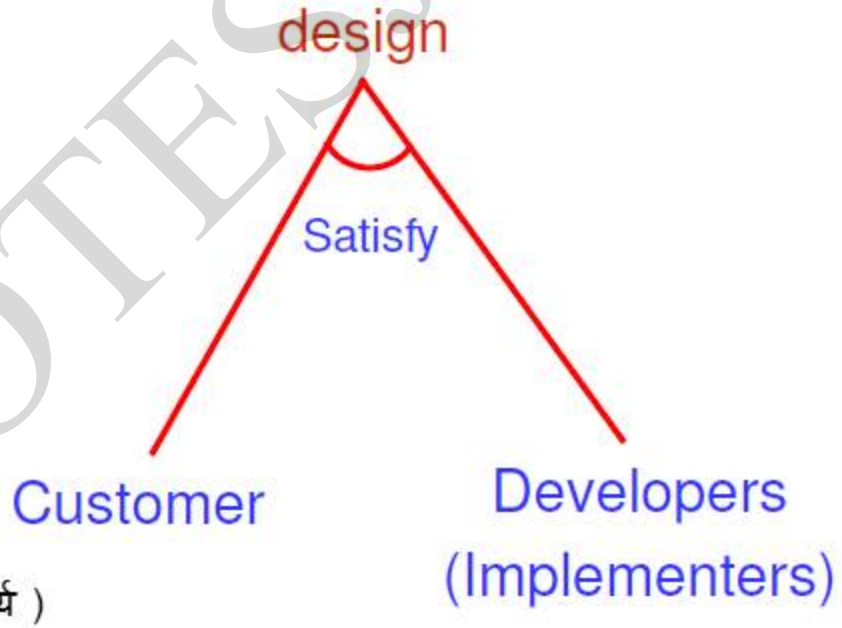
Unit 3: Software Design

Shree Harsh Attri
(Assistant Professor)
Department: Bachelor of Computer Applications

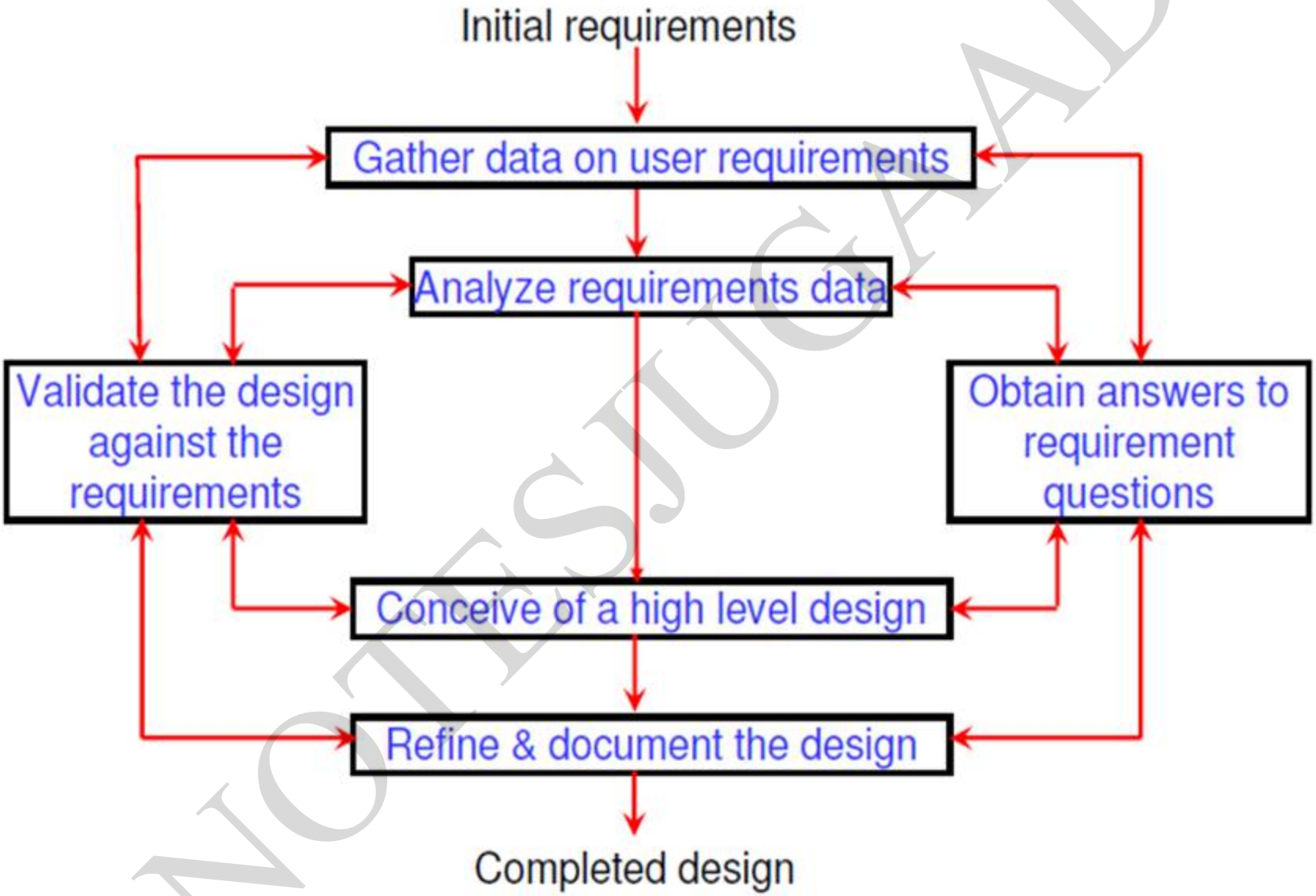
JIMS Engineering Management Technical Campus
Greater Noida (U.P)

Software Design

- Software design is a mechanism to *transform user requirements into some suitable form*, which helps the programmer in software coding and implementation.
- It deals with representing the client's requirement, as described in SRS (Software Requirement Specification) document, into a form, i.e., easily implementable using programming language.



SOFTWARE DESIGN

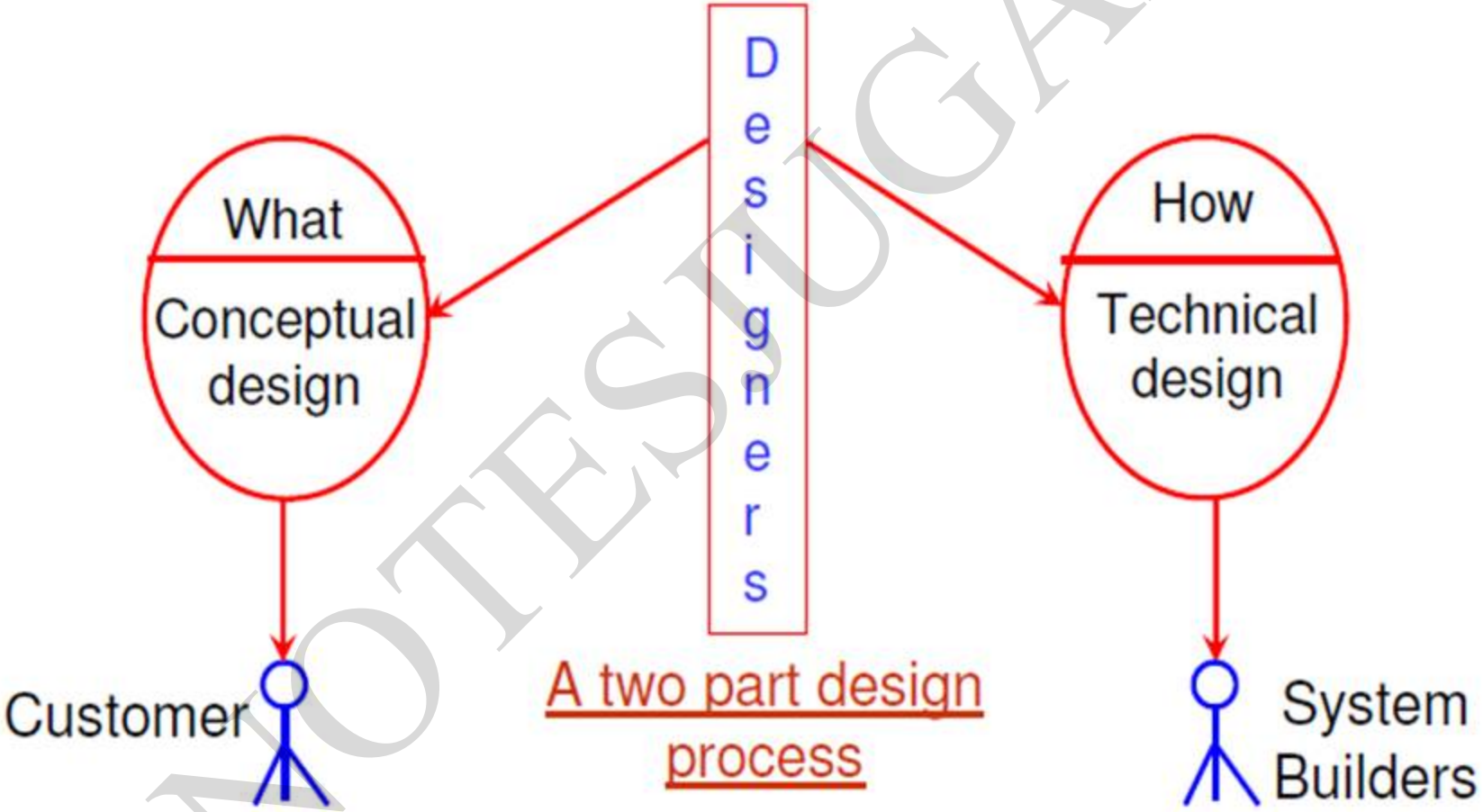


Design Framework

SOFTWARE DESIGN

Two types of Designing Process

Conceptual Design and Technical Design



SOFTWARE DESIGN

jims Greater Noida

Conceptual Design Answers

1. Where will the data come from ?
2. What will happen to data in the system?
3. How will the system look to users?
4. What choices will be offered to users?
5. What is the timings of events?
6. How will the reports & screens look like?

Technical Design Describes

1. Hardware configuration
2. Software needs
3. Communication interfaces
4. I/O of the system
5. Software architecture
6. Network architecture
7. Any other thing that translates the requirements in to a solution to the customer's problem.

SOFTWARE DESIGN

jims Greater Noida

Objectives of Software Design

1. **Correctness:** Software design should be correct as per requirement.
2. **Completeness:** The design should have all components like data structures, modules, and external interfaces, etc.
3. **Efficiency:** Resources should be used efficiently by the program.
4. **Flexibility:** Able to modify on changing needs.
5. **Consistency:** There should not be any inconsistency in the design.
6. **Maintainability:** The design should be so simple so that it can be easily maintainable by other designers.

jims Greater Noida

MODULARITY

1. Modularity specifies to the ***division of software into separate modules*** which are differently named and addressed and are integrated later on in to obtain the completely functional software.
2. It is the only property that allows a program to be intellectually manageable.

The desirable properties of a modular system are:

a) Each module is a well-defined system that can be used with other applications.

b) Each module has single specified objectives.

c) Modules can be separately compiled and saved in the library.

d) Modules should be easier to use than to build.

e) Modules are simpler from outside than inside.

jims Greater Noida

Advantages and Disadvantages of Modularity

Advantages

1. It allows large programs to be written by several or different people.
2. It encourages the creation of commonly used routines to be placed in the library and used by other programs.
3. It simplifies the overlay procedure of loading a large program into main storage.
4. It provides more checkpoints to measure progress.
5. It provides a framework for complete testing, more accessible to test.
6. It produced the well designed and more readable program.

Disadvantages

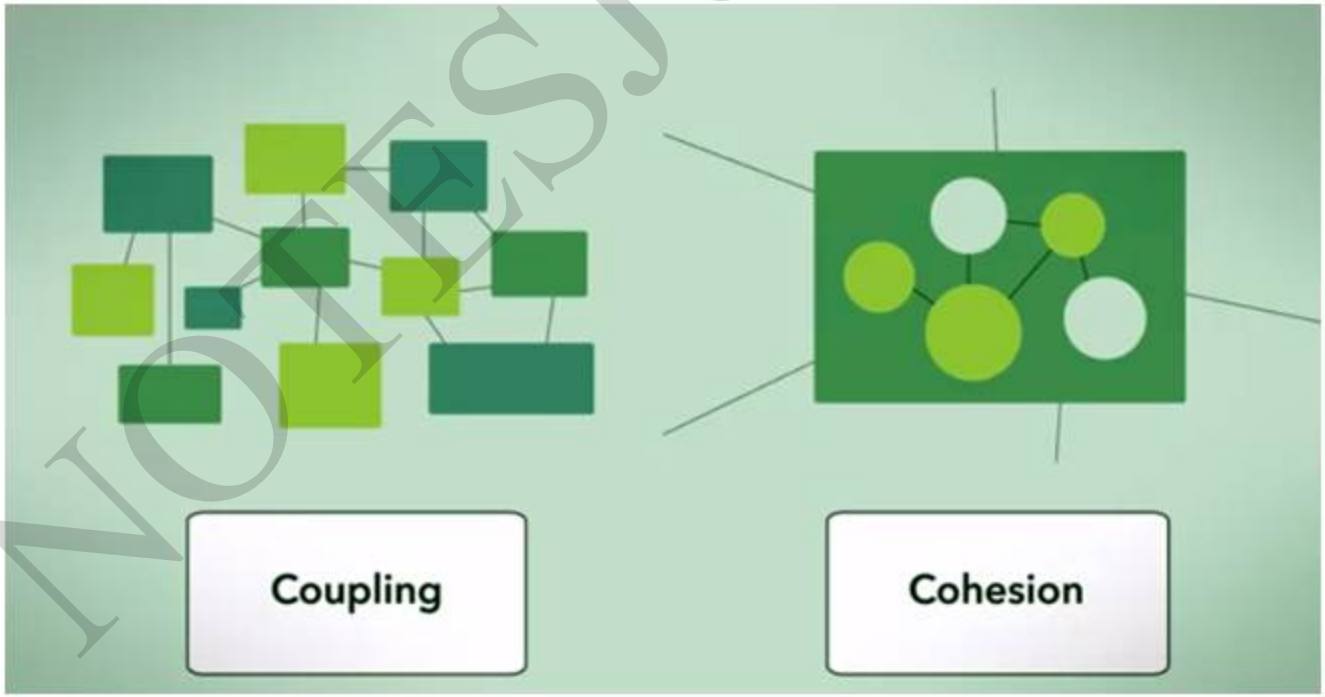
1. Execution time maybe, but not certainly, longer.
2. Storage size perhaps, but is not certainly, increased.
3. Compilation and loading time may be longer.
4. Inter-module communication problems may be increased.
5. More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done.

SOFTWARE DESIGN

Modular Design: reduces the design complexity and results in easier and faster implementation by allowing parallel development of various parts of a system.

It is measured using two criteria:

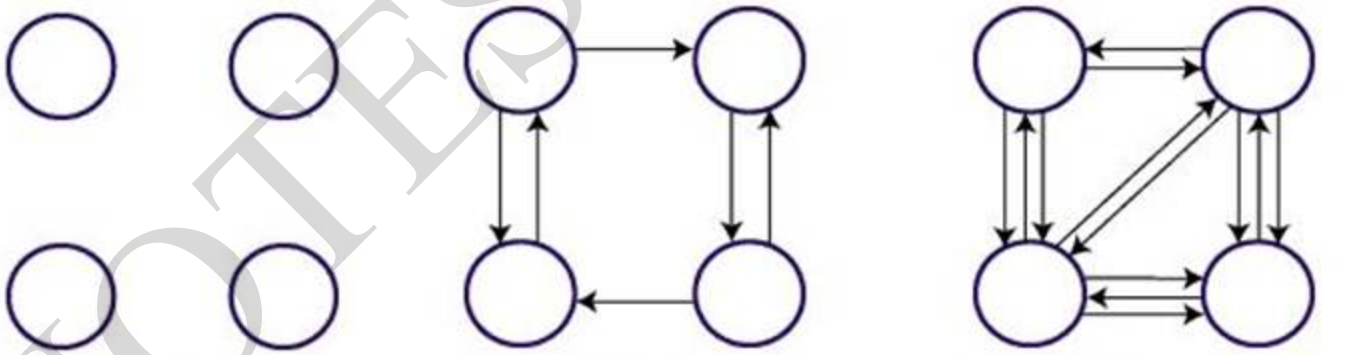
1. **Cohesion:** It measures the relative function strength of a module.
2. **Coupling:** It measures the relative interdependence among modules.



SOFTWARE DESIGN: COUPLING AND COHESION

Module Coupling

1. The coupling is the degree of interdependence between software modules.
2. Two modules that are tightly coupled are strongly dependent on each other.
3. Two modules that are loosely coupled are not dependent on each other.
4. **Uncoupled modules** have no interdependence at all within them.



