Software Engineering (MCA-122)
MCA 3rd Sem (2020-21)

D. S. Singh
Associate Professor
Department of ITCA
MMMUT Gorakhpur

Software Design

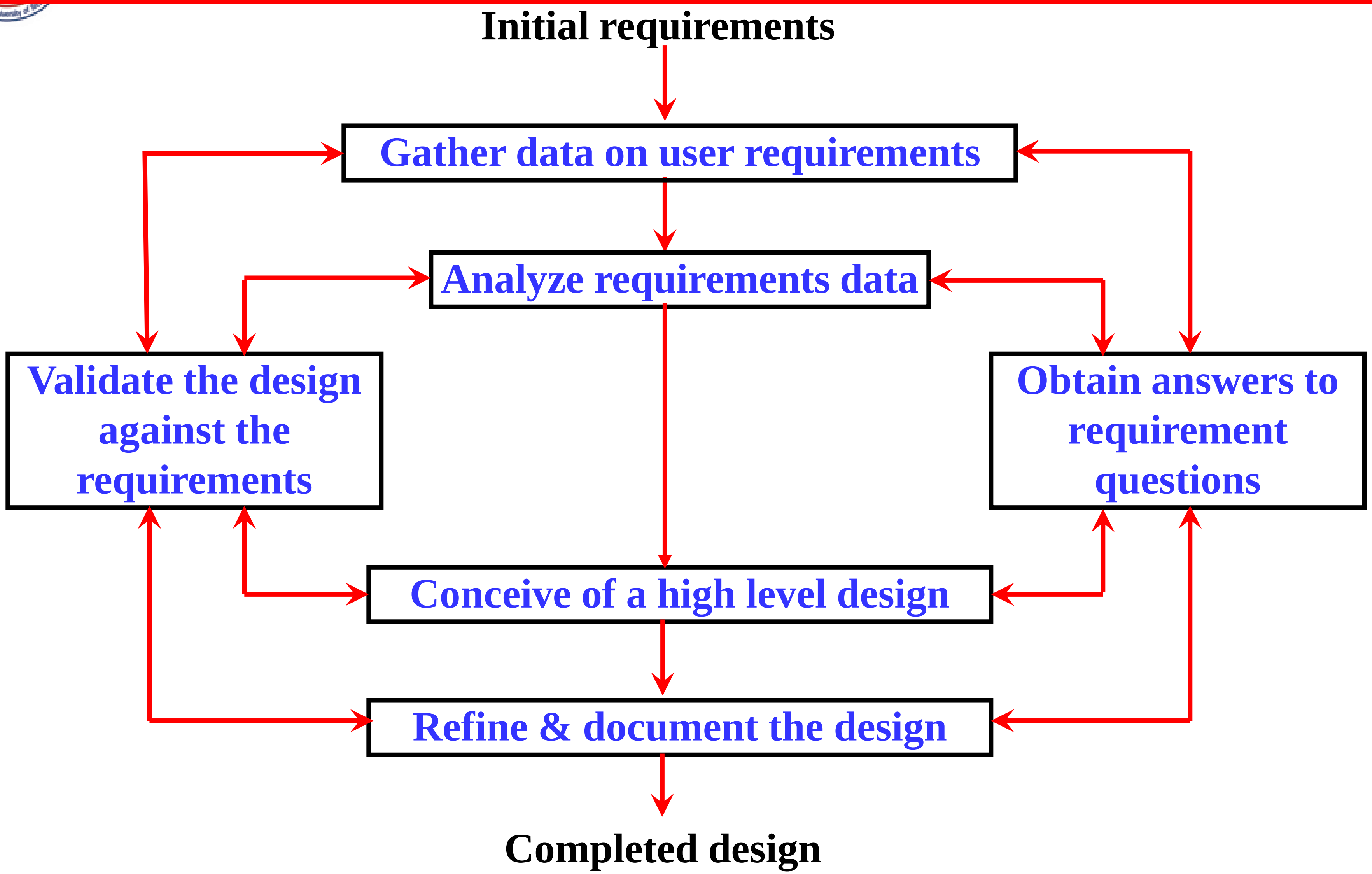
- **More creative than analysis**
- **Problem solving activity**

WHAT IS DESIGN

‘HOW’

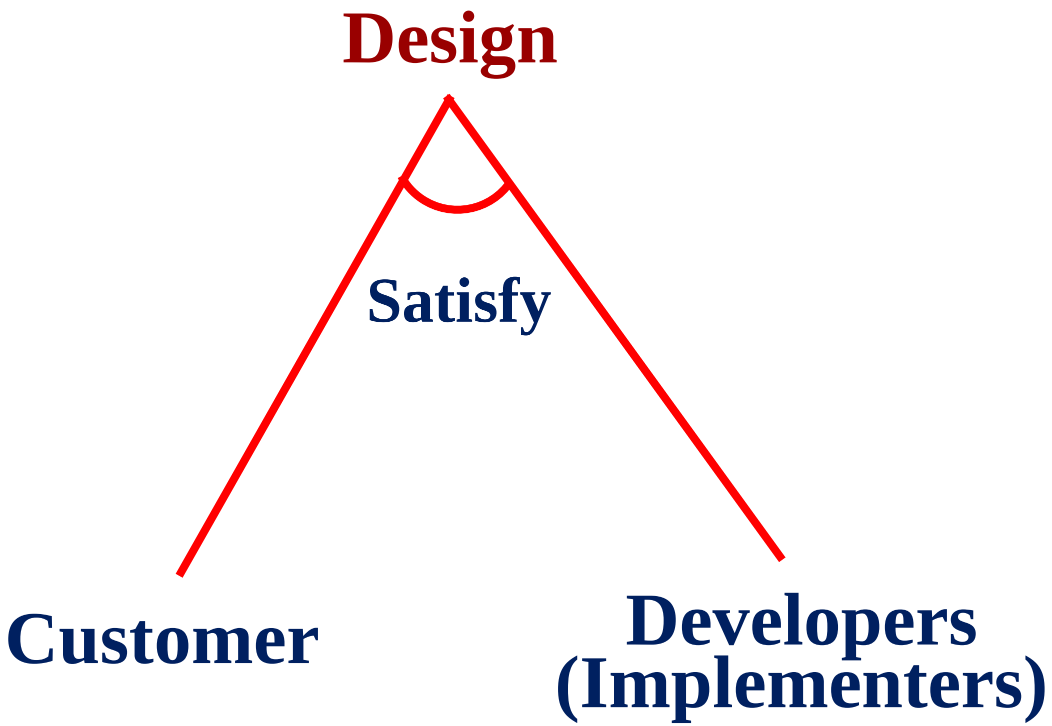
→ **Software design document (SDD)**

Software Design



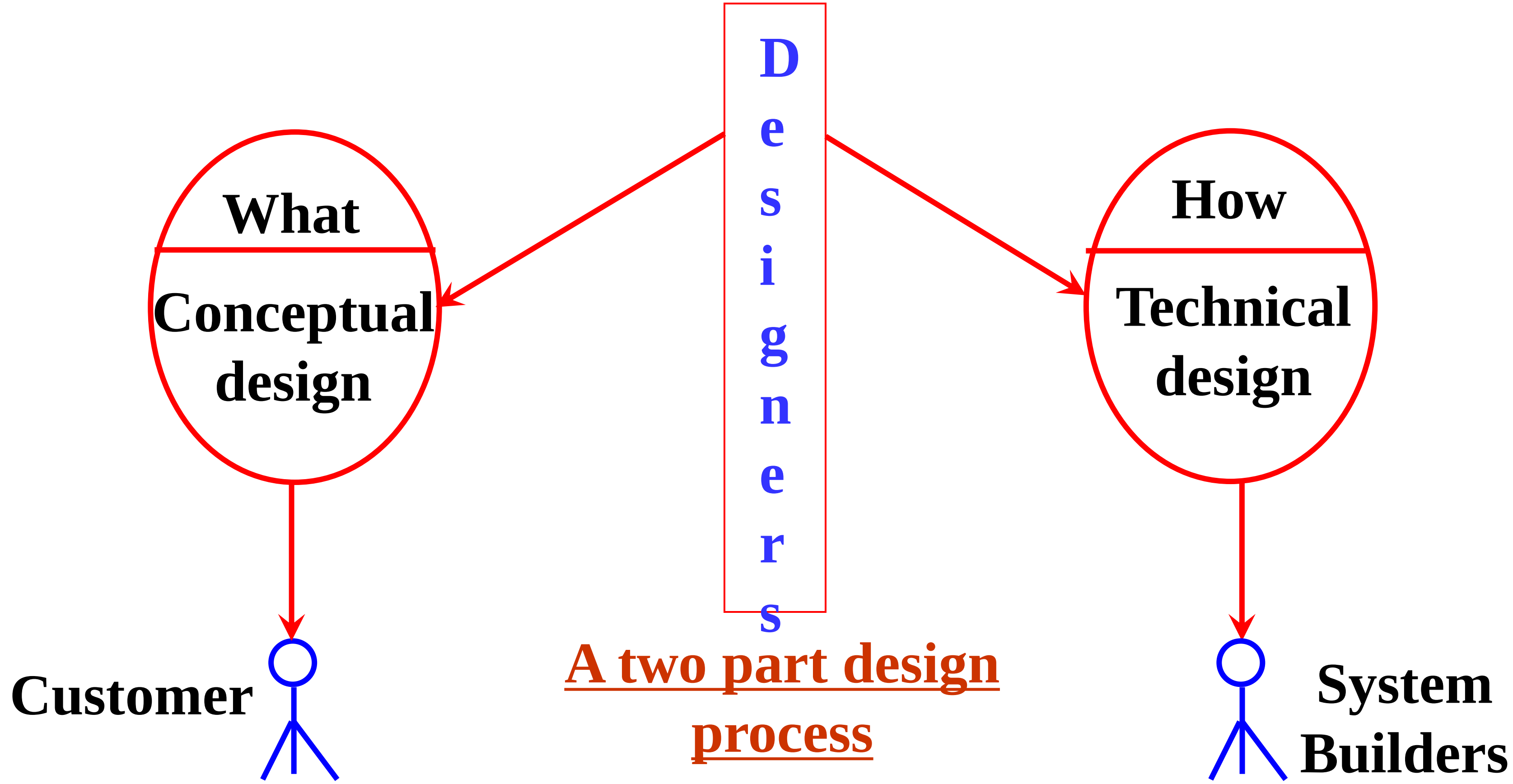
Design framework

Software Design



Software Design

Conceptual Design and Technical Design



A two part design process

Software Design

Conceptual design answers :

- 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?

Software Design

Technical design describes :

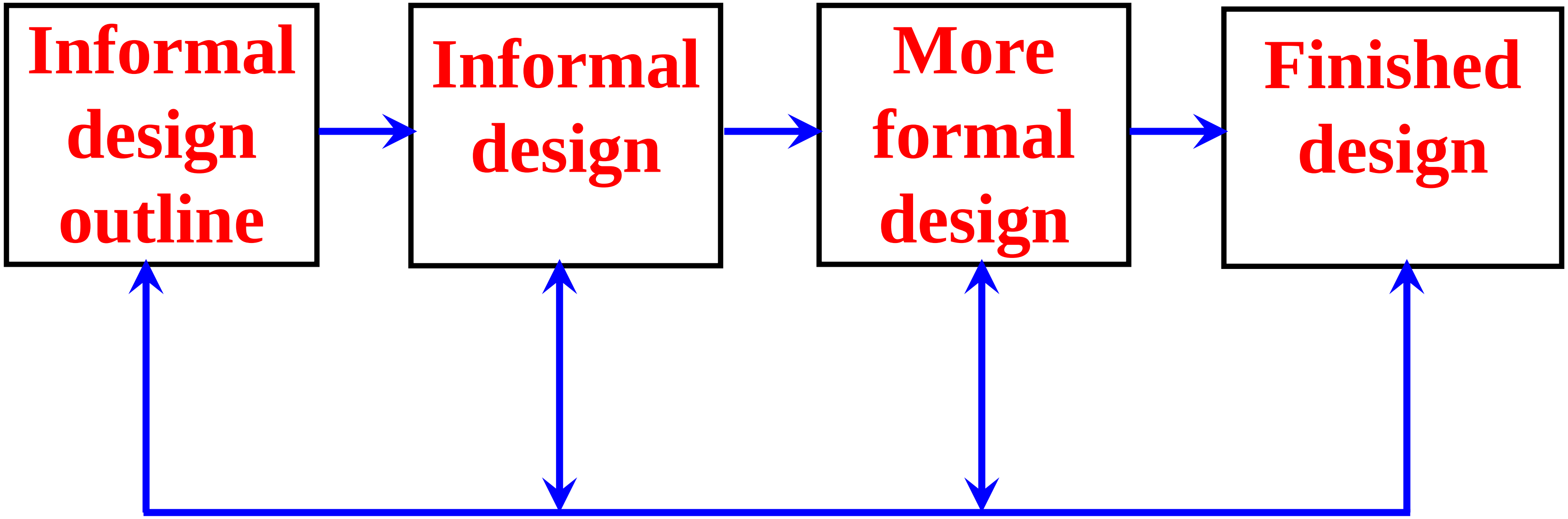
- Hardware configuration
- Software needs
- Communication interfaces
- I/O of the system
- Software architecture
- Network architecture
- Any other thing that translates the requirements in to a solution to the customer's problem.

Software Design

The design needs to be

- Correct & complete
- Understandable
- At the right level
- Maintainable

Software Design



The transformation of an informal design to a detailed design.

Software Design

MODULARITY

There are many definitions of the term module. Range is from :

- Fortran subroutine
- Ada package
- Procedures & functions of PASCAL & C
- C++ / Java classes
- Java packages
- Work assignment for an individual programmer

Software Design

All these definitions are correct. A modular system consist of well defined manageable units with well defined interfaces among the units.

Software Design

Properties :

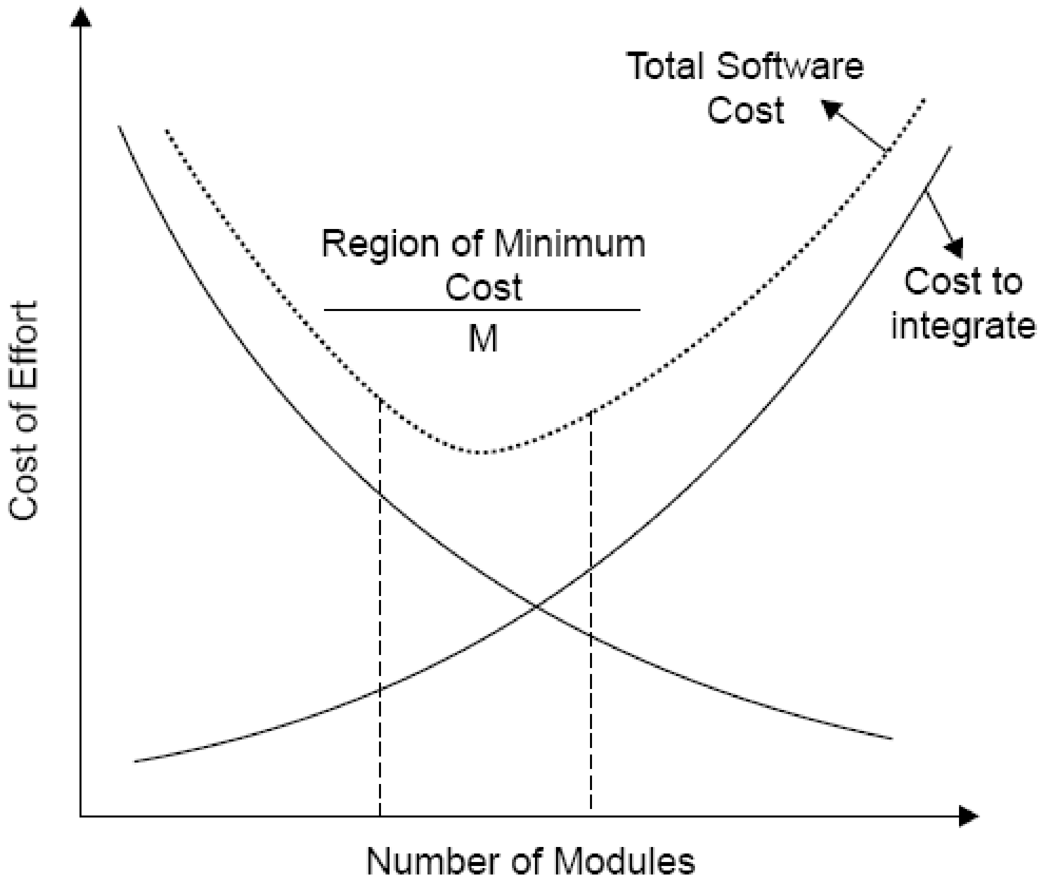
- Well defined subsystem
- Well defined purpose
- Can be separately compiled and stored in a library.
- Module can use other modules
- Module should be easier to use than to build
- Simpler from outside than from the inside.