SOFTWARE DESIGN: COUPLING AND COHESION

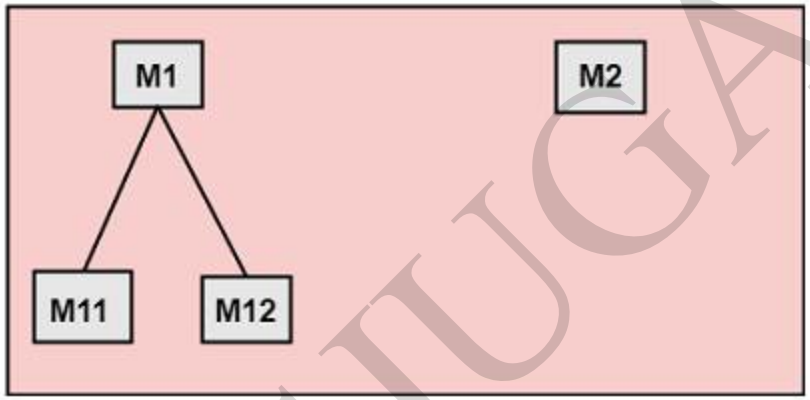
Types of Module Coupling

Types of Coupling	
No Direct Coupling	**Best**
Data Coupling	
Stamp Coupling	
Control Coupling	
External Coupling	
Common Coupling	
Content Coupling	**Worst**

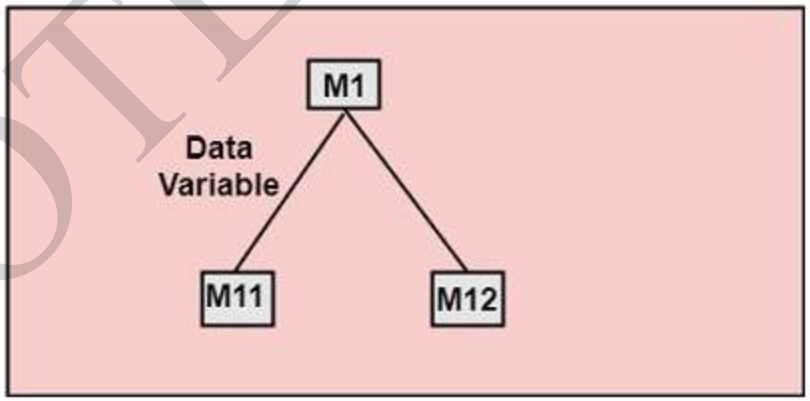
SOFTWARE DESIGN: COUPLING AND COHESION

Types of Module Coupling

1. No Direct Coupling: There is no direct coupling between M1 and M2.



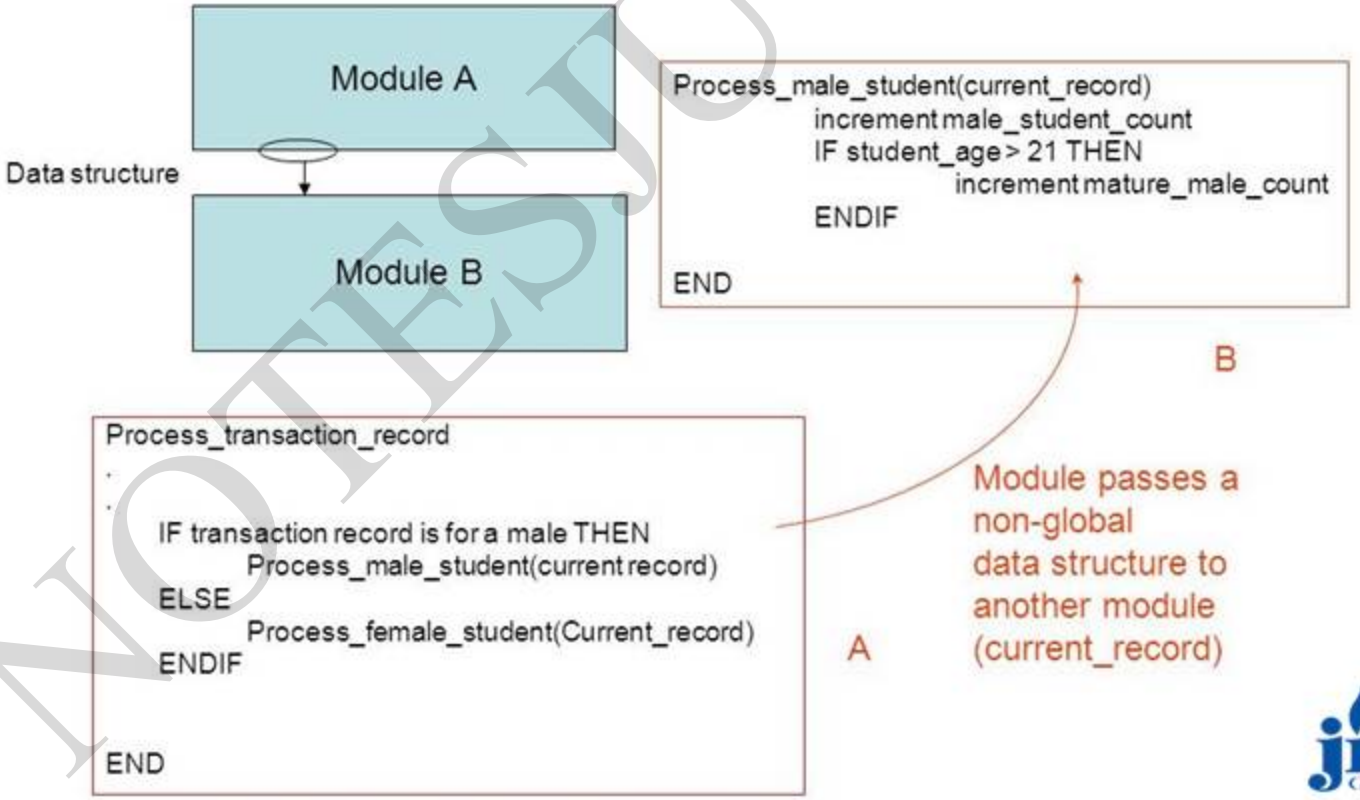
2. Data Coupling: When data of one module is passed to another module, this is called data coupling.



SOFTWARE DESIGN: COUPLING AND COHESION

Types of Module Coupling

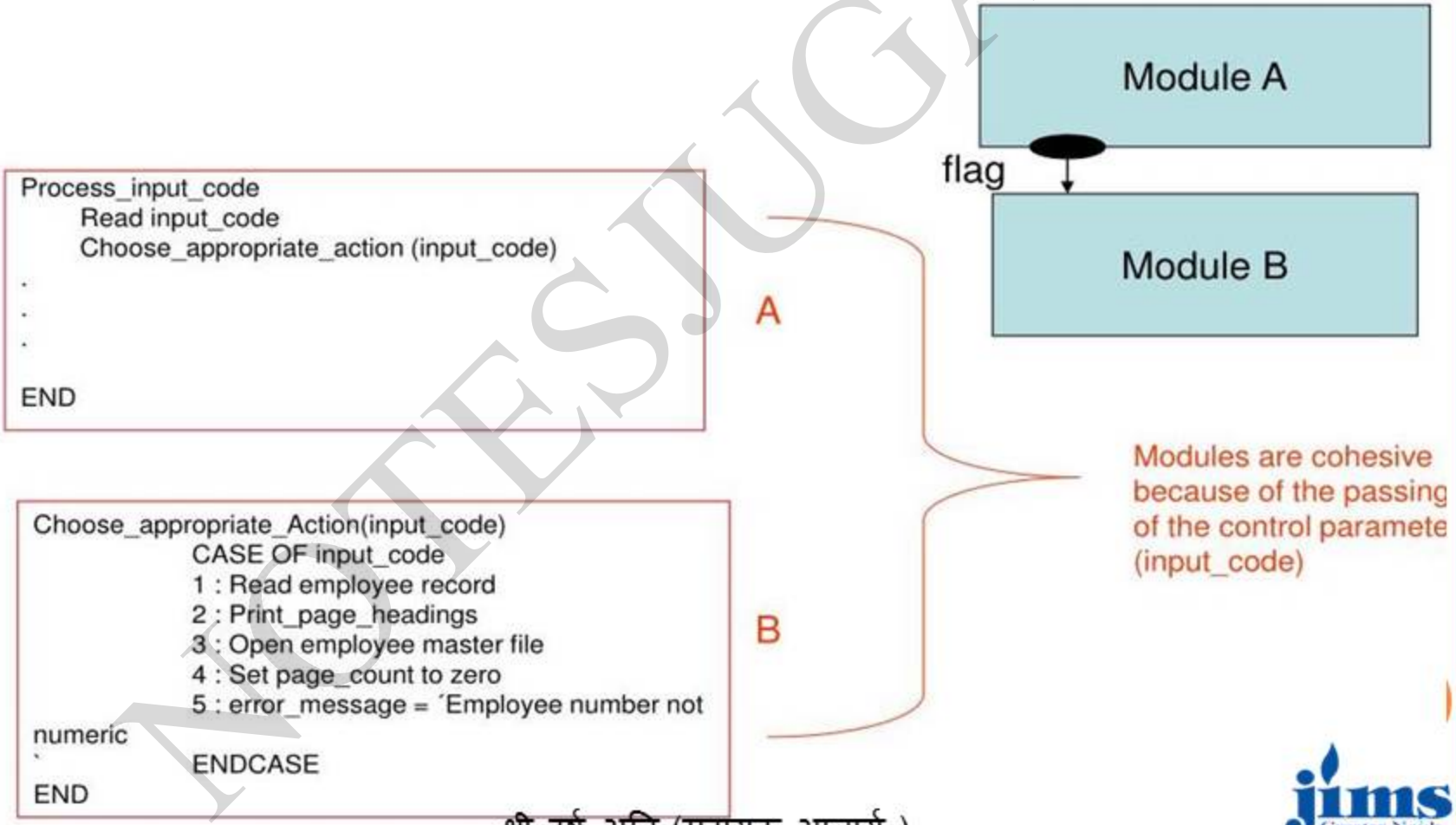
3. **Stamp Coupling:** When multiple modules share common data structure and work on different part of it, is called stamp coupling. For example, passing structure variable in C or object in C++ language to a module.



SOFTWARE DESIGN: COUPLING AND COHESION

Types of Module Coupling

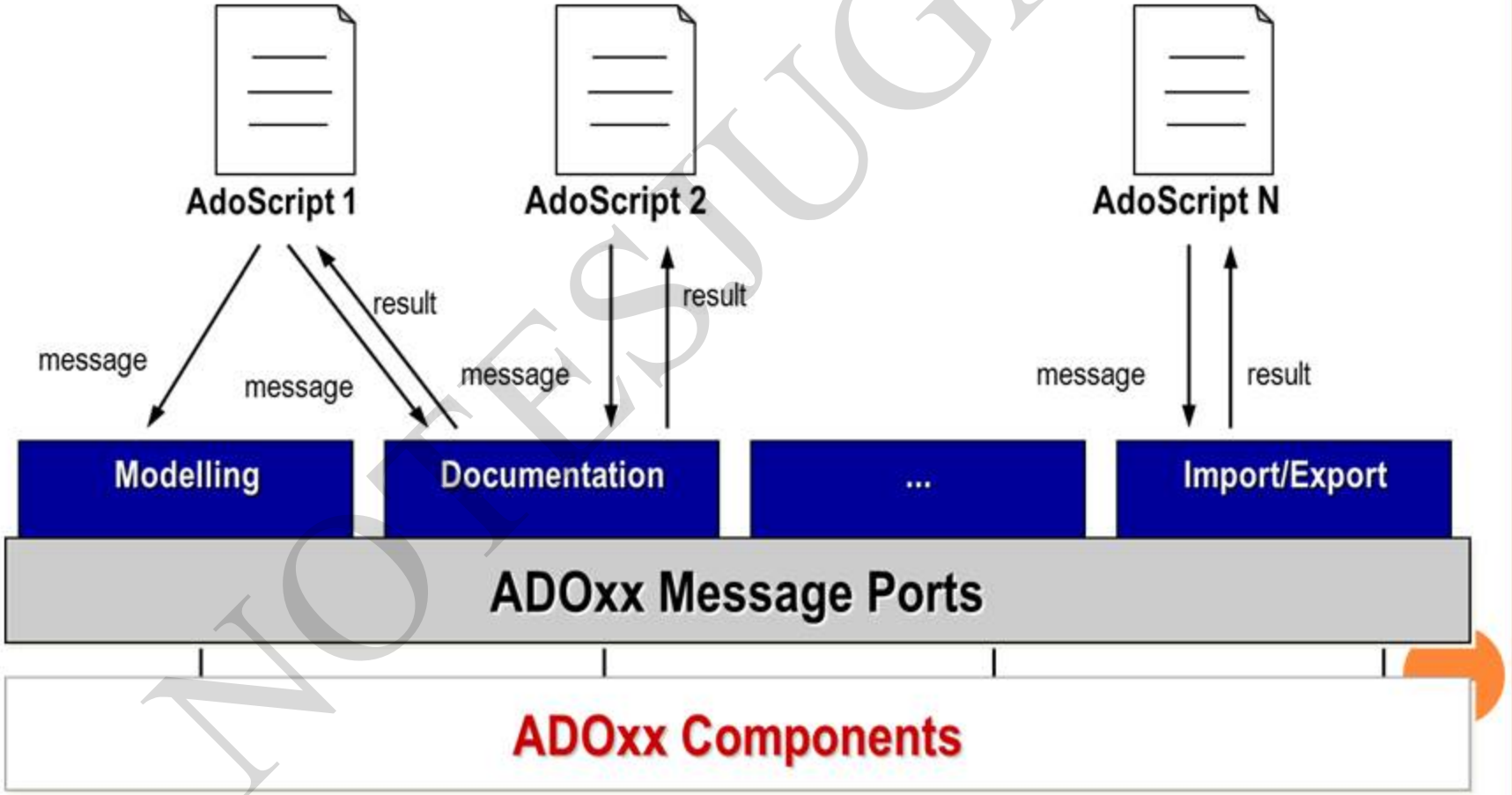
4. **Control Coupling:** Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.



Software Design: Coupling and Cohesion

Types of Module Coupling

5. External Coupling: External Coupling arises when two modules share an externally ***imposed data format, communication protocols, or device interface***. This is related to communication to external tools and devices.



SOFTWARE DESIGN: COUPLING AND COHESION

Types of Module Coupling

6. **Common Coupling:** Two modules are common coupled if they share information through some global data items.


```
Calculate_Sales_tax
        IF product is sales exempt THEN
                sales_tax = 0
        ELSE
                IF product_price < $50.00 THEN
                        sales_tax = product_price * 0.25

                ENDIF
        ENDIF
END

Calculate_amount_due


        amount_due = total_amount + sales_tax
END
```

Modules access the same Global data variable (sales_tax)

jims Greater Noida

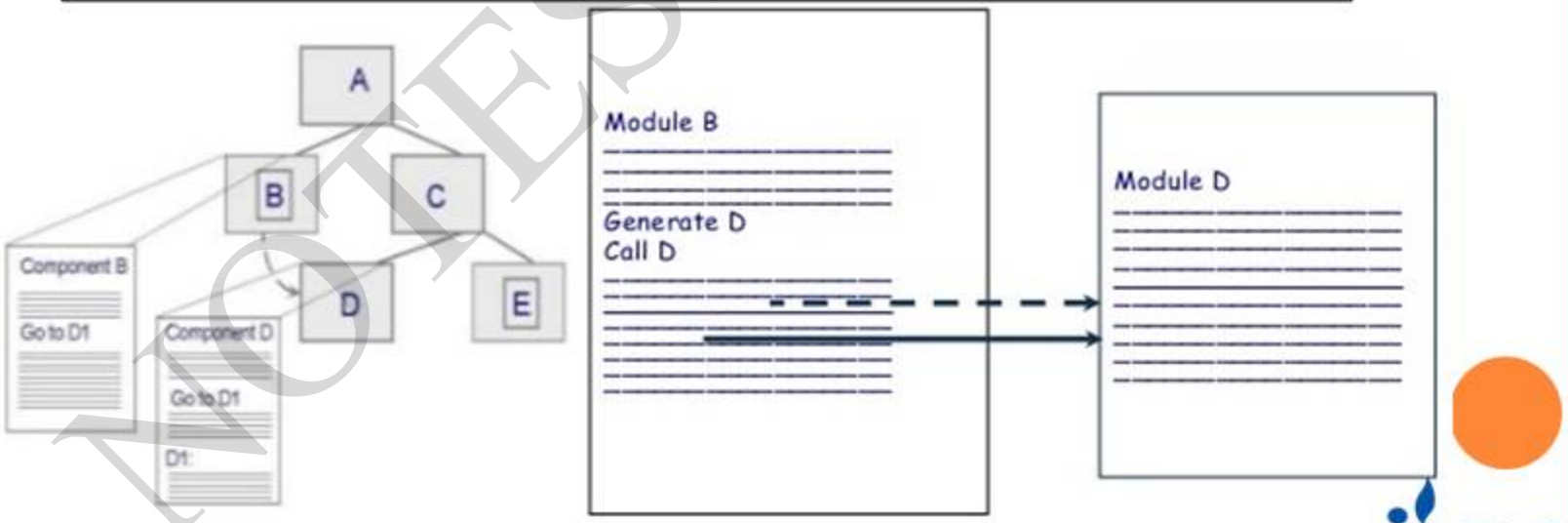
SOFTWARE DESIGN: COUPLING AND COHESION

Types of Module Coupling

7. Content Coupling: Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

Example of Content Coupling

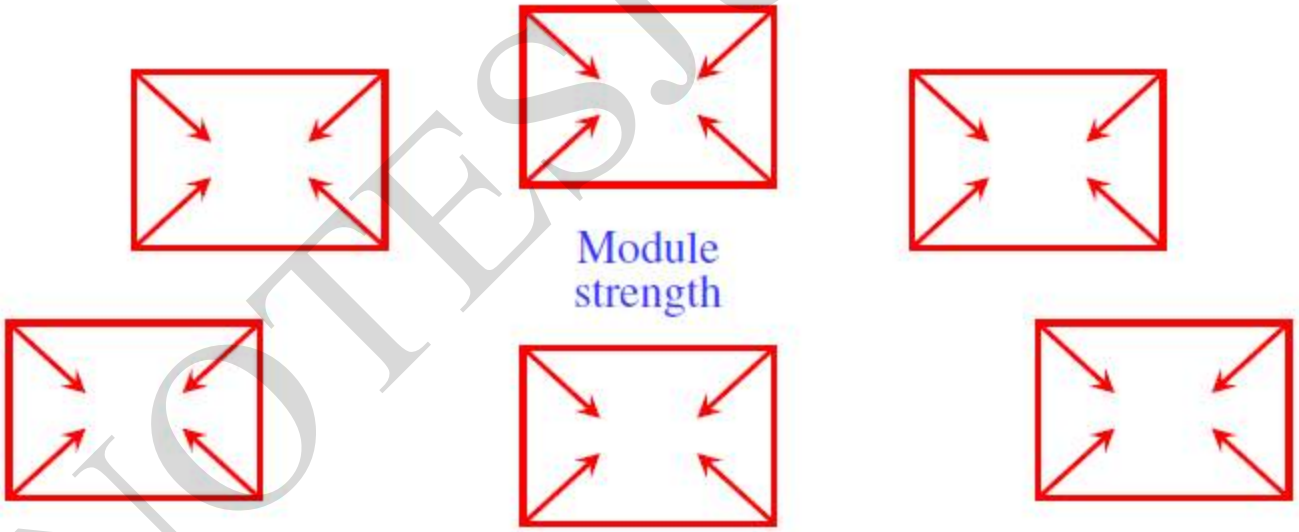
- Occurs when one component modifies an internal data item in another component, or when one component branches into the middle of another component



SOFTWARE DESIGN: COUPLING AND COHESION

Module Cohesion

1. Cohesion defines to the degree to which the elements of a module belong together.
2. Cohesion of a module represents how tightly bound the internal elements of the module are to one another.
3. Cohesion measures the strength of relationships between pieces of functionality within a given module. For example, in highly cohesive systems, functionality is strongly related.

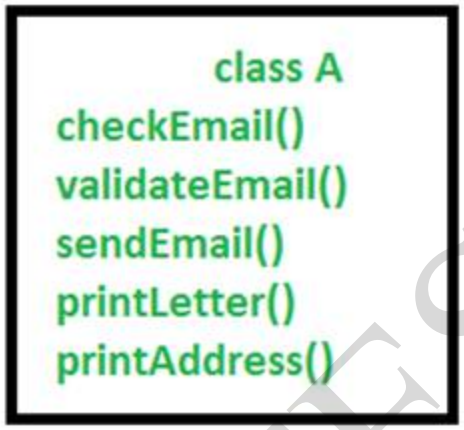


Cohesion = Strength of relations within modules

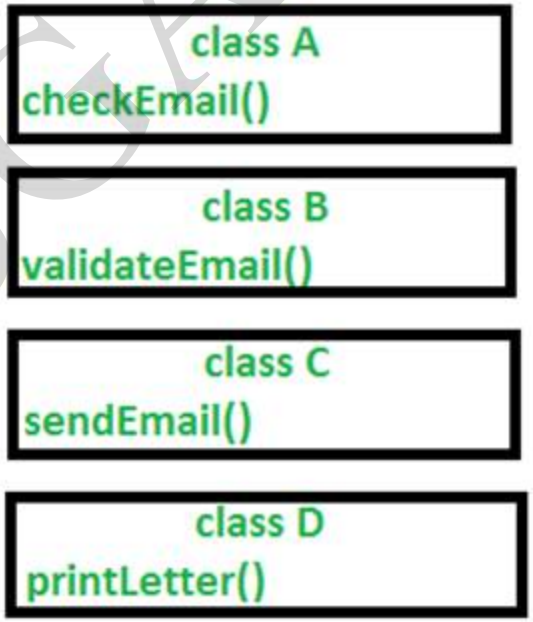
SOFTWARE DESIGN: COUPLING AND COHESION

Module Cohesion

Cohesion is an **ordinal** type of measurement and is generally described as ***"high cohesion"*** or ***"low cohesion"***.



Low Cohesion



High Cohesion

SOFTWARE DESIGN: COUPLING AND COHESION

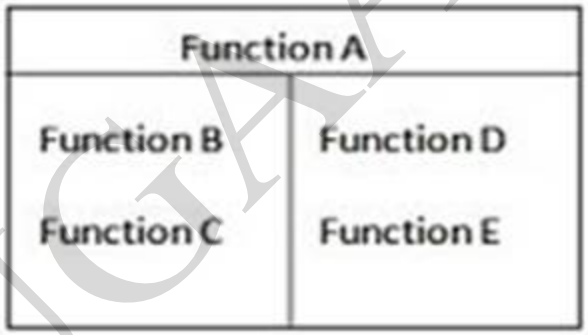
Types of Module Cohesion

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

SOFTWARE DESIGN: COUPLING AND COHESION

Types of Module Cohesion

1. **Co-Incidental 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.



Coincidental
(Parts Unrelated)

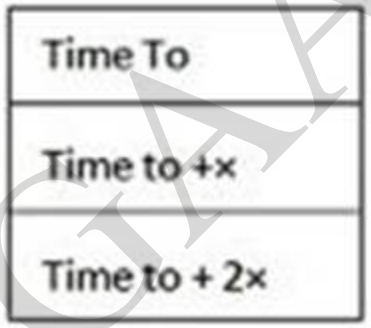
2. **Logical Cohesion** - all the elements of the module perform a similar operation because they fall into the same logical class of functions. For example *Error handling, data input and data output*, etc.

Logical
(Similar Functions)

SOFTWARE DESIGN: COUPLING AND COHESION

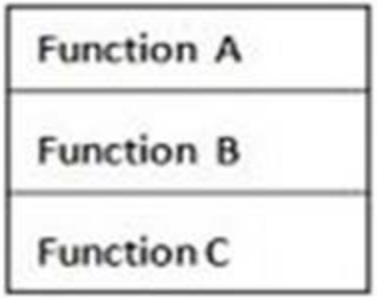
Types of Module Cohesion

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.



Temporal
(Related by Time)

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.

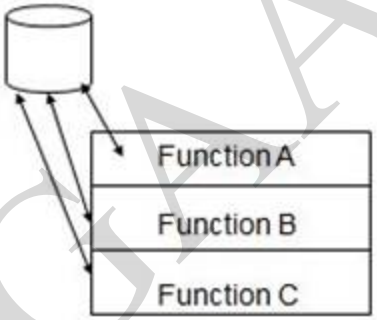


Functions
(Related by order
of functions)

SOFTWARE DESIGN: COUPLING AND COHESION

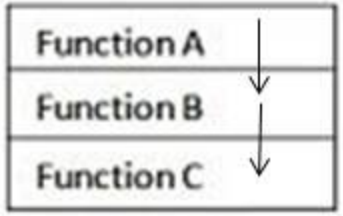
Types of Module Cohesion

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.



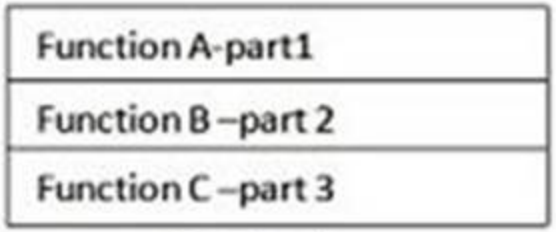
Communicational
(Access same data)

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.



Sequential
(Output of one part is input to next)

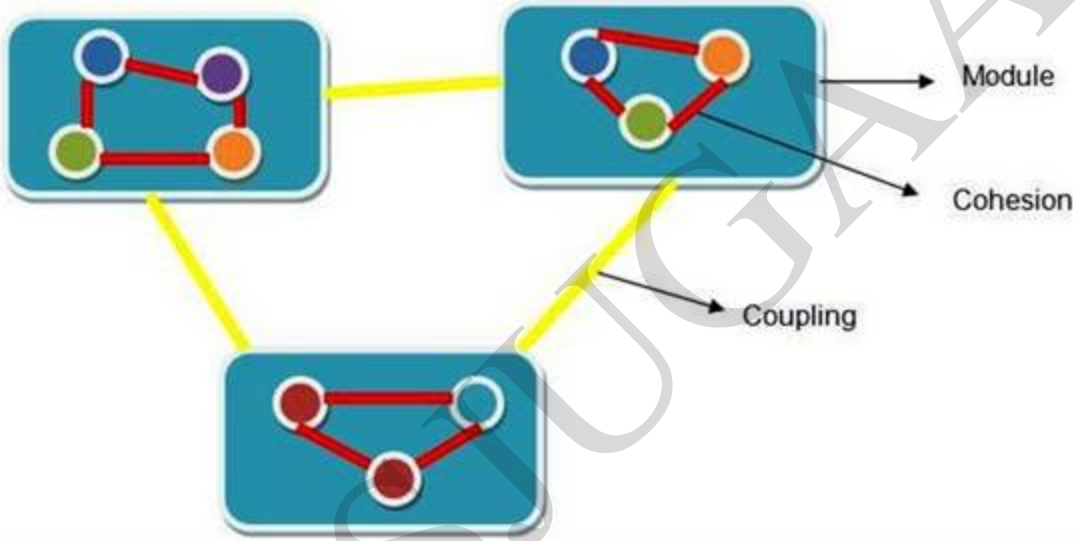
7. **Functional Cohesion** - It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

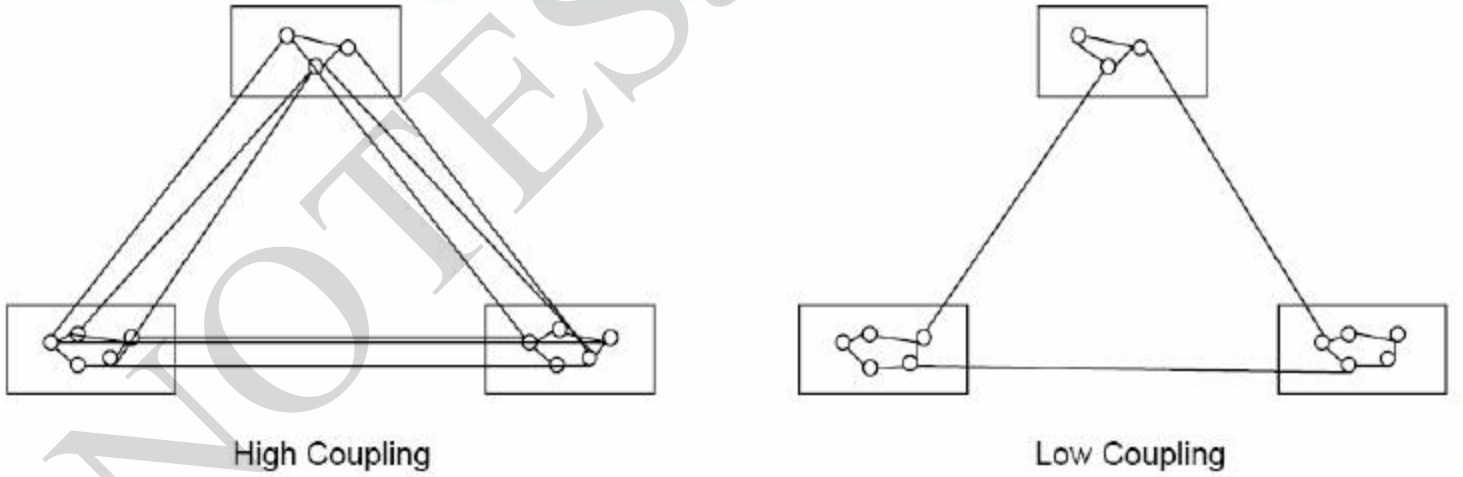


Functional
(Sequential with complete, related functions)

SOFTWARE DESIGN: COUPLING AND COHESION

Relationship between Coupling and Cohesion





A software engineer must design the modules with goal of High Cohesion and Low Coupling.

jims Greater Noida

Layered Arrangement of Modules

Neat and Clear arrangement of modules in a hierarchy:

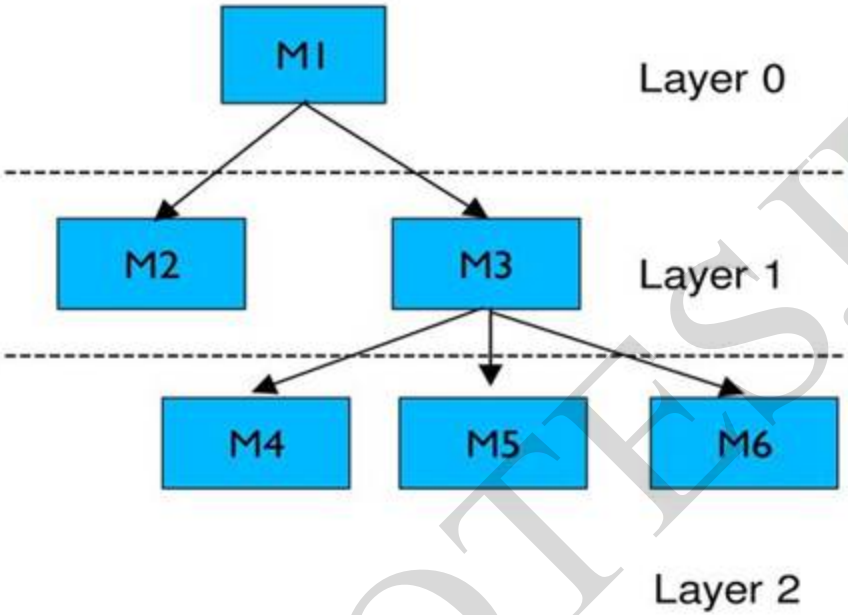
- ✓ Low Fan Out
- ✓ Control Abstraction

Characteristics of Module Hierarchy

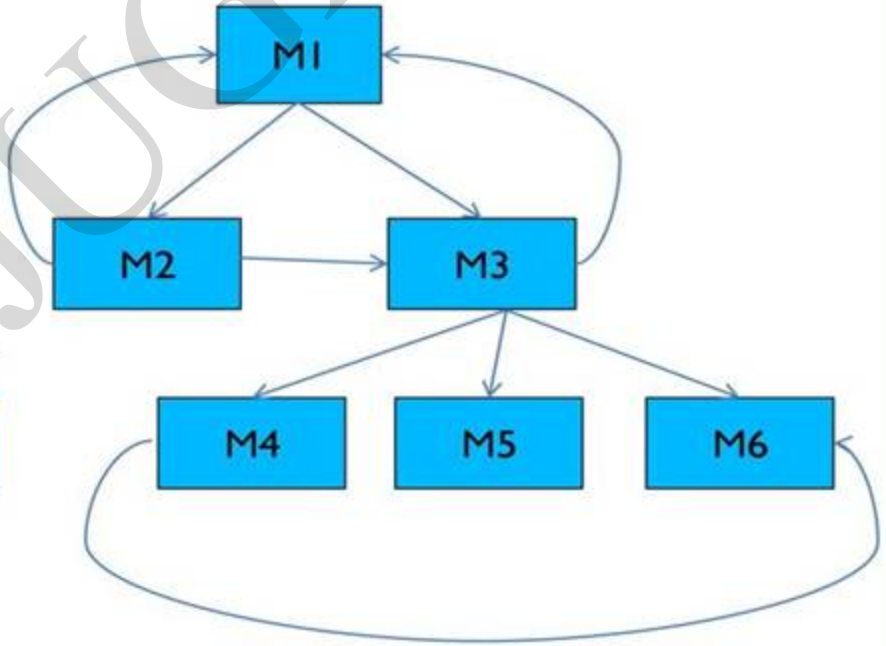
- ✓ **Depth:** Number of levels of control
- ✓ **Width:** Overall span of control.
- ✓ **Fan-Out:** A measure of the number of modules directly controlled by given module.
- ✓ **Fan-In:**
  - Indicates how many modules directly invoke a given module.
  - High fan-in represents code reuse and is in general encouraged.

SOFTWARE DESIGN

LAYERED ARRANGEMENT OF MODULES



Layered design with good Control abstraction



Layered design showing poor Control abstraction

SOFTWARE DESIGN

jims Greater Noida

Function Oriented Design

Function Oriented design is a method to software design where ***the model is decomposed into a set of interacting units or modules where each unit or module has a clearly defined function***. Thus, the system is designed from a functional viewpoint.

Design Notations:

Design Notations are primarily meant to be used during the process of design and are used to represent design or design decisions. For a function-oriented design, the design can be represented graphically or mathematically by the following:

- Data flow diagrams
- Data Dictionaries
- Structure Charts
- Pseudocode

SOFTWARE DESIGN

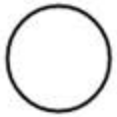
Data Flow Diagram

- Data-flow design is concerned with designing a series of functional transformations that convert system inputs into the required outputs. The design is described as data-flow diagrams.
- These diagrams show how data flows through a system and how the output is derived from the input through a series of functional transformations.
- Data-flow design is an integral part of several design methods, and most CASE tools support data-flow diagram creation. Different ways may use different icons to represent data-flow diagram entities, but their meanings are similar.

SOFTWARE DESIGN

jims Greater Noida

Data Flow Diagram Notations

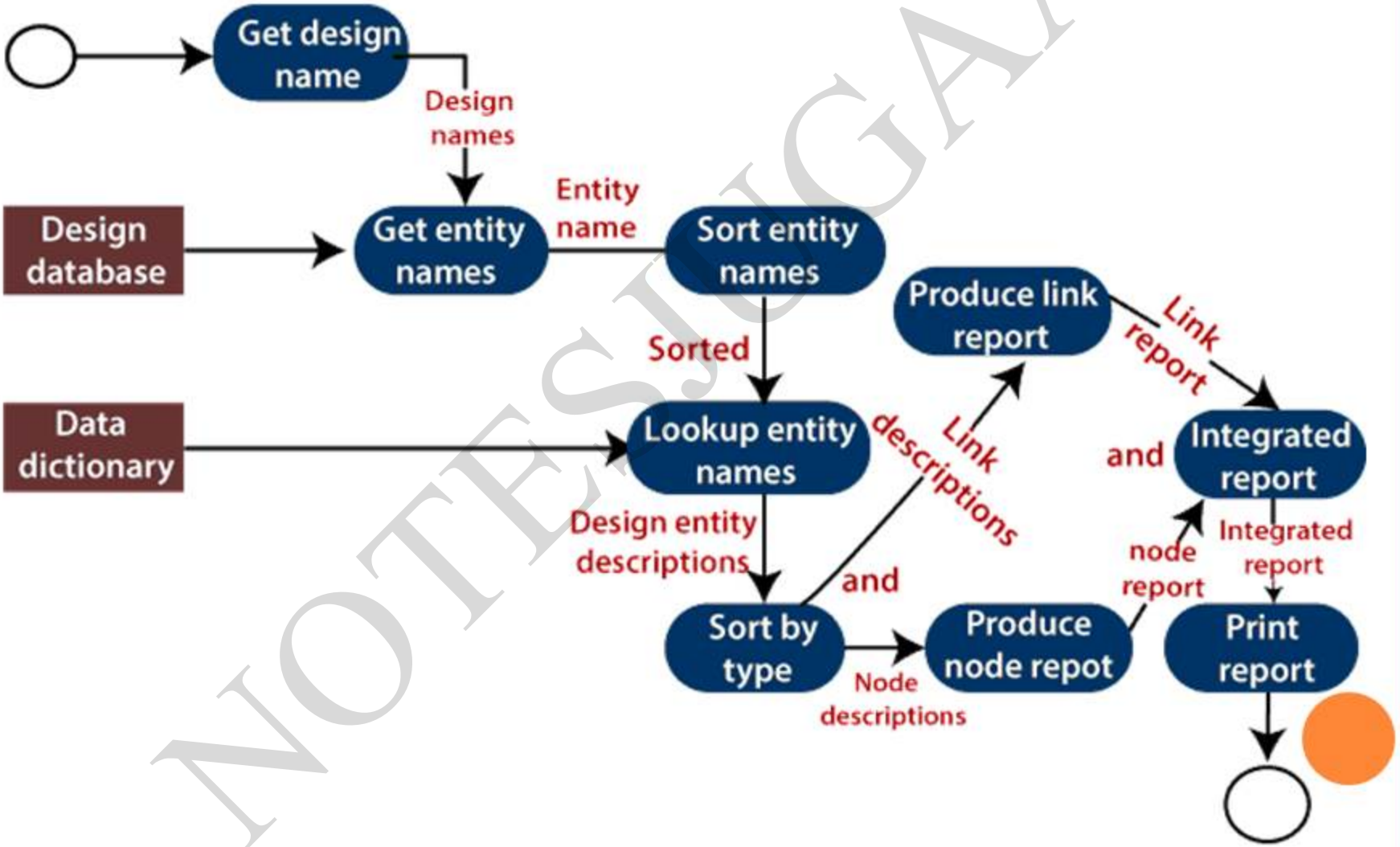
Symbol	Name	Meaning
	Rounded Rectangle	It represents functions which transforms input to output. The transformation name indicates its function.
	Rectangle	It represents data stores. Again, they should give a descriptive name.
	Circle	It represents user interactions with the system that provides input or receives output.
	Arrows	It shows the direction of data flow. Their name describes the data flowing along the path.
"and" and "or"	Keywords	The keywords "and" and "or". These have their usual meanings in boolean expressions. They are used to link data flows when more than one data flow may be input or output from a transformation.

श्री हर्ष अत्रि (सहायक आचार्य)

SOFTWARE DESIGN

jims Greater Noida

Data flow diagram of a design report generator



Data Dictionaries

- A data dictionary lists all data elements appearing in the DFD model of a system. The data items listed contain all data flows and the contents of all data stores looking on the DFDs in the DFD model of a system.
- A data dictionary lists the objective of all data items and the definition of all composite data elements in terms of their component data items. For example, a data dictionary entry may contain that the data *grossPay* consists of the parts *regularPay* and *overtimePay*.