Software Design

- 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.

Software Design

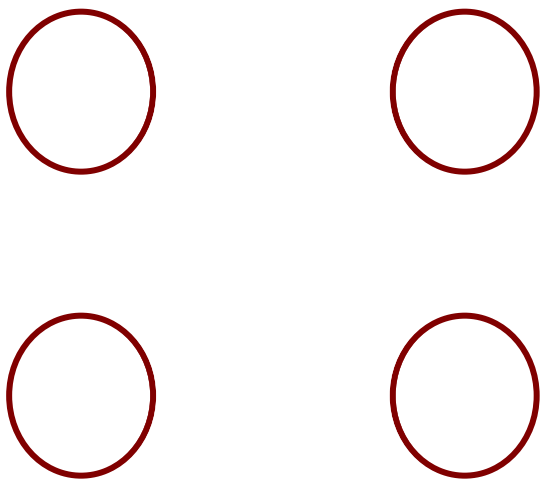


Modularity and software cost

Software Design

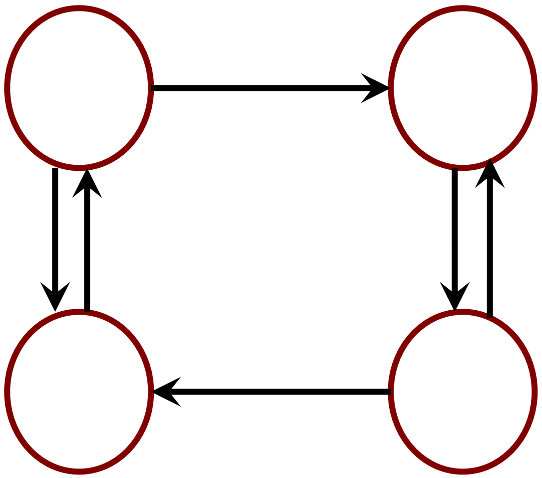
Module Coupling

Coupling is the measure of the degree of interdependence between modules.



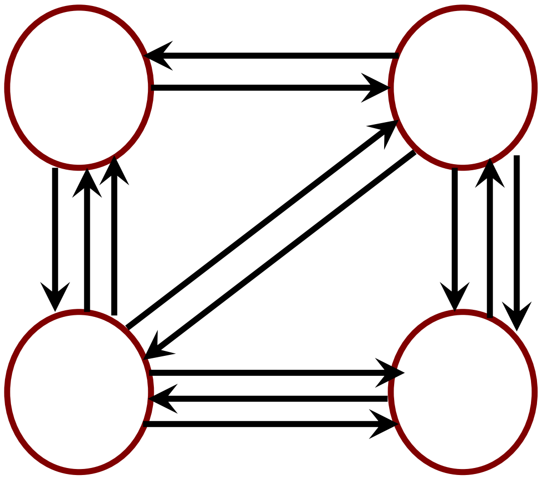
(Uncoupled : no dependencies)

Software Design



Loosely coupled: some dependencies

(B)



Highly coupled: many dependencies

(C)

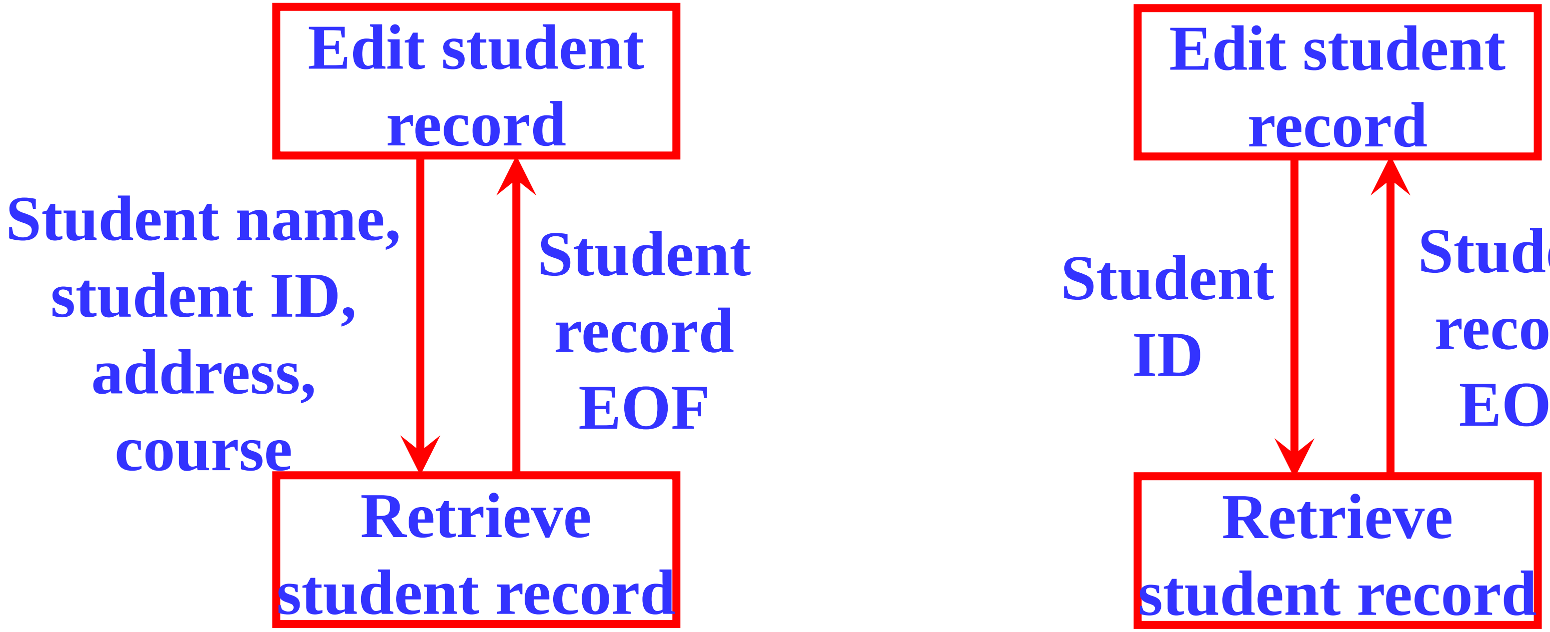
Module coupling

Software Design

This can be achieved as:

- ❖ Controlling the number of parameters passed amongst modules.
- ❖ Avoid passing undesired data to calling module.
- ❖ Maintain parent / child relationship between calling & called modules.
- ❖ Pass data, not the control information.

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



Poor design: Tight Coupling

Good design: Loose Coupling

Example of coupling

Software Design

Data coupling	Best
Stamp coupling	↑
Control coupling	
External coupling	
Common coupling	
Content coupling	Worst

The types of module coupling

Given two procedures A & B, we can identify number of ways in which they can be coupled.

Software Design

Data coupling

The dependency between module A and B is said to be data coupled if their dependency is based on the fact they communicate by only passing of data. Other than communicating through data, the two modules are independent.

Stamp coupling

Stamp coupling occurs between module A and B when complete data structure is passed from one module to another.