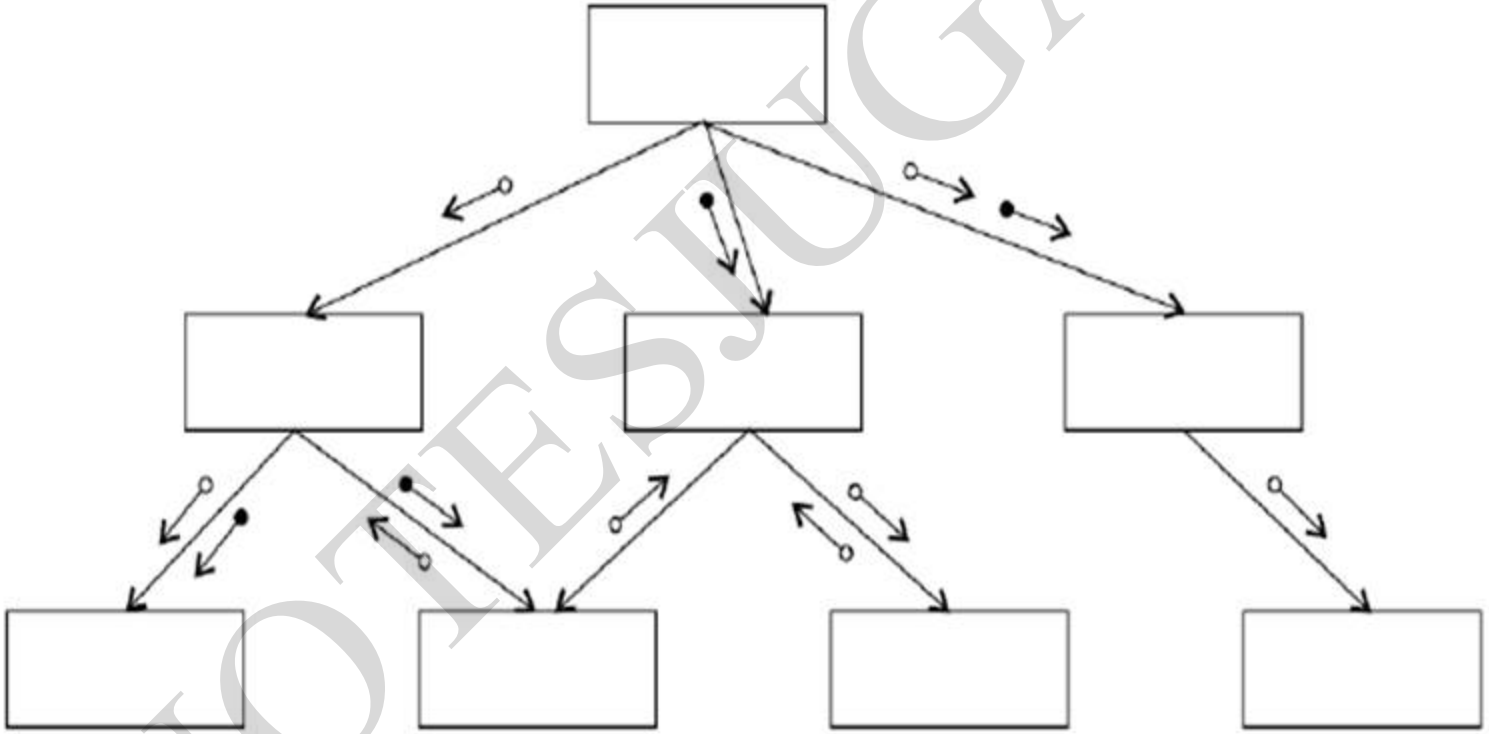
$$\textbf{grossPay} = \textbf{regularPay} + \textbf{overtimePay}$$

A data dictionary plays a significant role in any software development process because of the following reasons:

- A Data dictionary provides a standard language for all relevant information for use by engineers working in a project.
- A consistent vocabulary for data items is essential since, in large projects, different engineers of the project tend to use different terms to refer to the same data, which unnecessarily causes confusion.
- The data dictionary provides the analyst with a means to determine the definition of various data structures in terms of their component elements.

Structure Chart

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



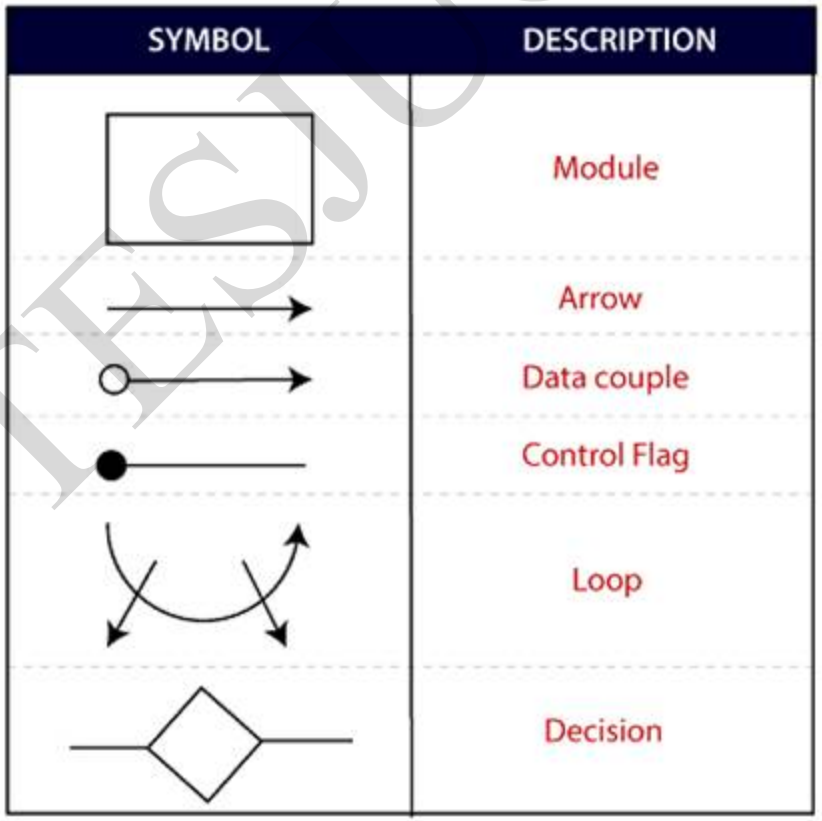
Hierarchical format of a structure chart

SOFTWARE DESIGN – STRUCTURE CHART

Structured Chart is a graphical representation which shows:

- System partitions into modules
- Hierarchy of component modules
- The relation between processing modules
- Interaction between modules
- Information passed between modules

The following notations are used in structured chart:



SOFTWARE DESIGN – STRUCTURE CHART

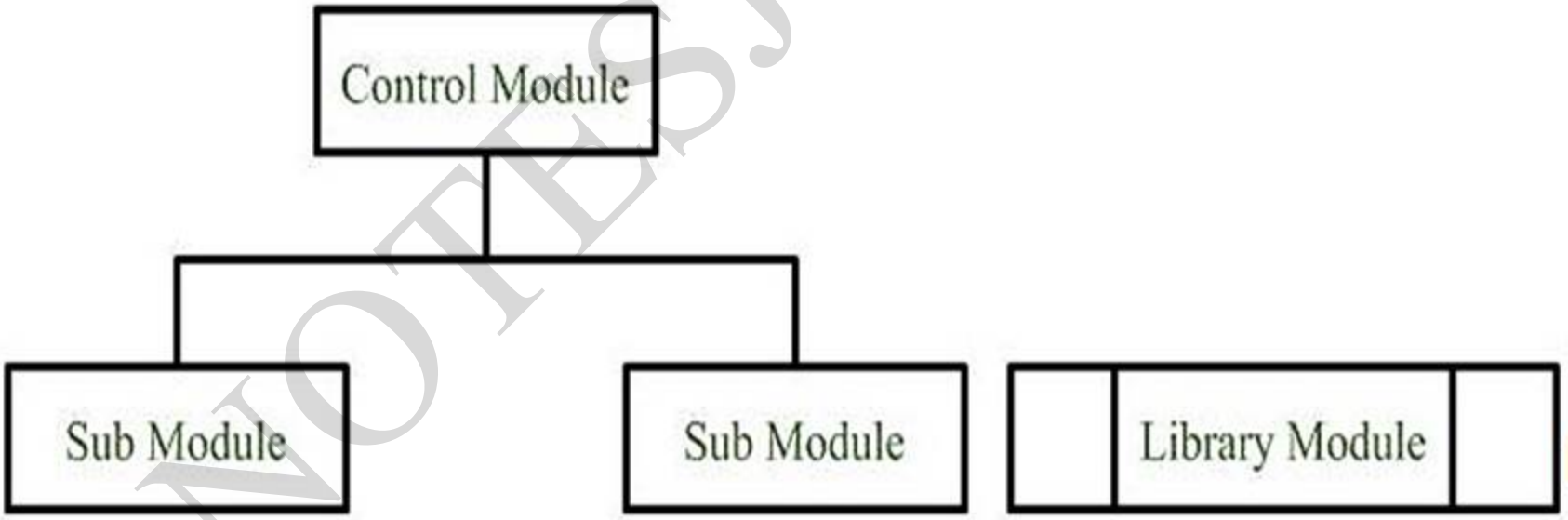
1. **Module**

   It represents the process or task of the system. It is of three types.

   a) **Control Module**: A control module branches to more than one sub module.

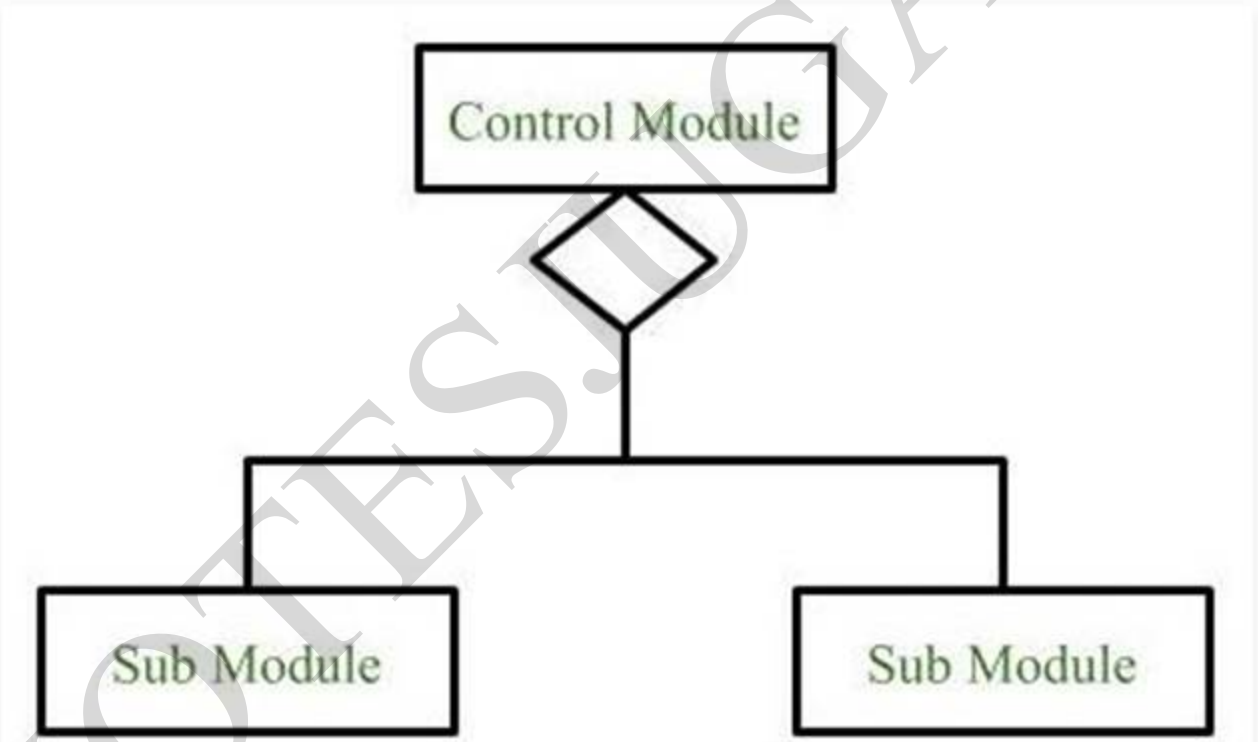
   b) **Sub Module:** Sub Module is a module which is the part (Child) of another module.

   c) **Library Module**: Library Module are reusable and invokable from any module.



2. **Conditional Call**

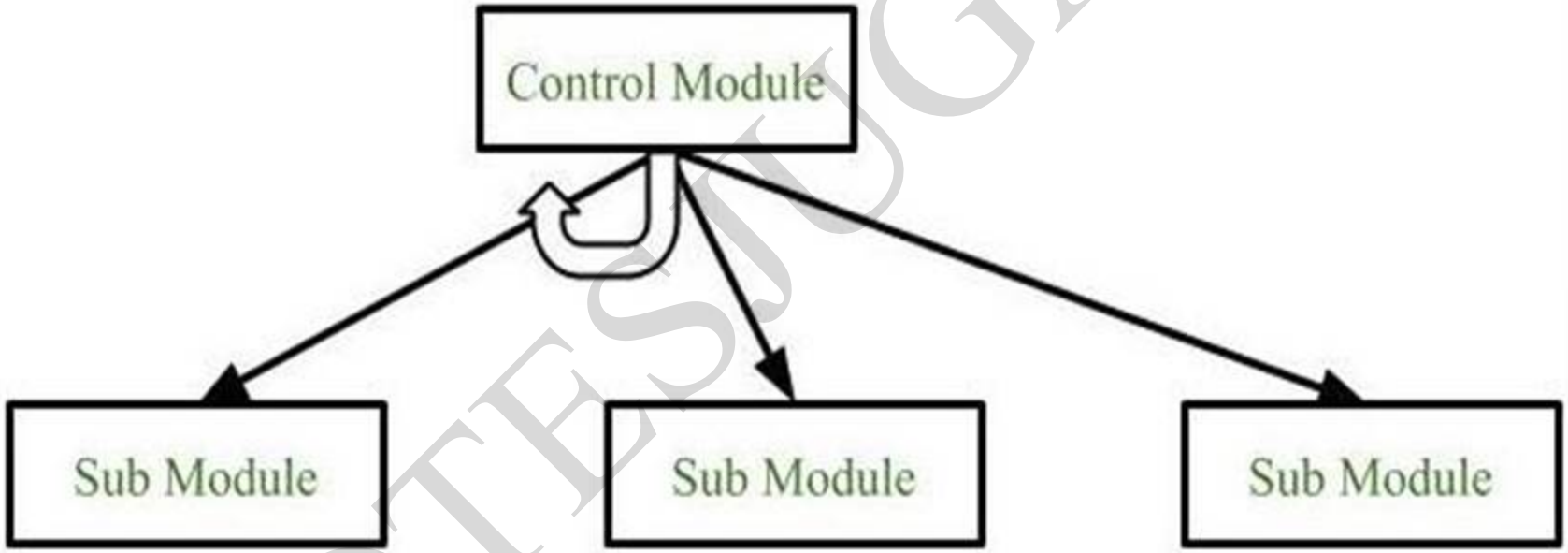
It represents that control module can select any of the sub module on the basis of some condition.



3. **Loop (Repetitive call of module)**

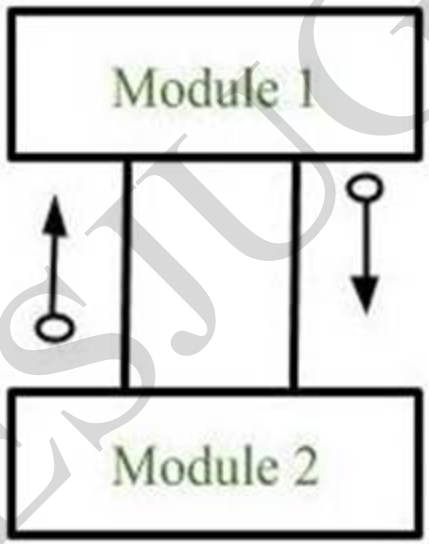
It represents the repetitive execution of module by the sub module.

A curved arrow represents loop in the module.



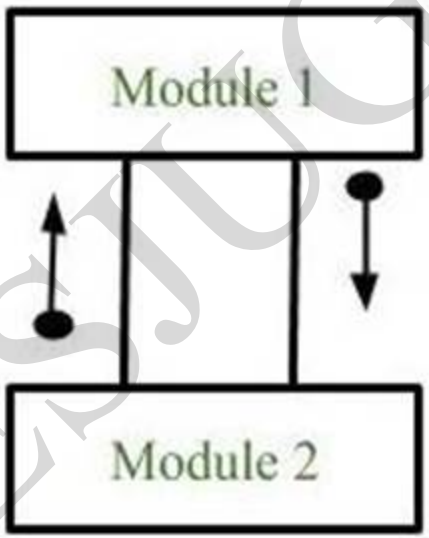
4. **Data Flow**

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



5. **Control Flow**

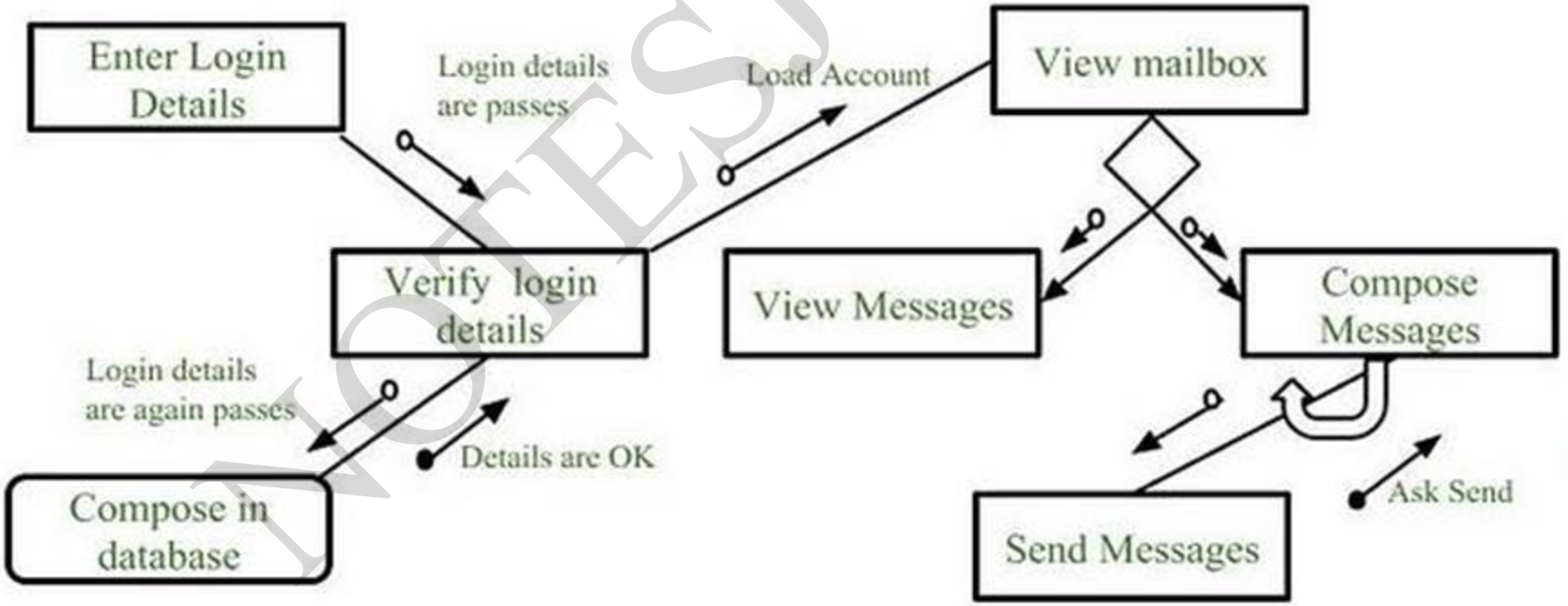
It represents the flow of control between the modules. It is represented by directed arrow with filled circle at the end.