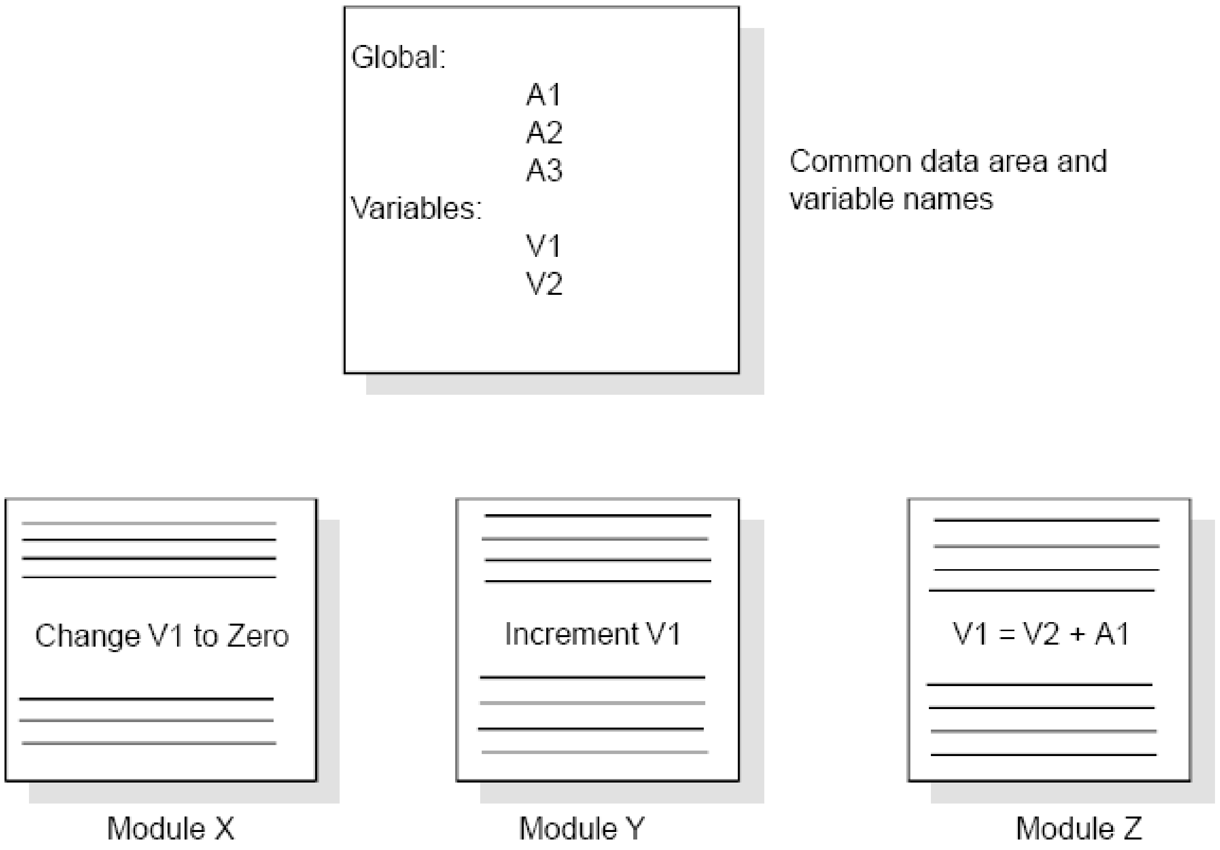
Control coupling

Module A and B are said to be control coupled if they communicate by passing of control information. This is usually accomplished by means of flags that are set by one module and reacted upon by the dependent module.

Common coupling

With common coupling, module A and module B have shared data. Global data areas are commonly found in programming languages. Making a change to the common data means tracing back to all the modules which access that data to evaluate the effect of changes.

Software Design

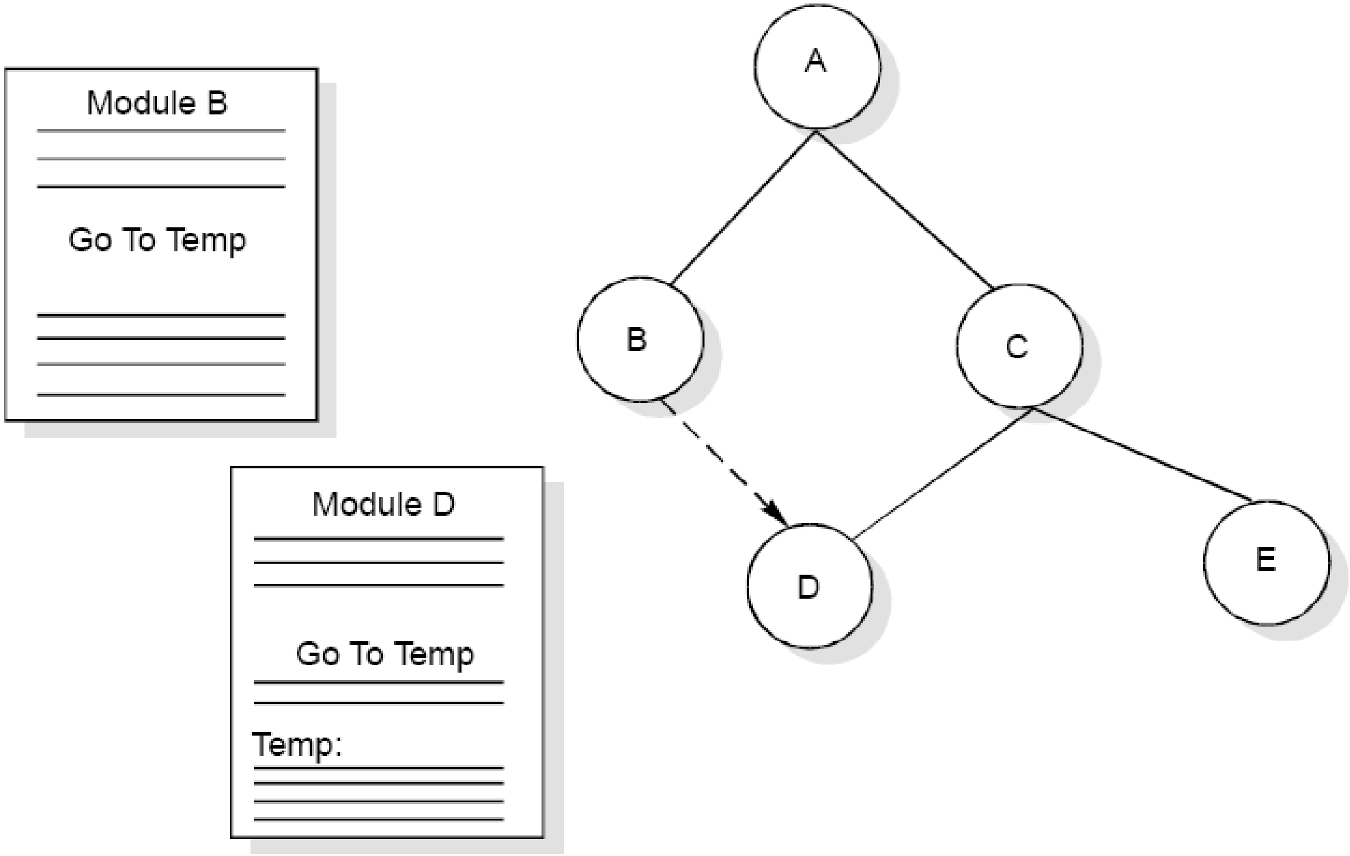


Example of common coupling

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. 9, module B branches into D, even though D is supposed to be under the control of C.

Software Design

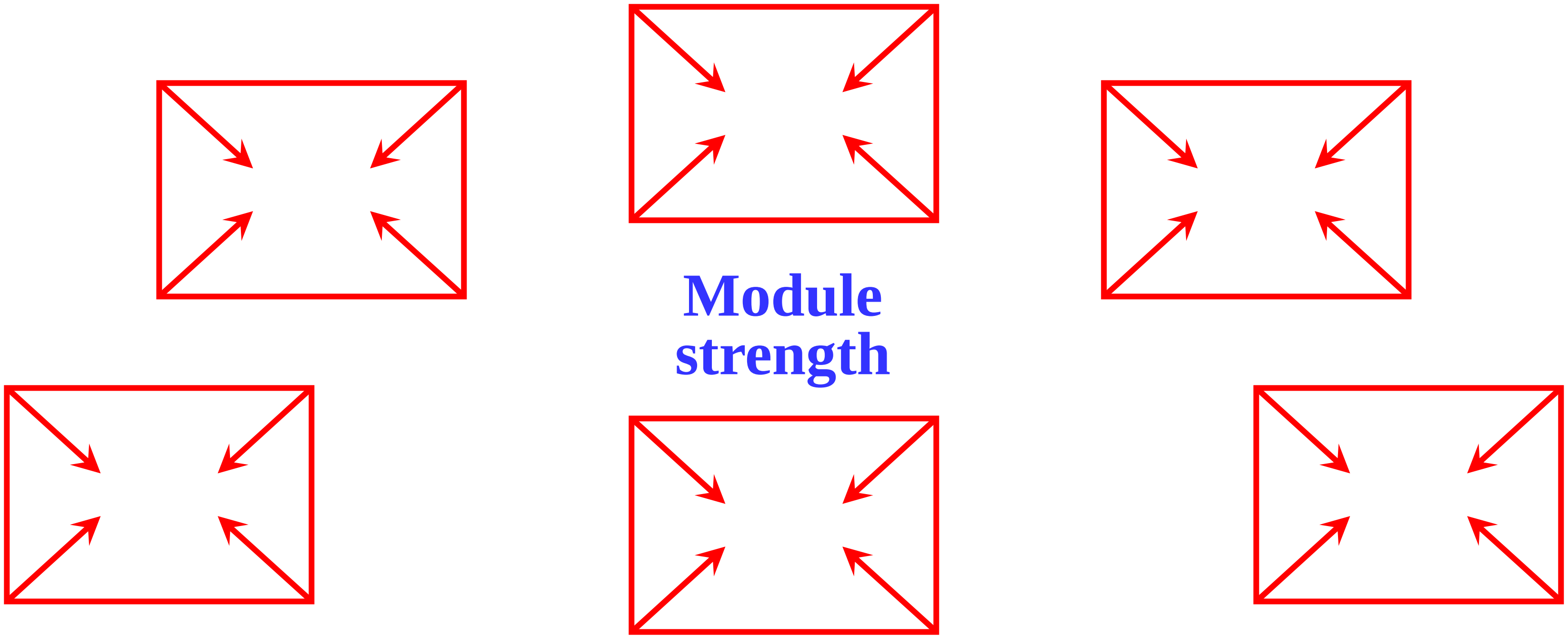


Example of content coupling

Software Design

Module Cohesion

Cohesion is a measure of the degree to which the elements of a module are functionally related.