6. **Physical Storage**

Physical Storage is that where all the information are to be stored.

Example : Structure chart for an Email server



Pseudo-Code

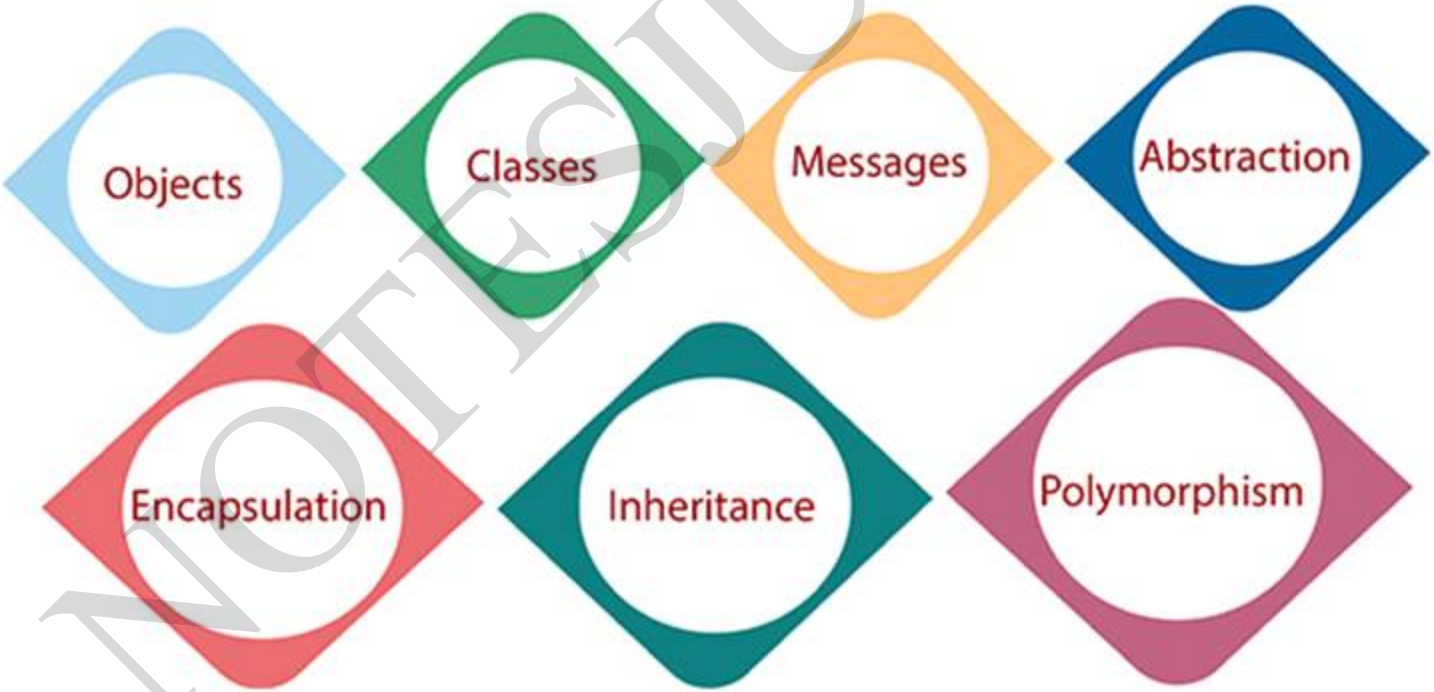
- Pseudo-code notations can be used in both the preliminary and detailed design phases.
- Using pseudo-code, the designer describes system characteristics using *short, concise, English Language phases* that are structured by keywords such as *If-Then-Else, While-Do, and End*.

SOFTWARE DESIGN

Object Oriented Design

- In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities).
- The state is distributed among the objects, and each object handles its state data.

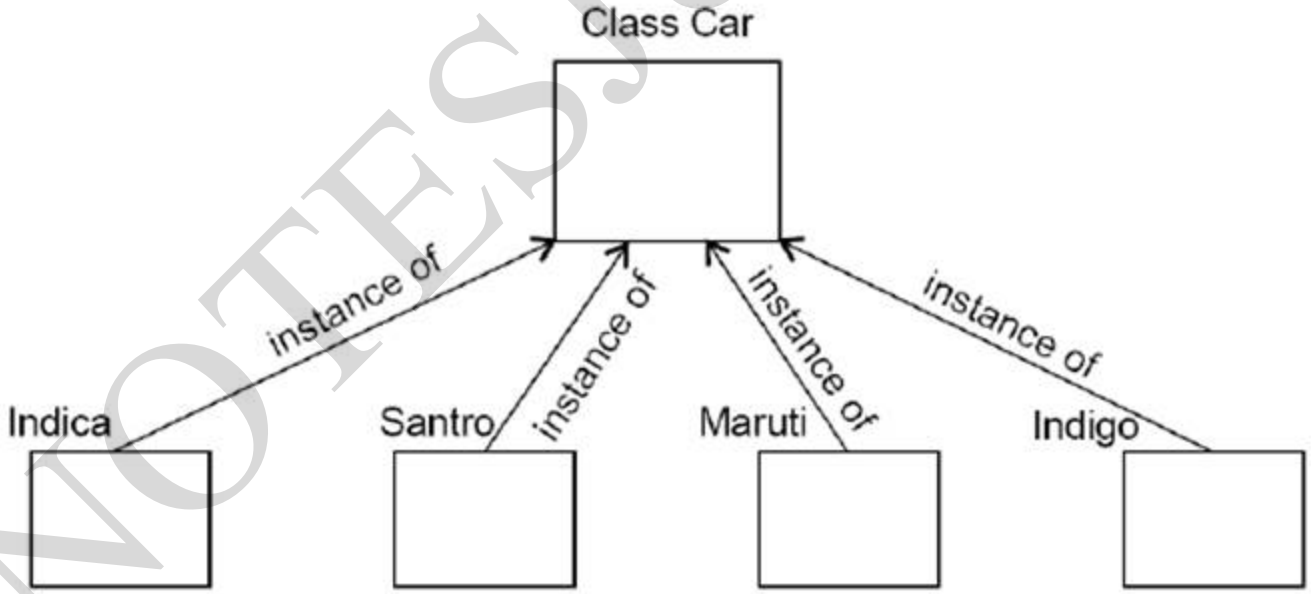
Object Oriented Design



SOFTWARE DESIGN

Object Oriented Design

1. **Objects:** All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.
2. **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 methods, which represents the functionality of the object.



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

SOFTWARE DESIGN

Object Oriented Design

3. **Messages:** Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.

4. **Abstraction** In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.

5. **Encapsulation:** Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.

SOFTWARE DESIGN

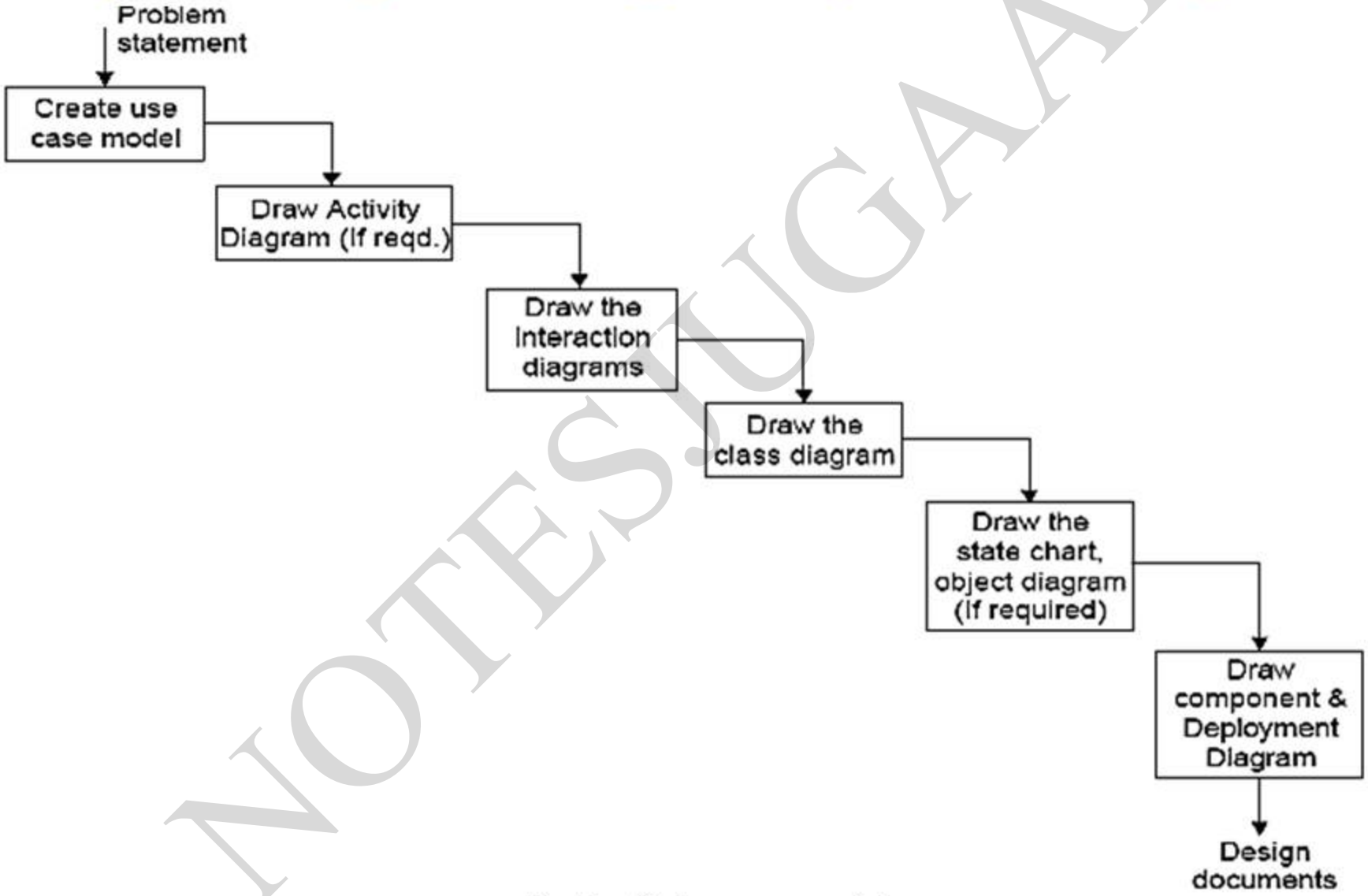
Object Oriented Design

6. **Inheritance:** OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate super classes. This property of OOD is called an inheritance. This makes it easier to define a specific class and to create generalized classes from specific ones.

7. **Polymorphism:** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types. Depending upon how the service is invoked, the respective portion of the code gets executed.

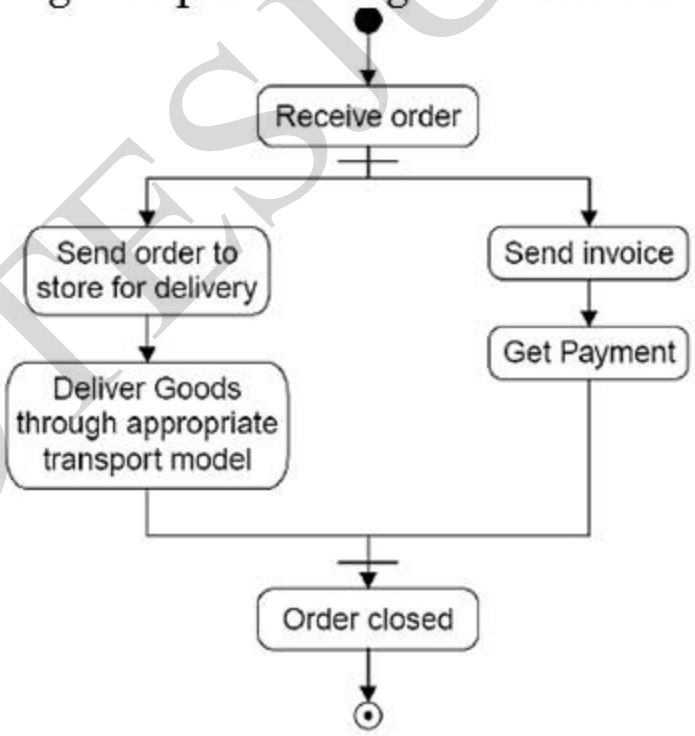
SOFTWARE DESIGN

Steps for Analysis & Design of Object Oriented System



Steps for Analysis & Design of Object Oriented System

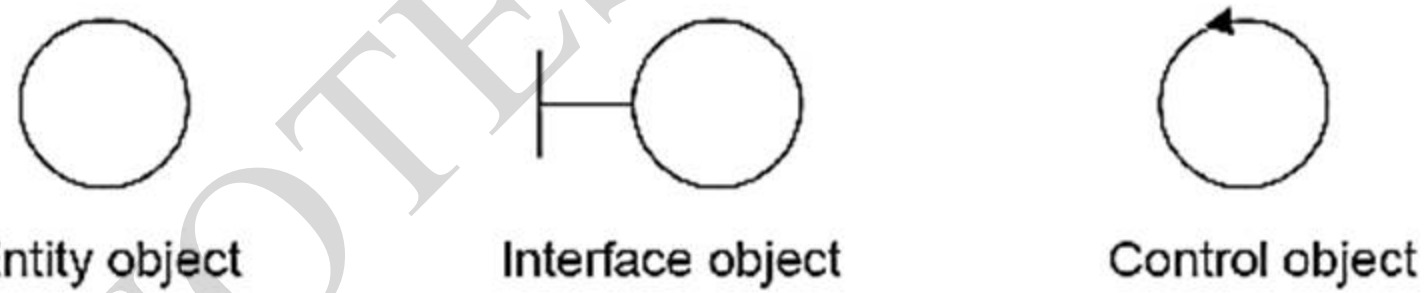
1. **Create use Case Model:** First step is to identify the actors interacting with the system. We should then write the use case and draw the use case diagram.
2. **Draw Activity Diagram (if required):** Activity Diagram illustrate the dynamic nature of a system by modeling the flow of control form activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Fig. below shows the activity diagram processing an order to deliver some goods.



Steps for Analysis & Design of Object Oriented System

3. **Draw the Interaction diagram:** An interaction diagram shows an interaction, consisting of a set of objects and their relationship, including the messages that may be dispatched among them. Interaction diagrams address the dynamic view of a system.
   a) Firstly, we should identify that the objects with respects to every use case.
   b) We draw the sequence diagrams for every use case.
   c) We draw the collaboration diagrams for every use case.

The object types used in this analysis model are entity objects, interface objects and control objects as given in fig. below



jims Greater Noida

Steps for Analysis & Design of Object Oriented System

4. **Draw the Class Diagram:** The class diagram shows the relationship amongst classes. There are four types of relationships in class diagrams.

   a) **Association** are semantic connection between classes. When an association connects two classes, each class can send messages to the other in a sequence or a collaboration diagram. Associations can be bi-directional or unidirectional.

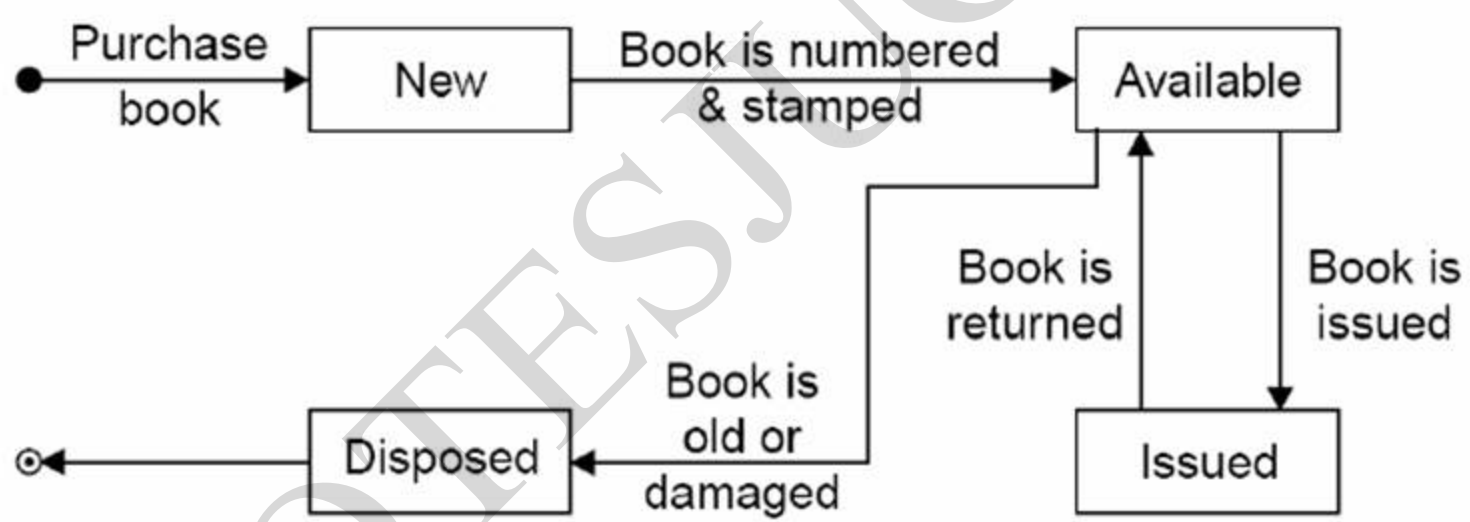
   b) **Dependencies** connect two classes. Dependencies are always unidirectional and show that one class, depends on the definitions in another class.

   c) **Aggregations** are stronger form of association. An aggregation is a relationship between a whole and its parts.

   d) **Generalizations** are used to show an inheritance relationship between two classes.

Steps for Analysis & Design of Object Oriented System

5. **Design of State Chart Diagrams:** A state chart diagram is used to show the state space of a given class, the event that cause a transition from one state to another, and the action that result from a state change.
   A state transition diagram for a "book" in the library system is given in fig. below:



Transition chart for "book" in a library system

Steps for Analysis & Design of Object Oriented System

6. Draw Component and Development Diagram:

Component diagrams address the static implementation view of a system they are related to class diagrams in that a component typically maps to one or more classes, interfaces or collaboration.

SOFTWARE DESIGN

jims Greater Noida

Example

A software has to be developed for automating the manual library of a University. The system should be stand alone in nature. It should be designed to provide functionality's as explained below:

Issue of Books:

- A student of any course should be able to get books issued.
- Books from General Section are issued to all but Book bank books are issued only for their respective courses.
- A limitation is imposed on the number of books a student can issue.
- A maximum of 4 books from Book bank and 3 books from General section is issued for 15 days only.The software takes the current system date as the date of issue and calculates date of return.
- A bar code detector is used to save the student as well as book information.
- The due date for return of the book is stamped on the book.

jims Greater Noida

Example

Return of Books:

- Any person can return the issued books.
- The student information is displayed using the bar code detector.
- The system displays the student details on whose name the books were issued as well as the date of issue and return of the book.
- The system operator verifies the duration for the issue.
- The information is saved and the corresponding updating take place in the database.