Cohesion=Strength of relations within modules

Software Design

Types of cohesion

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

Software Design

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

Types of module cohesion

Software Design

Functional Cohesion

A and B are part of a single functional task. This is very good reason for them to be contained in the same procedure.

Sequential Cohesion

Module A outputs some data which forms the input to B. This is the reason for them to be contained in the same procedure.

Software Design

Procedural Cohesion

Procedural Cohesion occurs in modules whose instructions although accomplish different tasks yet have been combined because there is a specific order in which the tasks are to be completed.

Temporal Cohesion

Module exhibits temporal cohesion when it contains tasks that are related by the fact that all tasks must be executed in the same time-span.

Logical Cohesion

Logical cohesion occurs in modules that contain instructions that appear to be related because they fall into the same logical class of functions.

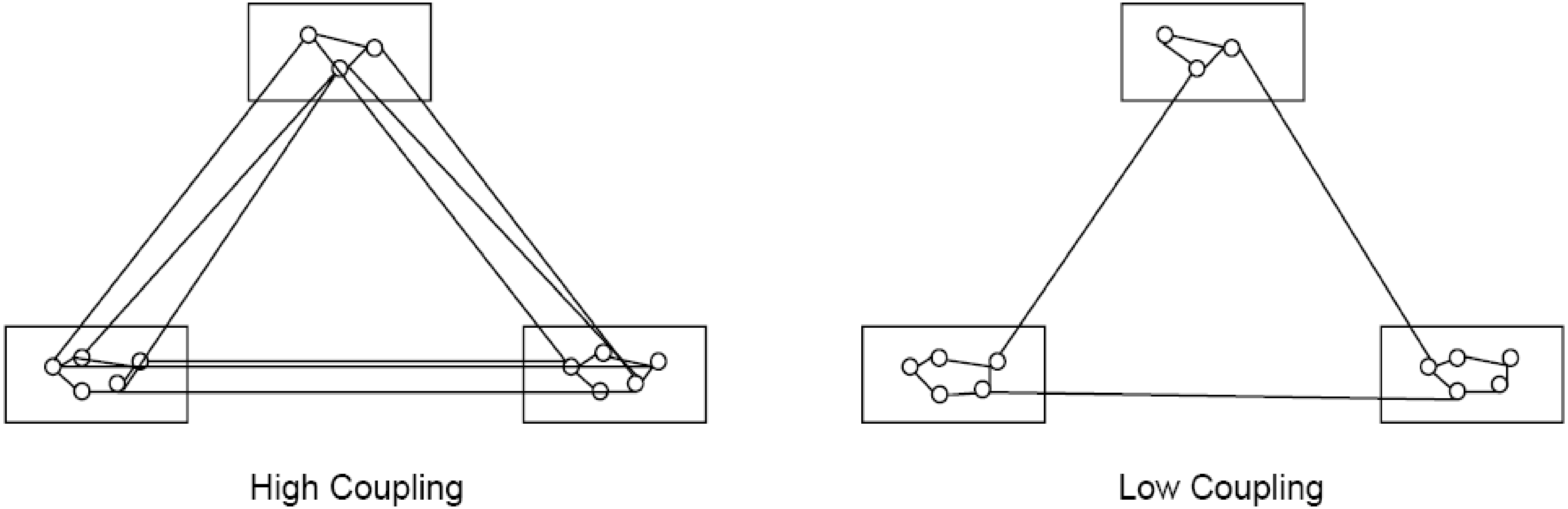
Coincidental Cohesion

Coincidental cohesion exists in modules that contain instructions that have little or no relationship to one another.

Software Design

Relationship between Cohesion & Coupling

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.



View of cohesion and coupling

Software Design

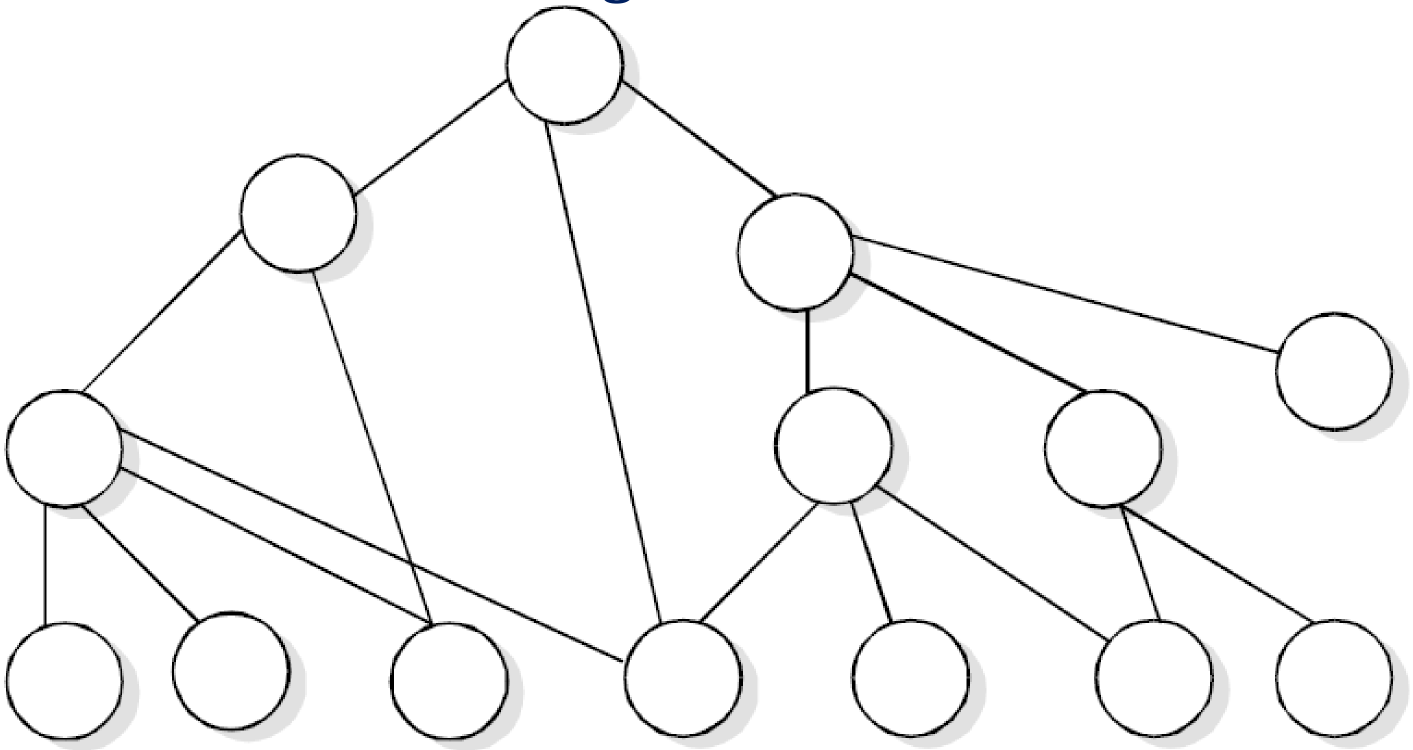
STRATEGY OF DESIGN

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.

Bottom-Up Design

These modules are collected together in the form of a "library".



Bottom-up tree structure

Software Design

Top-Down Design

A top down design approach starts by identifying the major modules of the system, decomposing them into their lower level modules and iterating until the desired level of detail is achieved. This is stepwise refinement; starting from an abstract design, in each step the design is refined to a more concrete level, until we reach a level where no more refinement is needed and the design can be implemented directly.

Software Design

Hybrid Design

For top-down approach to be effective, some bottom-up approach is essential for the following reasons:

To permit common sub modules.

Near the bottom of the hierarchy, where the intuition is simpler, and the need for bottom-up testing is greater, because there are more number of modules at low levels than high levels.

In the use of pre-written library modules, in particular, reuse of modules.

Software Design

FUNCTION ORIENTED DESIGN

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.

Software Design

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

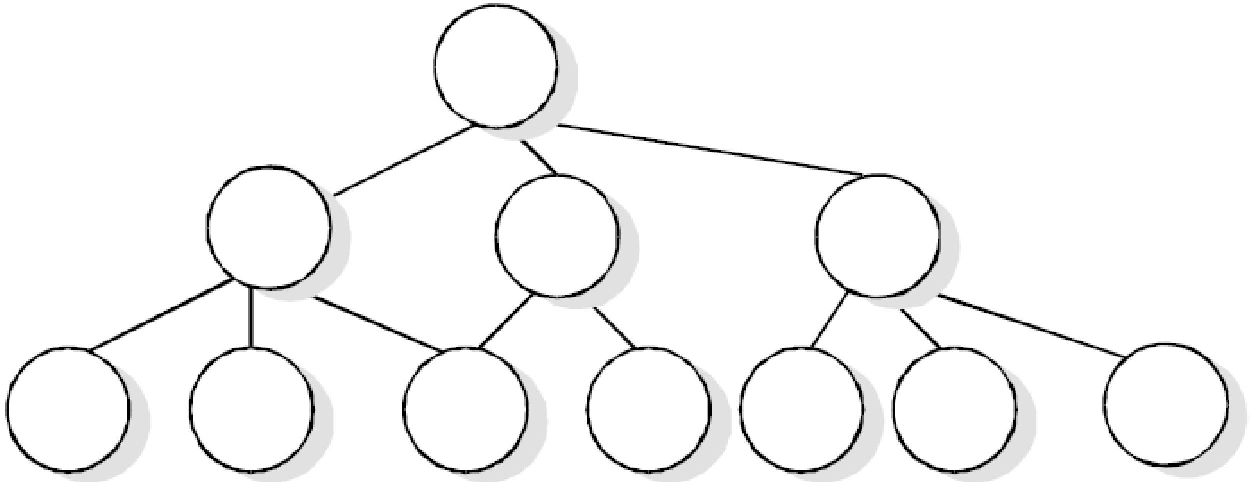
```
While (not finished)
{
        Read an expression from the terminal;
        Evaluate the expression;
        Print the value;
}
```