SOFTWARE DESIGN

jims Greater Noida

Example

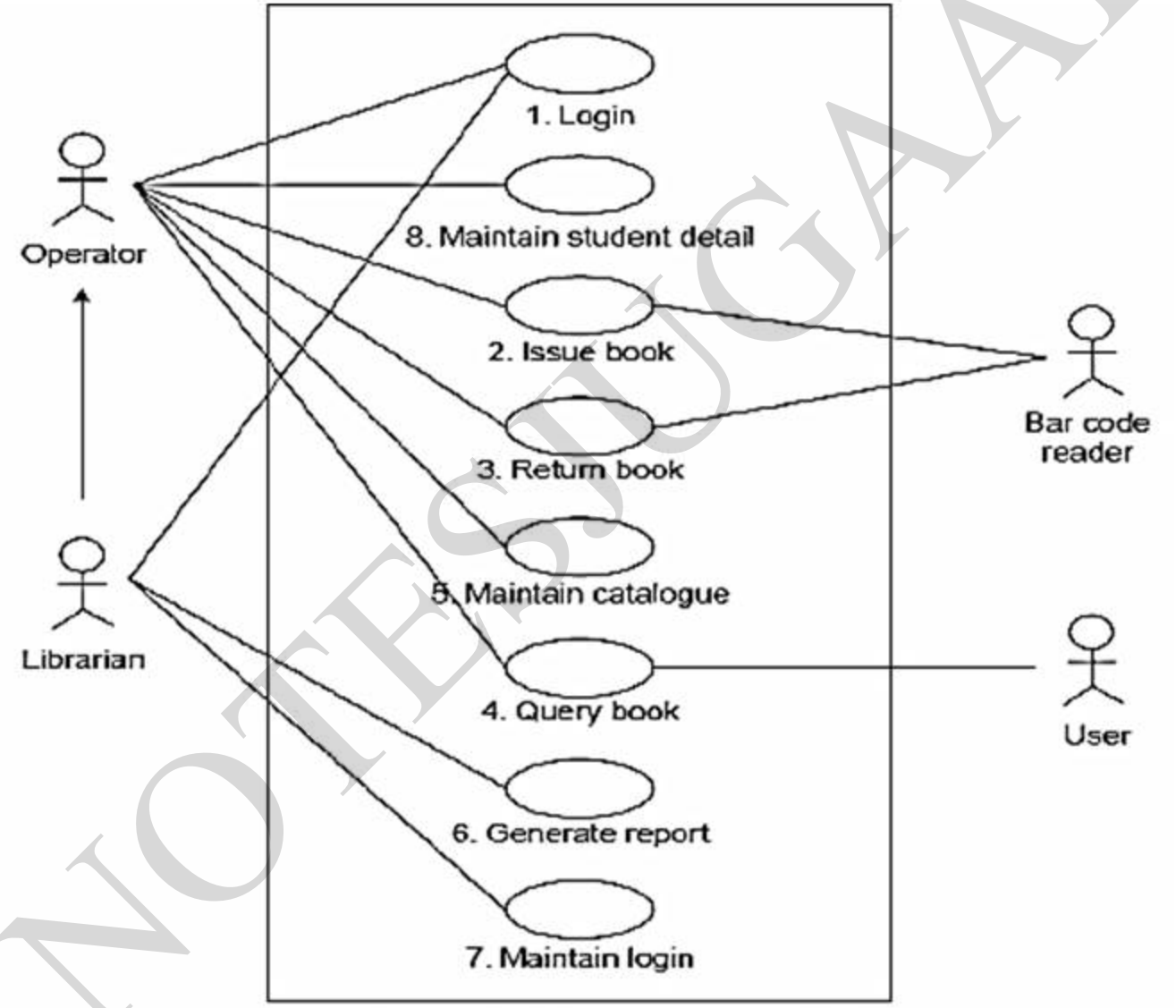
Query Processing:

- The system should be able to provide information like:
- Availability of a particular book.
- Availability of book of any particular author.
- Number of copies available of the desired book.

The system should also be able to generate reports regarding the details of the books available in the library at any given time. The corresponding printouts for each entry (issue/return) made in the system should be generated. Security provisions like the 'login authenticity should be provided. Each user should have a user id and a password. Record of the users of the system should be kept in the log file. Provision should be made for full backup of the system.