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*/
        :::
}
```

Software Design

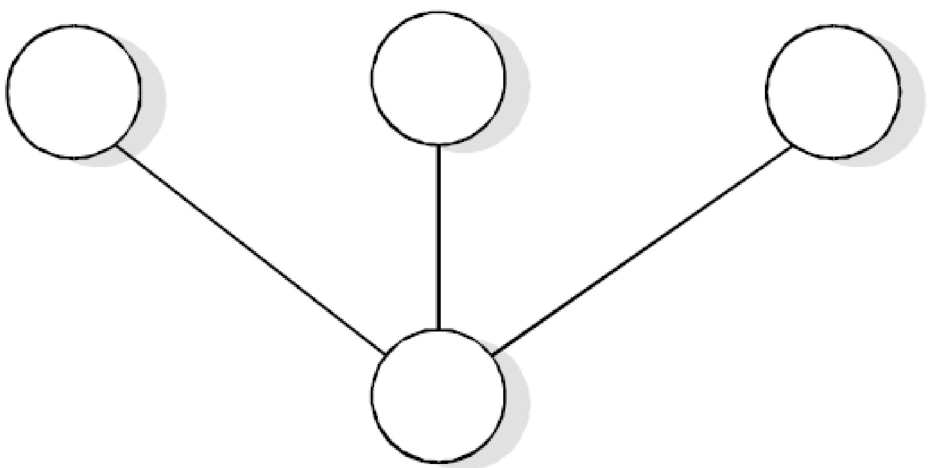
We continue the refinement of each module until we reach the statement level of our programming language. At that point, we can describe the structure of our program as a tree of refinement as in design top-down structure as shown below:



Top-down structure

Software Design

If a program is created top-down, the modules become very specialized. As one can easily see in top down design structure, each module is used by at most one other module, its parent. For a module, however, we must require that several other modules as in design reusable structure as shown below:



Design reusable structure

Software Design

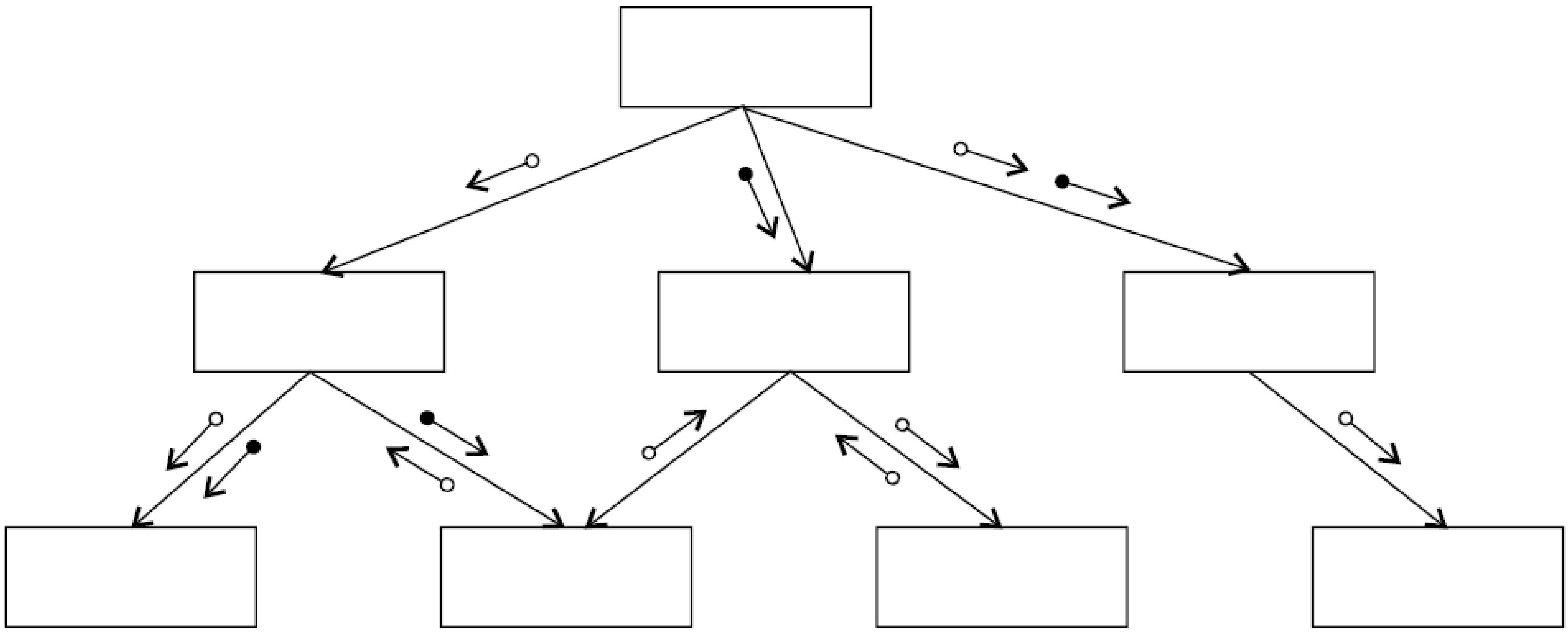
Design Notations

Design notations are largely 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**

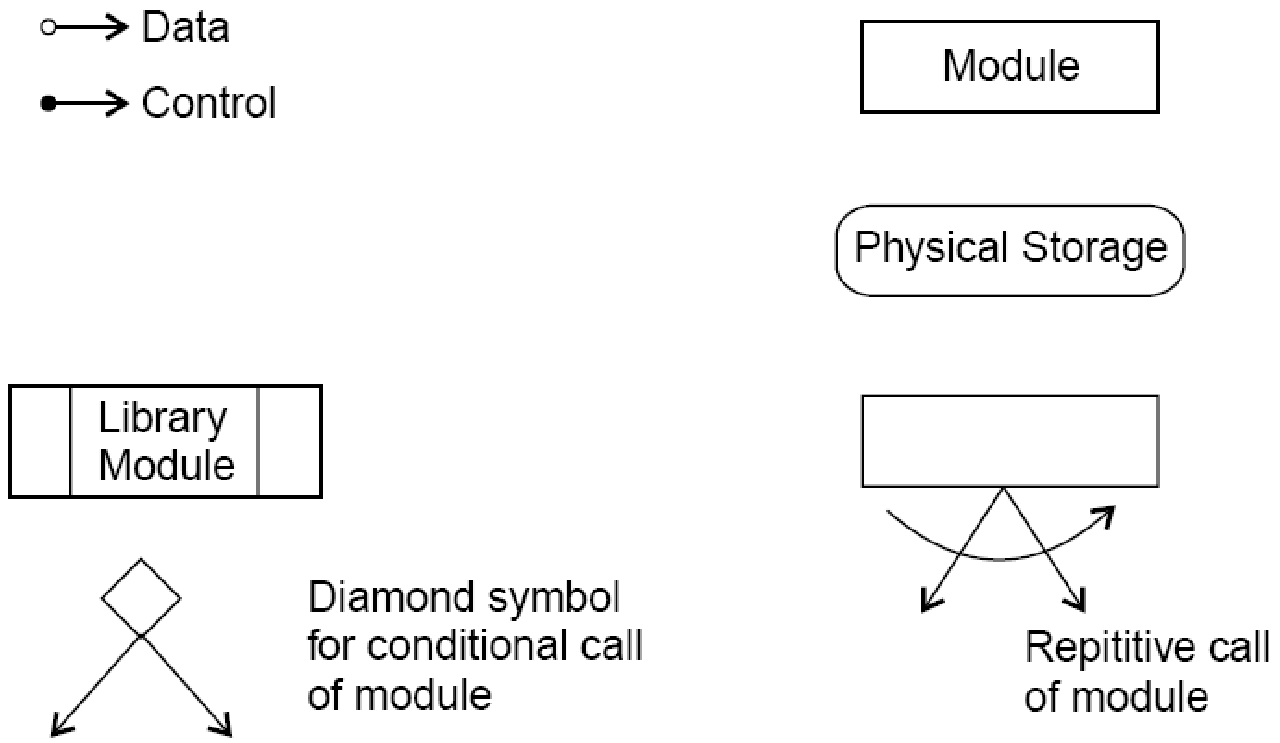
Structure Chart

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



Hierarchical format of a structure chart

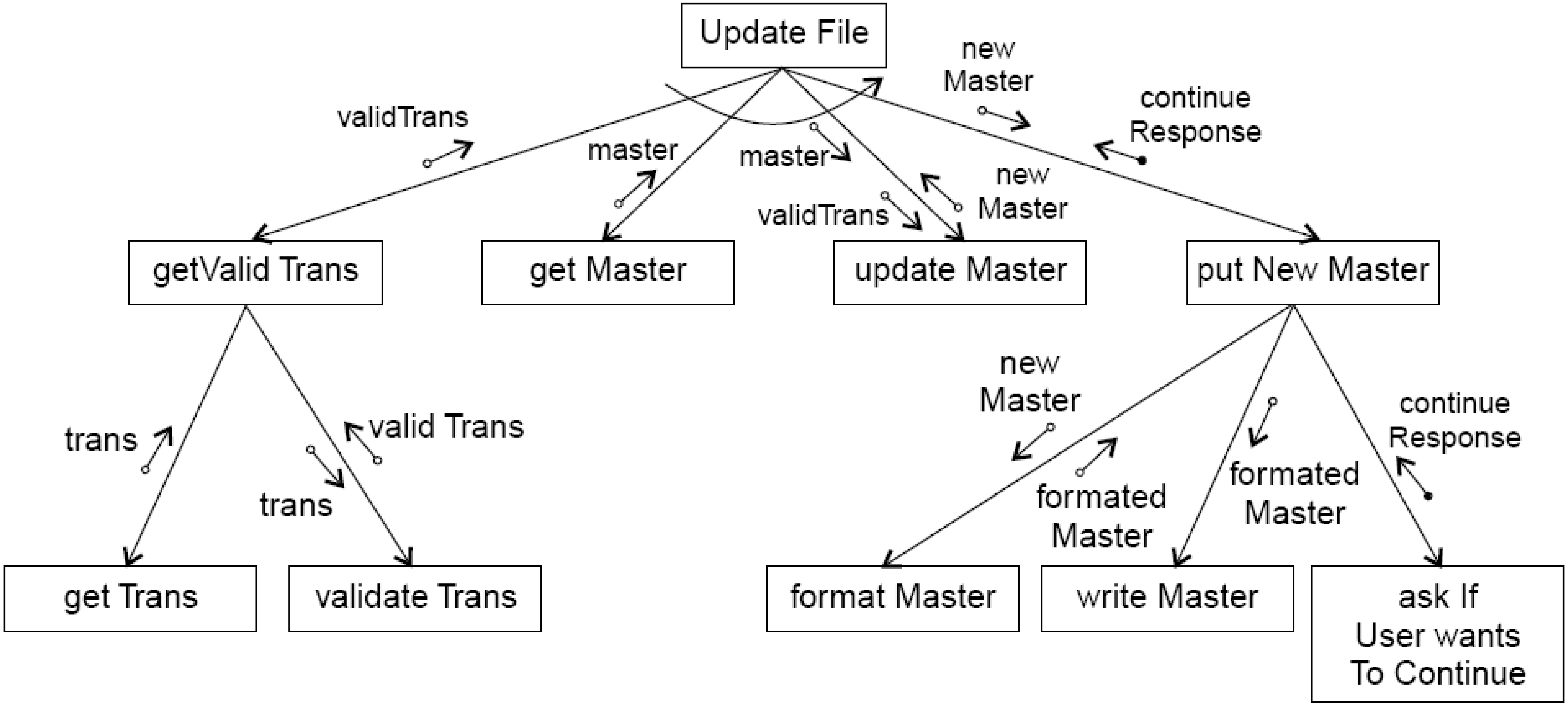
Structure Chart



Structure chart notations

Structure Chart

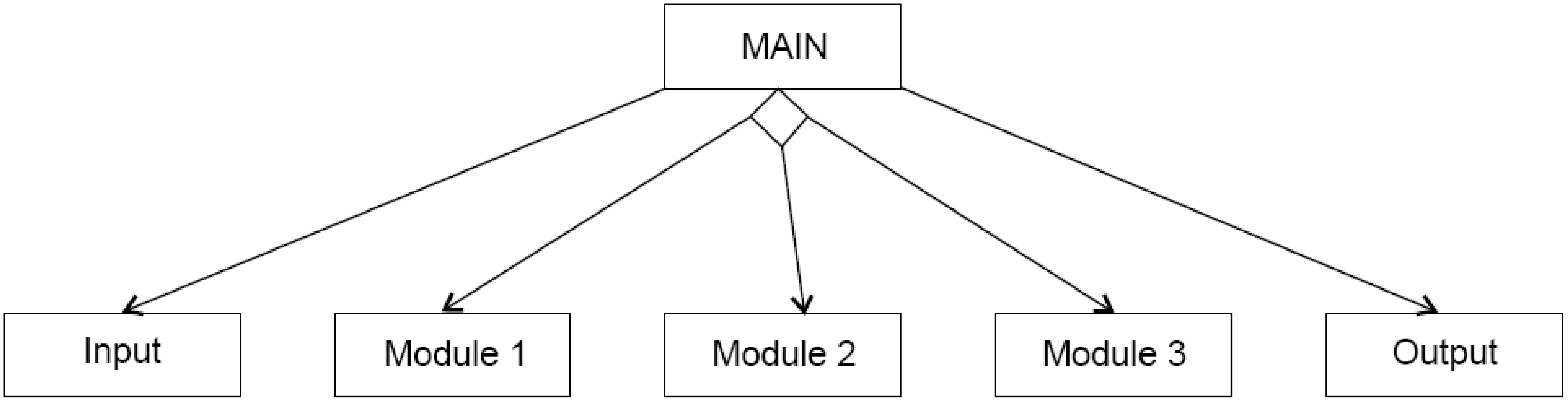
A structure chart for "update file" is given below.



Update file

Structure Chart

A transaction centered structure describes a system that processes a number of different types of transactions. It is illustrated in following Figure.



Transaction-centered structure

Structure Chart

In the above figure the MAIN module controls the system operation its functions is to:

- invoke the INPUT module to read a transaction;
- determine the kind of transaction and select one of a number of transaction modules to process that transaction, and
- output the results of the processing by calling OUTPUT module.

Pseudocode

- **Pseudocode notation can be used in both the preliminary and detailed design phases.**
- **Using pseudocode, 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.**

Object Oriented Design

- 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. Thus, it is orthogonal to function oriented design.

- Object Oriented Design begins with an examination of the real world "things" that are part of the problem to be solved. These things (which we will call objects) are characterized individually in terms of their attributes and behavior.

Object Oriented Design

Basic Concepts

- Object Oriented Design is not dependent on any specific implementation language. Problems are modeled using objects. Objects have:
- Behavior (they do things)
- State (which changes when they do things)