SOFTWARE DESIGN

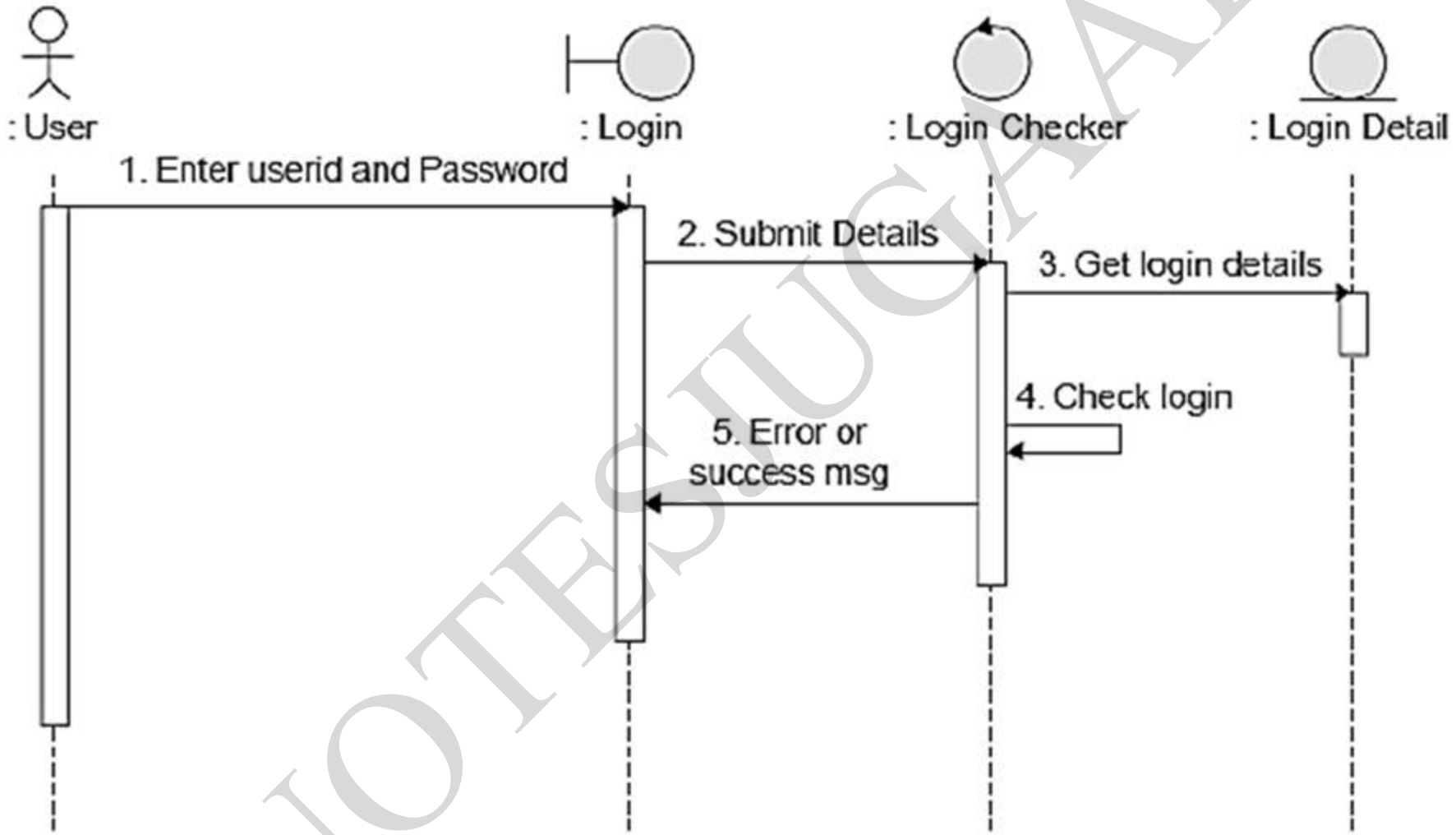
Solution



Use case diagram for library management system

SOFTWARE DESIGN

Solution

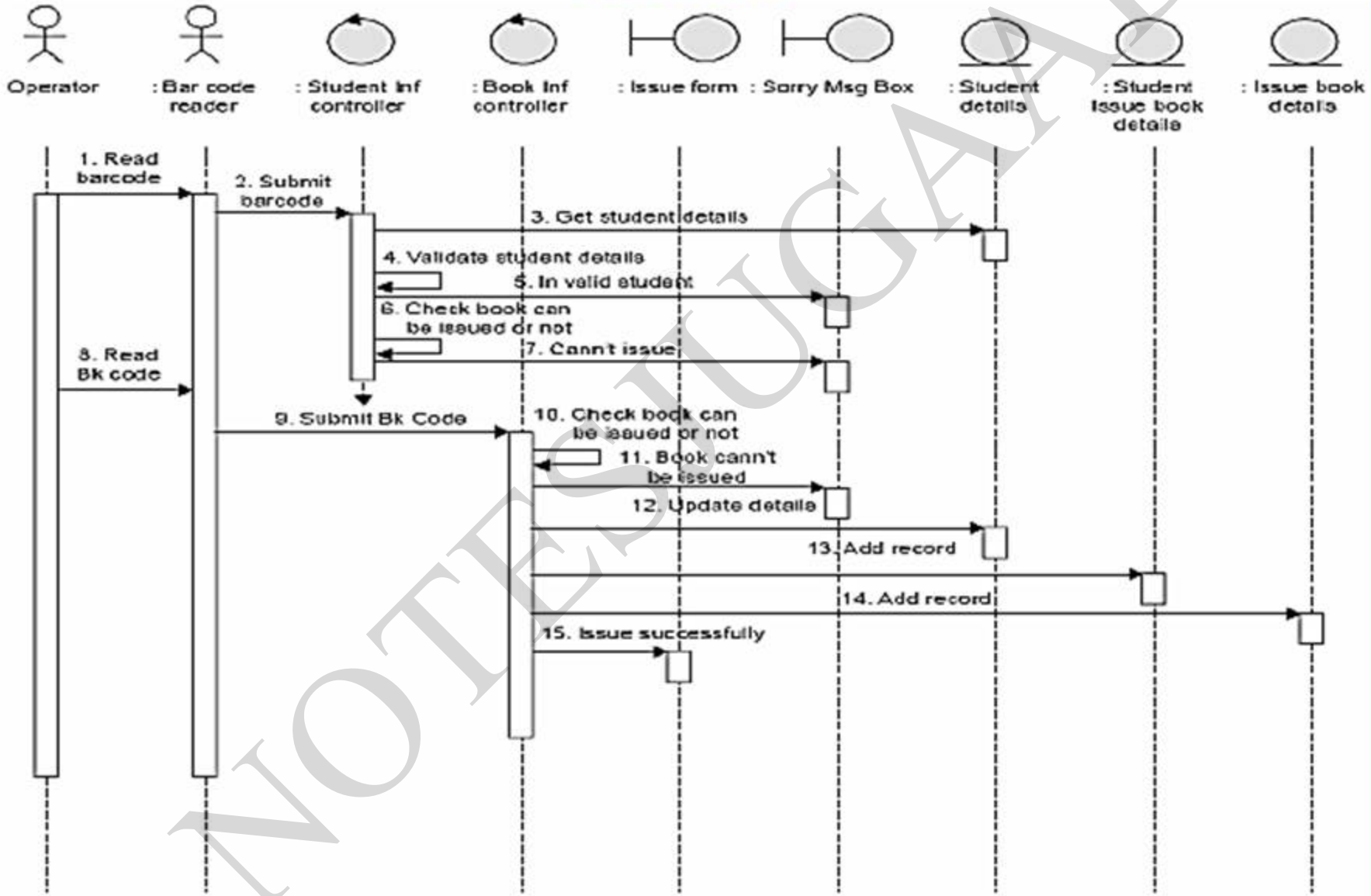


Sequence diagram—Login

SOFTWARE DESIGN

jims Greater Noida

Solution

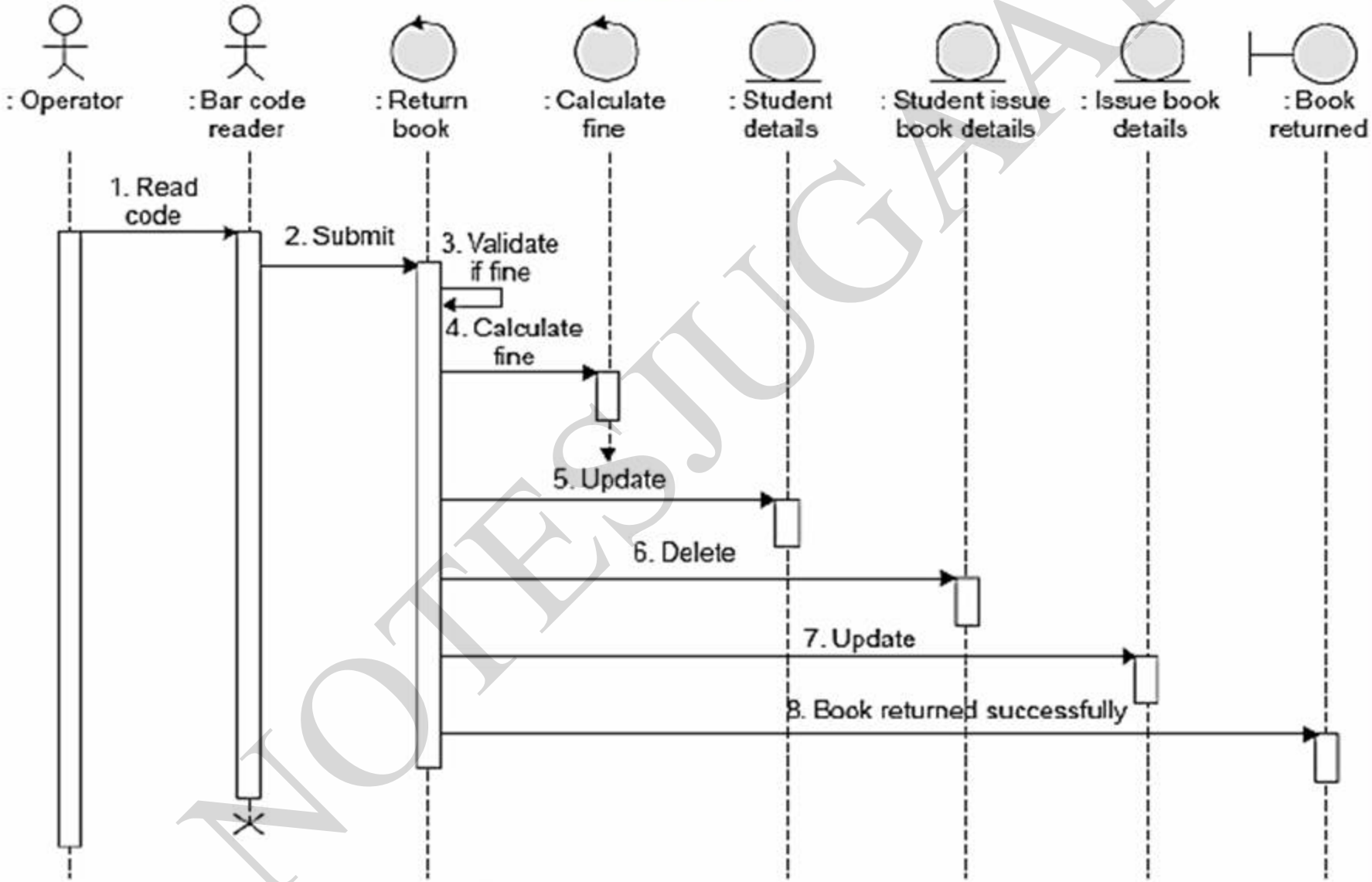


Sequence diagram—issue book

SOFTWARE DESIGN

jims Greater Noida

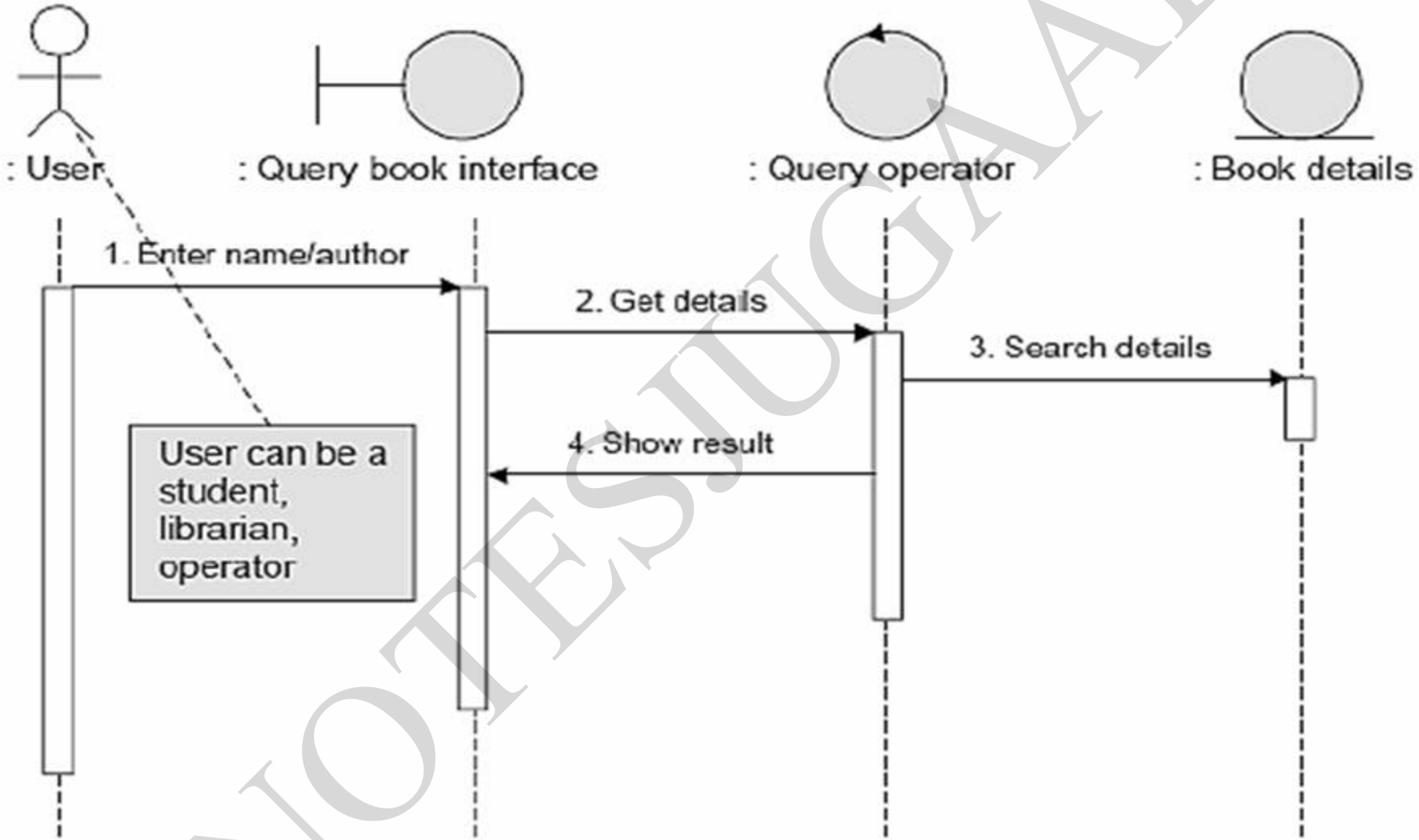
Solution



Sequence diagram—return book

SOFTWARE DESIGN

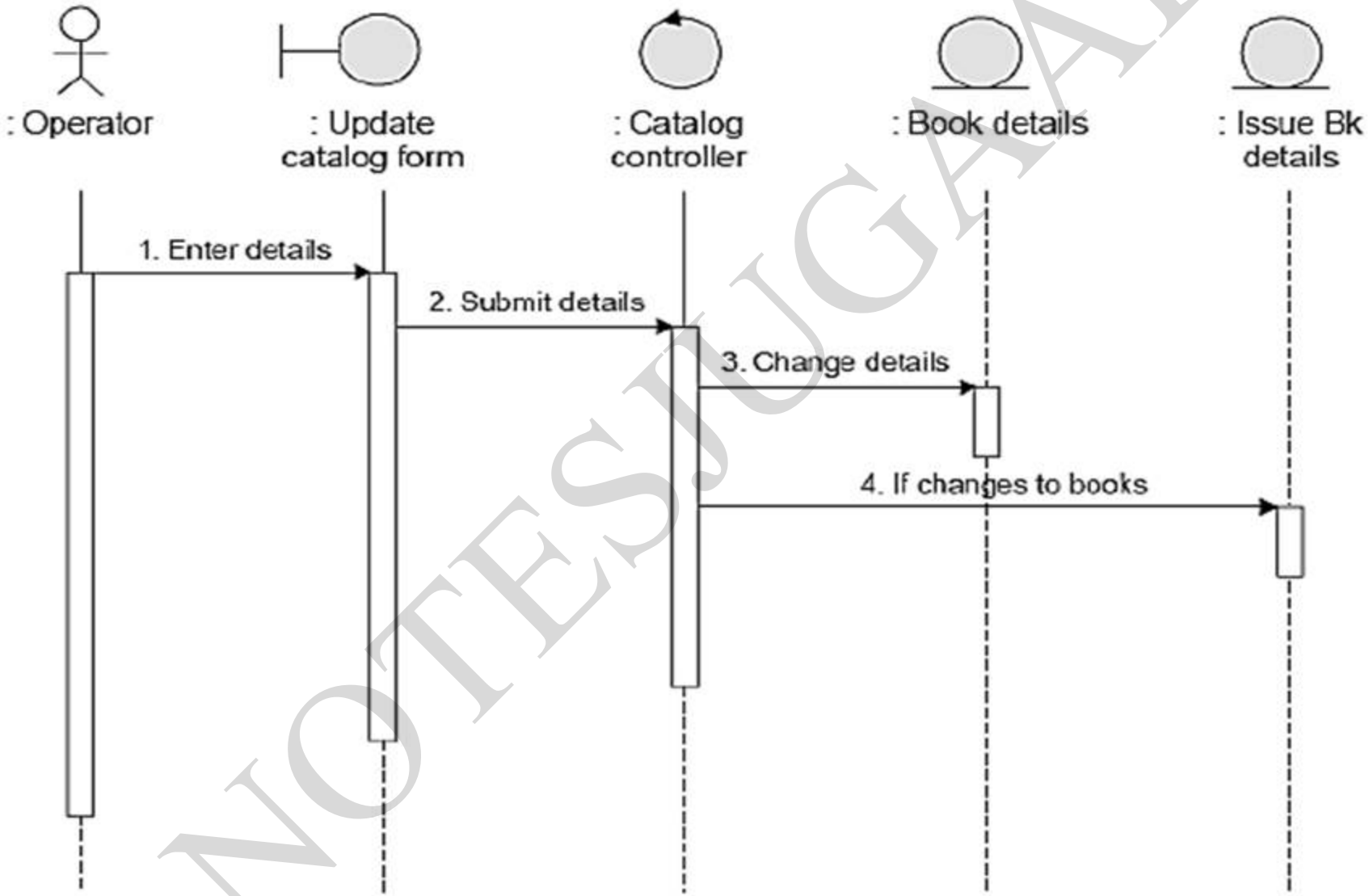
Solution



Sequence diagram—query book

SOFTWARE DESIGN

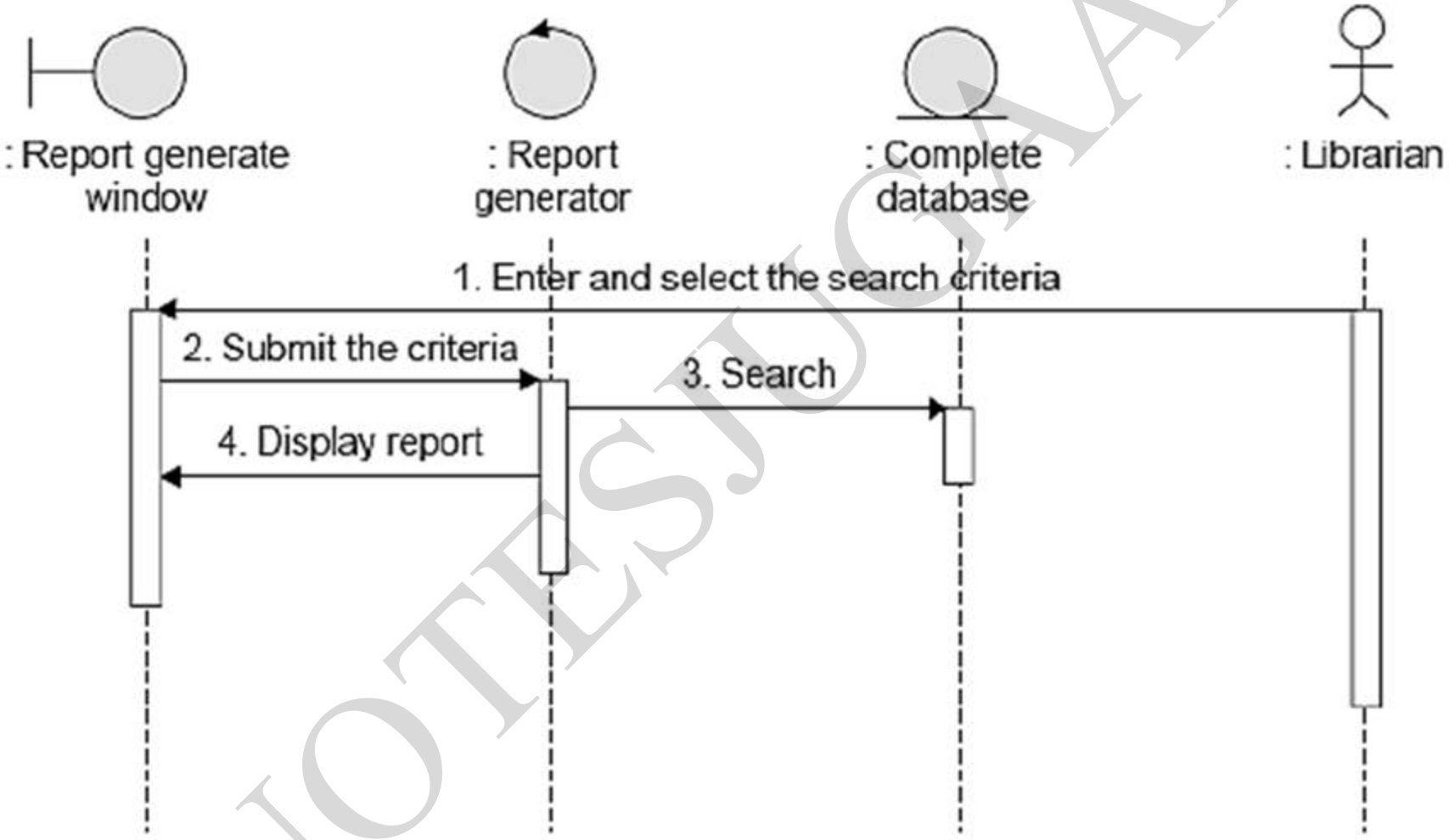
Solution



Sequence diagram—maintain catalog

SOFTWARE DESIGN

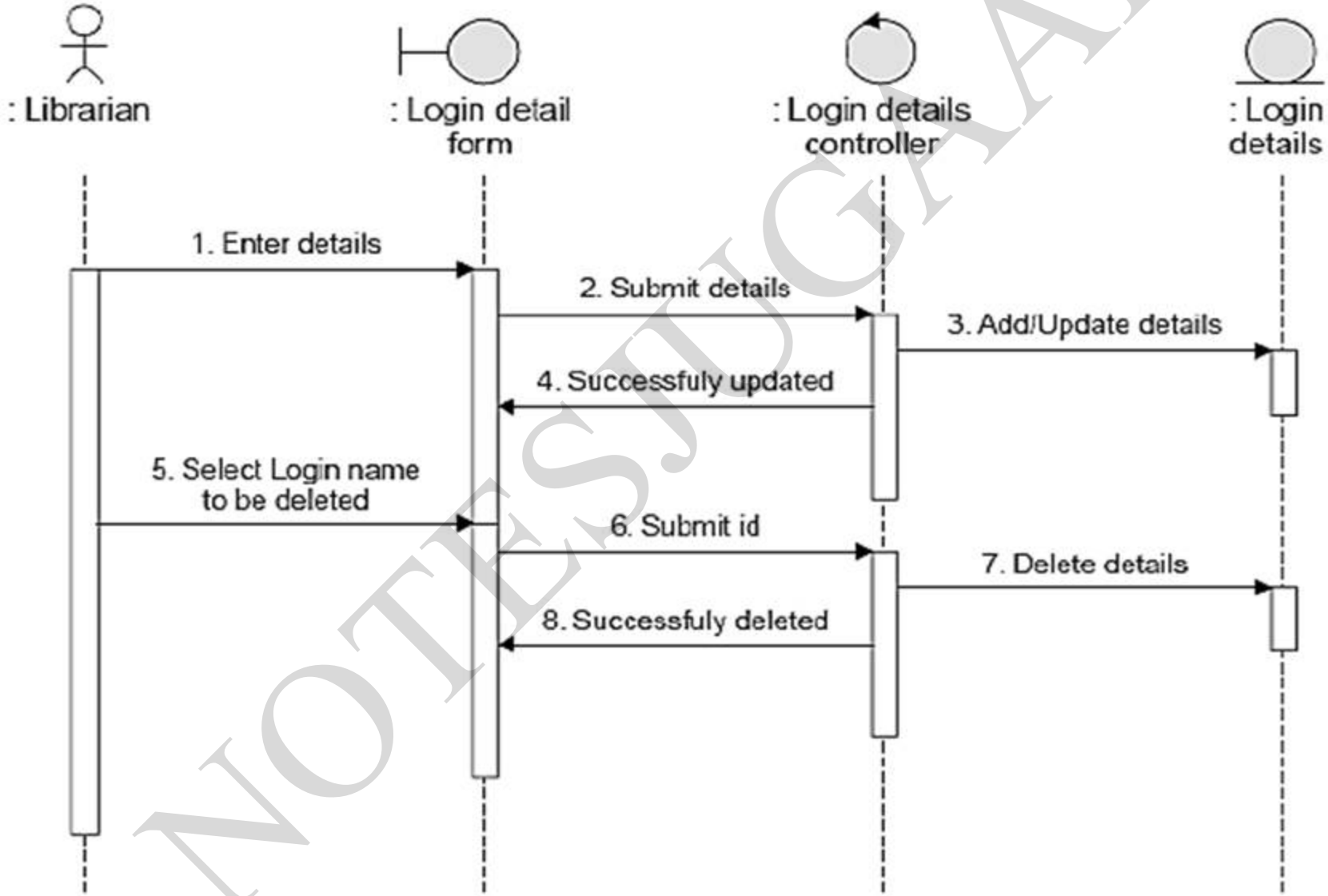
Solution



Sequence diagram—generate reports

SOFTWARE DESIGN

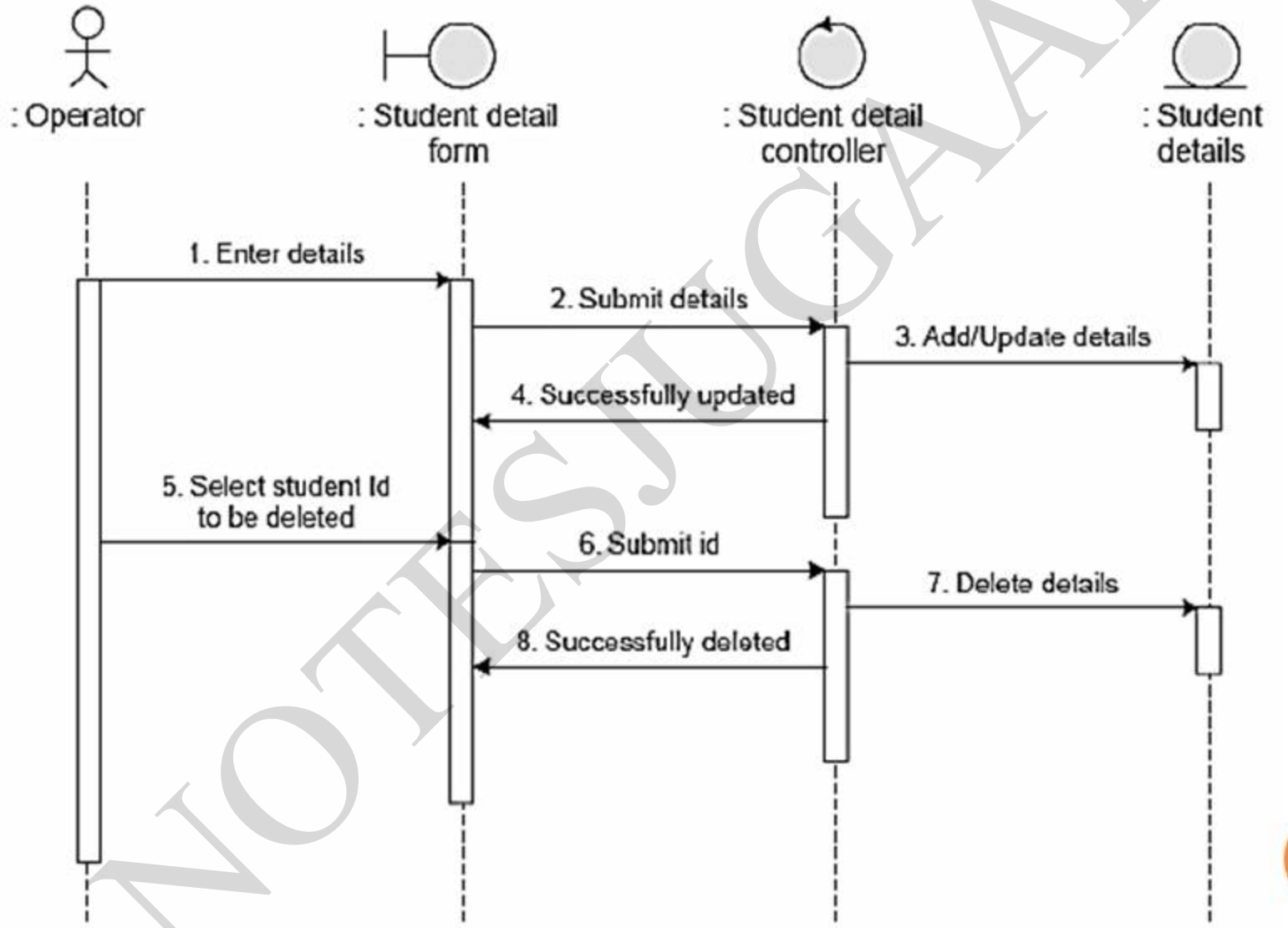
Solution



Sequence diagram—maintain login

SOFTWARE DESIGN

Solution

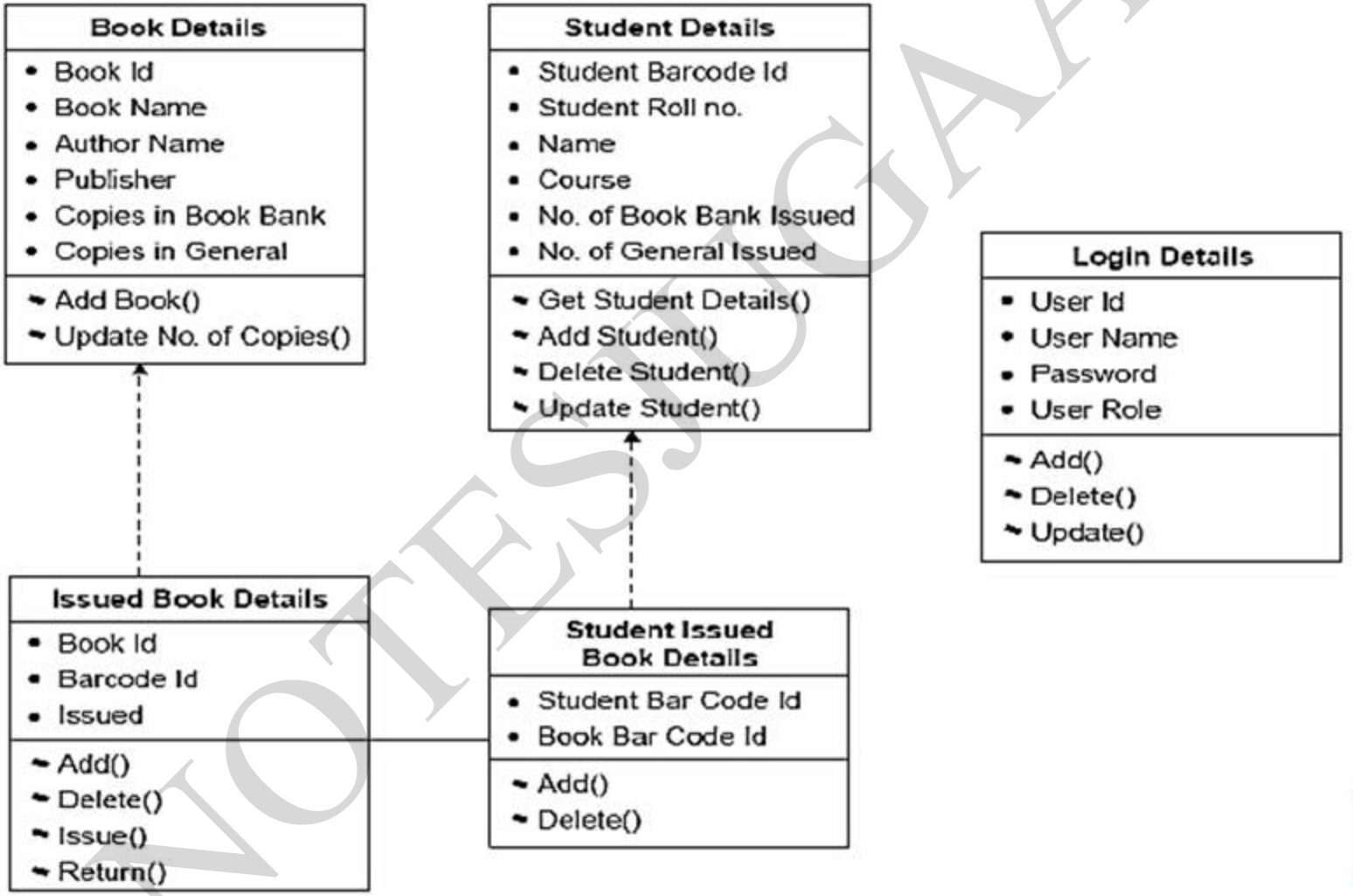


Sequence diagram—maintain student details

SOFTWARE DESIGN

Solution

Class diagram of entity classes



Class diagram of entity classes

REFERENCE

Book:

1. Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh, Copyright © New Age International Publishers, 2007
2. An Integrated Approach to Software Engineering by Pankaj Jalote (IIT Kanpur), © 2005 by Springer Science-i-Business Media, Inc.

Website

- https://www.javatpoint.com/software-engineering-software-design
- https://www.tutorialspoint.com/software_engineering/software_design_basics.htm
- https://www.adoxx.org/live/external-coupling-overview
- https://www.geeksforgeeks.org/cohesion-in-java/
- https://www.geeksforgeeks.org/software-engineering-structure-charts/
- https://www.javatpoint.com/software-engineering-coupling-and-cohesion
- http://girfahelp.blogspot.com/2016/05/difference-between-coupling-and-cohesion.html

SOFTWARE ENGINEERING

Unit 3: Software Metrics

Shree Harsh Attri
(Assistant Professor)
Department: Bachelor of Computer Applications

JIMS Engineering Management Technical Campus
Greater Noida (U.P)

Software Metrics

jims Greater Noida

What & Why

1. How to measure the size of a software?
2. How much will it cost to develop a software?
3. How many bugs can we expect?
4. When can we stop testing?
5. When can we release the software?
6. What is the complexity of a module?
7. What is the module strength and coupling?
8. What is the reliability at the time of release?
9. Which test technique is more effective?
10. Are we testing hard or are we testing smart?
11. Do we have a strong program or a week test suite?

SOFTWARE MEASUREMENT

Measurement: A measurement is an indicator of the ***size, quantity, amount or dimension*** of a particular attributes of a product or process. It is the act of determine a measure.

Software Measurement: is concerned with deriving a numeric value for an attribute of a software product or process.

Need of Software Measurement:

Software is measured to:

1. Create the quality of the current product or process.
2. Anticipate future qualities of the product or process.
3. Enhance the quality of a product or process.
4. Regulate the state of the project in relation to budget and schedule.

SOFTWARE MEASUREMENT

jims Greater Noida

Classification of Software Measurement

There are 2 types of software measurement:

1. **Direct Measurement:** In direct measurement the product, process or thing is measured directly using standard scale. *For example: Size, complexity, dependency among modules.*

2. **Indirect Measurement:** In indirect measurement the quantity or quality to be measured is measured using related parameter i.e. by use of reference. *For example: The total number of failures experienced by a user, the length of time it takes to search the database and retrieve information.*

SOFTWARE METRICS

A software metric is a ***measure of software characteristics which are measurable or countable***.

Software metrics are valuable for many reasons, ***including measuring software performance, planning work items, measuring productivity, and many other uses.***

Software metrics are similar to the four functions of management: **Planning, Organization, Control, or Improvement**.

Objectives of Software Metrics are

1. Measuring the size of the software quantitatively.
2. Assessing the level of complexity involved.
3. Assessing the strength of the module by measuring coupling.
4. Assessing the testing techniques.
5. Specifying when to stop testing.
6. Determining the date of release of the software.
7. Estimating cost of resources and project schedule.

SOFTWARE METRICS

Classification of Software Metrics

1. **Product Metrics:** These are the measures of various characteristics of the software product. The two important software characteristics are:

   a) Size and complexity of software.

   b) Quality and reliability of software.

   These metrics can be computed for different stages of SDLC.

2. **Process Metrics:** These are the measures of various characteristics of the software development process. It describes the effectiveness and quality of the processes that produce the software product for example,

   - effort required in the process
   - time to produce the product
   - effectiveness of defect removal during development
   - number of defects found during testing
   - maturity of the process

jims Greater Noida

Classification of Software Metrics

3. **Project Metrics:** Project metrics are the metrics used by the project manager to check the project's progress as follows:
   - Data from the past projects are used to collect various metrics, like time and cost; these estimates are used as a base of new software.
   - The project manager will check its progress from time-to-time and will compare the effort, cost, and time with the original effort, cost and time.
   - This metrics is used to decrease the development costs, time efforts and risks.
   - As quality improves, the number of errors and time, as well as cost required, is also reduced.

SOFTWARE METRICS: TOKEN COUNT

Halstead's Software Metrics

A computer program is an implementation of an algorithm considered to be a collection of **Tokens** which can be classified as either ***operators or operands.***

Token Count

All software science metrics can be defined in terms of symbols. These symbols are called as a **Token**.

The following base measures can be collected :

n1 = count of unique operators.
n2 = count of unique operands.
N1 = count of total occurrences of operators.
N2 = count of total occurrence of operands.

The length of the program in the terms of the total number of tokens used is

N = N1+N2

N : program length

N1 : total occurrences of operators

N2 : total occurrences of operands

SOFTWARE METRICS: TOKEN COUNT

Halstead Vocabulary – The total number of unique operator and unique operand occurrences.

$$n = n1 + n2$$

Program Volume (V)

The unit of measurement of volume is the standard unit for size "bits." It is the actual size of a program if a uniform binary encoding for the vocabulary is used.

$$V = N * \log_2 (n)$$

Program Level (L)

The value of L ranges between zero and one, with L = 1 representing a program written at the highest possible level (i.e., with minimum size).

$$L = V^* / V$$

Potential Minimum Volume

The potential minimum volume **V*** is defined as the volume of the most short program in which a problem can be coded.

$$V^* = (2 + n2^*) * \log2 (2 + n2^*)$$

Here, **n2*** is the count of unique input and output parameters

SOFTWARE METRICS: TOKEN COUNT

Program Difficulty

The difficulty level or error-proneness (D) of the program is as the volume of an implementation of a program increases, the program level decreases and the difficulty increases. Thus, programming practices such as redundant usage of operands, or the failure to use higher-level control constructs will tend to increase the volume as well as the difficulty.

$$D = (n1/2) * (N2/n2) = 1 / L$$

Programming Effort (E)

The unit of measurement of E is elementary mental discriminations.

$$E = V / L = D * V$$

Programming Time (T)

Shows time (in minutes) needed to translate the existing algorithm into implementation in the specified program language.

$$T = E / \beta$$

where **β** is normally set to **18** since this seemed to give best results in Halstead's earliest experiments, which compared the predicted times with observed programming times, including the time for design, coding, and testing.

SOFTWARE METRICS: TOKEN COUNT

Estimated Program Length

According to Halstead, The first Hypothesis of software science is that the length of a well-structured program is a function only of the number of unique operators and operands.

$$N=N1+N2$$

And estimated program length is denoted by $\hat{N}$

$$\hat{N} = n1 \log_2 n1 + n2 \log_2 n2$$

The following alternate expressions have been published to estimate program length:

$N_J = \log_2 (n1!) + \log_2 (n2!)$

$N_B = n1 * \log_2 n2 + n2 * \log 2n1$

$N_C = n1 * \text{sqrt}(n1) + n2 * \text{sqrt}(n2)$

$N_S = (n * \log_2 n) / 2$

Estimated Program Level

$$\hat{L} = \frac{2}{n1} X \frac{n2}{N2}$$

SOFTWARE METRICS: TOKEN COUNT

Language Level - Shows the algorithm implementation program language level.

$$\lambda = L \times V^* = L^2 V$$

Using this formula, Halstead and other researchers determined the language level for various languages

Language	Language level λ	Variance σ
PL/1	1.53	0.92
ALGOL	1.21	0.74
FORTRAN	1.14	0.81
CDC Assembly	0.88	0.42
PASCAL	2.54	-
APL	2.42	-
C	0.857	0.445

Language levels

SOFTWARE METRICS: TOKEN COUNT

Metric	Meaning	Mathematical Representation
n	Vocabulary	n1 + n2
N	Size or Length	N1 + N2
V	Volume	Length * log2 Vocabulary
V*	Potential Minimum Volume	$(2 + n2^*) * \log_2 (2 + n2^*)$
L	Program Level	V* / V
D	Difficulty	(n1/2) * (N1/n2) = 1 / L
E	Efforts	Difficulty * Volume
B	Errors	Volume / 3000
T	Testing time	Time = Efforts / S, where S=18 seconds.
λ	Language Level	

Summary of Halstead's Software Metrics

SOFTWARE METRICS: TOKEN COUNT

Counting rules for C language

1. Comments are not considered.
2. The identifier and function declarations are not considered.
3. All the variables and constants are considered operands.
4. Global variables used in different modules of the same program are counted as multiple occurrences of the same variable.
5. Local variables with the same name in different functions are counted as unique operands.
6. Functions calls are considered as operators.
7. All looping statements e.g., do {…} while ( ), while ( ) {…}, for ( ) {…}, all control statements e.g., if ( ) {…}, if ( ) {…} else {…}, etc. are considered as operators.
8. In control construct switch ( ) {case:…}, switch as well as all the case statements are considered as operators.
9. The reserve words like return, default, continue, break, sizeof, etc., are considered as operators.
10. All the brackets, commas, and terminators are considered as operators.

SOFTWARE METRICS: TOKEN COUNT

Counting rules for C language

11. GOTO is counted as an operator and the label is counted as an operand.
12. The unary and binary occurrence of "+" and "-" are dealt separately. Similarly "*" (multiplication operator) are dealt separately.
13. In the array variables such as "array-name [index]" "array-name" and "index" are considered as operands and [ ] is considered as operator.
14. 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.
15. All the hash directive are ignored.

SOFTWARE METRICS: TOKEN COUNT

Example: 1

Consider the sorting program. List out the **operators** and **operands** and also calculate the values of software science measures like **n, N, V, E, λ** etc.

```
int sort (int x[ ], int n)

{
    int i, j, save, im1;
    /*This function sorts array x in ascending order */
    If (n< 2) return 1;
    for (i=2; i< =n; i++)
    {
        im1=i-1;
        for (j=1; j< =im1; j++)
            if (x[i] < x[j])
            {
                Save = x[i];
                x[i] = x[j];
                x[j] = save;
            }
    }
    return 0;
}
```

SOFTWARE METRICS: TOKEN COUNT

The list of operators and operands is given in the table

Operators	Occurrences	Operands	Occurrences
int	4	SORT	1
()	5	x	7
,	4	n	3
[]	7	i	8
if	2	j	7
<	2	save	3
;	11	im1	3
for	2	2	2
=	6	1	3
-	1	0	1
<=	2	-	-
++	2	-	-
return	2	-	-
{}	3	-	-
n1=14	**N1=53**	**n2=10**	**N2=38**

SOFTWARE METRICS: TOKEN COUNT

Here N1=53 and N2=38.

The program length **N**=N1+N2=53+38=91

Vocabulary of the program n=n1+n2=14+10=24

Volume **V**= N * $\log_2$ (n) = 91 x $\log_2$ 24 = 417 bits.

The estimate program length $\hat{N}$ of the program

$$= 14 \log_2 14 + 10 \log_2 10$$
$$= 14 * 3.81 + 10 * 3.32$$
$$\hat{N} = 53.34 + 33.2 = 86.45$$

Conceptually unique input and output parameters are represented by n2*.

$$n2^* = 3$$

{x: array holding the integer to be sorted. This is used as both input and output}

{N: the size of the array to be sorted}

The Potential Volume V* = 5 $\log_2$5 = 11.6

SOFTWARE METRICS: TOKEN COUNT

Since L = V* / V

$$= \frac{11.6}{417} = 0.027$$

$$D = I/L$$

$$= \frac{1}{0.027} = 37.03$$

Estimated Program Level

$$\hat{L} = \frac{2}{n1} X \frac{n2}{N2} = \frac{2}{14} X \frac{10}{38} = 0.038$$

SOFTWARE METRICS: TOKEN COUNT

Programming Effort (E)

$$\hat{E} = V / \hat{L} = \hat{D} \times V$$

$$= 417 / 0.038 = 10973.68$$

Therefore, 10974 elementary mental discrimination are required to construct the program.

$$T = E / \beta = \frac{10974}{18} = 610 \text{ seconds} = 10 \text{ minutes}$$

This is probably a reasonable time to produce the program, which is very simple

End of Example: 1

DATA STRUCTURE METRICS

The need for software development and other activities are to process data. Some data is input to a system, program or module; some data may be used internally, and some data is the output from a system, program, or A **count** of this data structure is called ***Data Structured Metrics.***

Program	*Data Input*	*Internal Data*	*Data Output*
Payroll	Name/ Social Security No./ Pay Rate/ Number of hours worked	Withholding rates Overtime factors Insurance premium Rates	Gross pay withholding Net pay Pay ledgers
Spreadsheet	Item Names/ Item amounts/ Relationships among items	Cell computations Sub-totals	Spreadsheet of items and totals
Software Planner	Program size/ No. of software developers on team	Model parameters Constants Coefficients	Est. project effort Est. project duration

Examples of Input, Internal, and Output data

DATA STRUCTURE METRICS

There are some Data Structure metrics to compute the **Effort** and **Time** required to complete the project.

The metrics are as follows:

1. The Amount of Data
2. The Usage of data within a Module
3. Program Weakness
4. The sharing of Data among Modules

DATA STRUCTURE METRICS

jims Greater Noida

1. The Amount of Data

To measure the amount of Data, there are further many different metrics, and these are:

- **Number of variable (VARS):** In this metric, the Number of variables used in the program is counted.
- **Number of Operands (η_2):** In this metric, the Number of operands used in the program is counted.

$$\eta_2 = \text{VARS} + \text{Constants} + \text{Labels}$$

- **Total number of occurrence of the variable (N2):** In this metric, the total number of occurrence of the variables are computed

DATA STRUCTURE METRICS

jims Greater Noida

2. The Usage of data within a Module

To measure this metric, the average numbers of **Live Variables** are computed.

Live Variable

Definitions :

1. A variable is live from the beginning of a procedure to the end of the procedure.
2. A variable is live at a particular statement only if it is referenced a certain number of statements before or after that statement.
3. A variable is live from its first to its last references within a procedure.

$$\text{Average no of Live variables (LV)} = \frac{\text{Sum of count live variables}}{\text{Sum of count of executable statements}}$$

2. The Usage of data within a Module

Variable Spans

A Metric that captures some of the essence of how often a variable is used in a program is called span. This metric is the number of statements between two successive references of the same variables.

```
...
21          scanf ("%d %d, &a, &b)
...
32          x =a;
...
45          y = a – b;
...
53          z = a;
...
60          printf ("%d %d, a, b);
...
```

Statements in program referring to variables a and b.

jims Greater Noida

3. Program Weakness

Program weakness depends on its Modules weakness. If Modules are weak(less Cohesive), then it increases the effort and time metrics required to complete the project.

$$\text{Average life of variables } (\gamma) = \frac{\text{Sum of count live variables}}{\text{Sum of count of executable statements}}$$

Module weakness (*WM*) has been defined as:

$$WM = \overline{LV} * \gamma$$

where:

$\overline{LV}$ = average number of live variables

γ = average life of variables

DATA STRUCTURE METRICS

A program is normally a combination of various modules, hence program weakness can be a useful measure and is defined as:

$$WP = \frac{\left(\sum_{i=1}^{m} WM_i \right)}{m}$$

where, WM_i : weakness of *i*th module

WP : weakness of the program

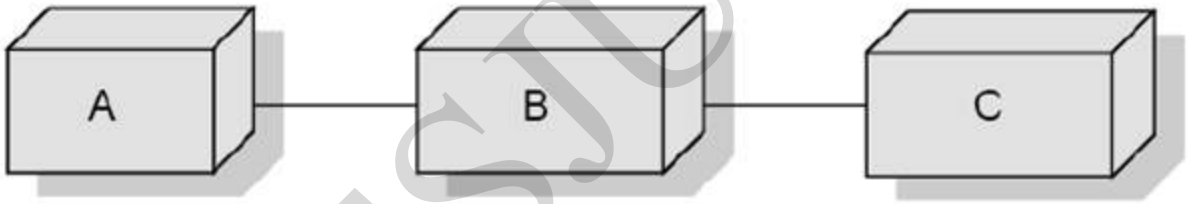
m : number of modules in the program

DATA STRUCTURE METRICS

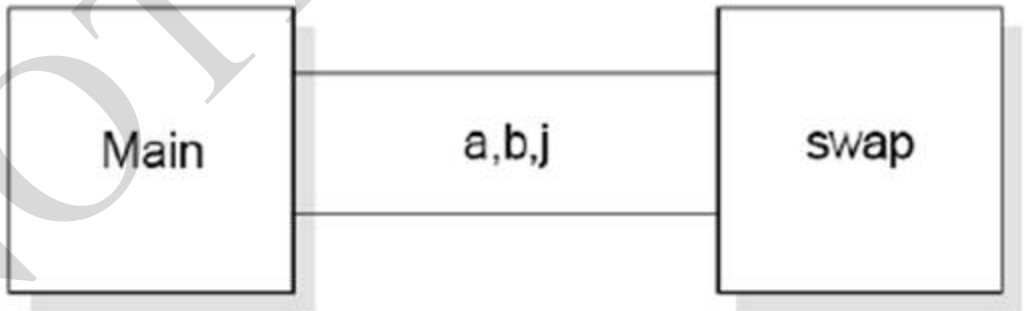
4. The sharing of Data among Modules

As the data sharing between the Modules increases (higher Coupling), no parameter passing between Modules also increased.

As a result, more effort and time are required to complete the project. So Sharing Data among Module is an important metrics to calculate effort and time.



Three modules from an imaginary program



The data shared in program bubble

REFERENCE

Book:

1. Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh, Copyright © New Age International Publishers, 2007
2. An Integrated Approach to Software Engineering by Pankaj Jalote (IIT Kanpur), © 2005 by Springer Science-i-Business Media, Inc.

Website

- https://www.geeksforgeeks.org/software-measurement-and-metrics/
- https://www.javatpoint.com/software-engineering-software-metrics
- http://ecomputernotes.com/software-engineering/software-metrics
- https://www.tutorialspoint.com/software_engineering/software_design_complexity.htm
- https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/
- https://www.javatpoint.com/software-engineering-halsteads-software-metrics
- https://www.javatpoint.com/software-engineering-data-structure-metrics