Object Oriented Design

The various terms related to object design are:

Objects

The word "Object" is used very frequently and conveys different meaning in different circumstances. Here, meaning is an entity able to save a state (information) and which offers a number of operations (behavior) to either examine or affect this state. An object is characterized by number of operations and a state which remembers the effect of these operations.

Object Oriented Design

Messages

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

Abstraction

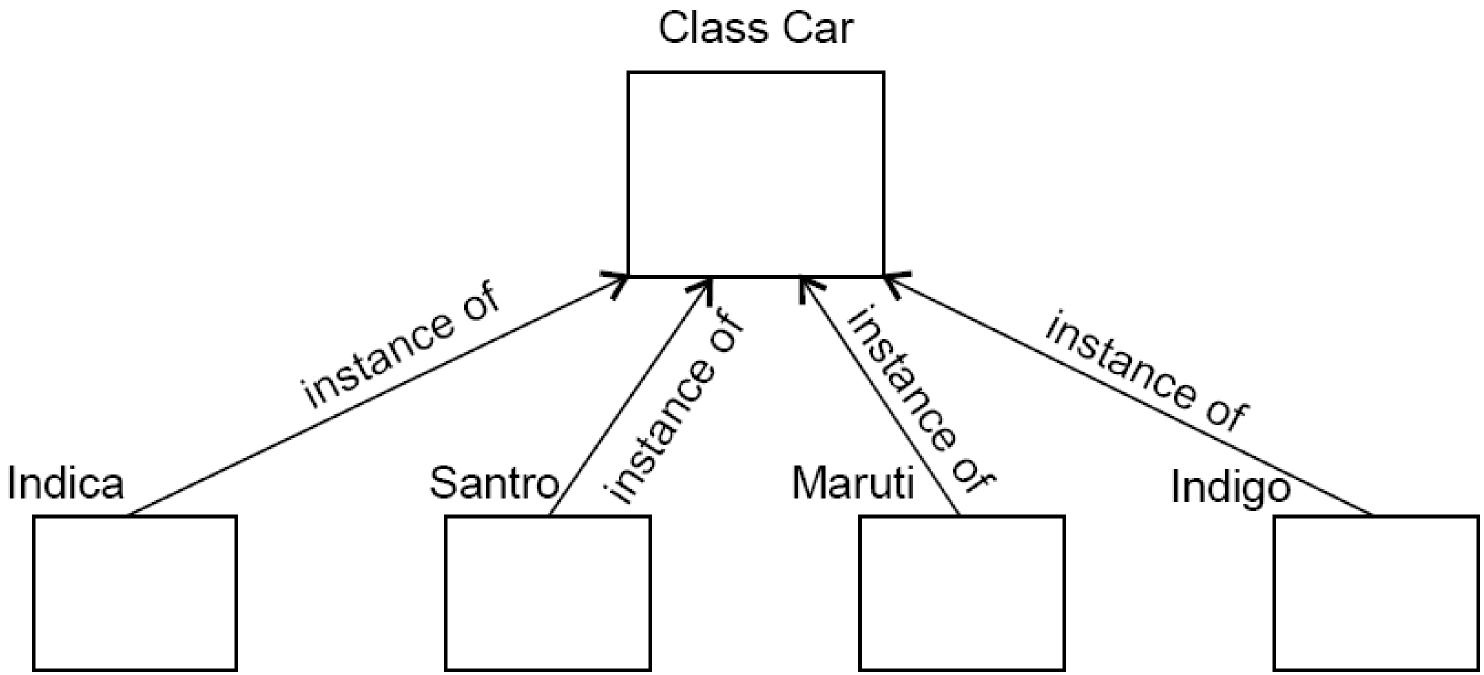
In object oriented design, complexity is managed using abstraction. Abstraction is the elimination of the irrelevant and the amplification of the essentials.

Object Oriented Design

Class

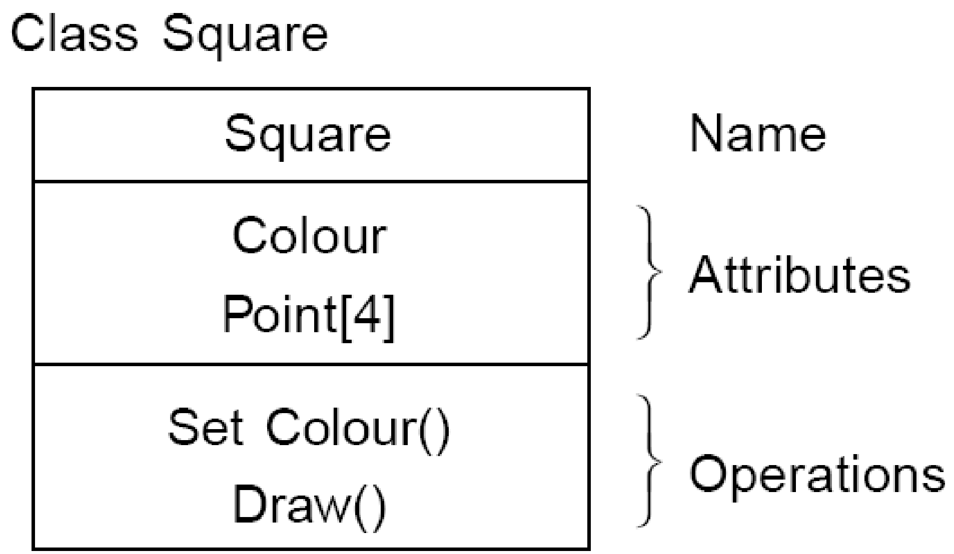
- In any system, there shall be number of objects. Some of the objects may have common characteristics and we can group the objects according to these characteristics. This type of grouping is known as a class. Hence, a class is a set of objects that share a common structure and a common behavior.
- We may define a class "car" and each object that represent a car becomes an instance of this class. In this class "car", Indica, Santro, Maruti, Indigo are instances of this class as shown in fig. 20.
- Classes are useful because they act as a blueprint for objects. If we want a new square we may use the square class and simply fill in the particular details (i.e. color and position). Following Figure shows how can we represent the square class.

Object Oriented Design



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

Object Oriented Design



The square class

Object Oriented Design

Attributes

An attributes is a data value held by the objects in a class. The square class has two attributes: a colour and array of points. Each attributes has a value for each object instance. The attributes are shown as second part of the class as shown in above figure.

Operations

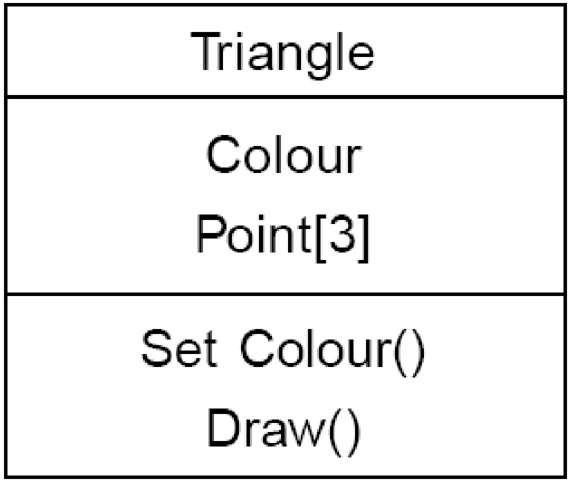
An operation is a function or transformation that may be applied to or by objects in a class. In the square class, we have two operations: set colour() and draw(). All objects in a class share the same operations. An object "knows" its class, and hence the right implementation of the operation. Operation are shown in the third part of the class as indicated in above figure.

Object Oriented Design

Inheritance

Imagine that, as well as squares, we have triangle class. Following Figure shows the class for a triangle.

Class Triangle

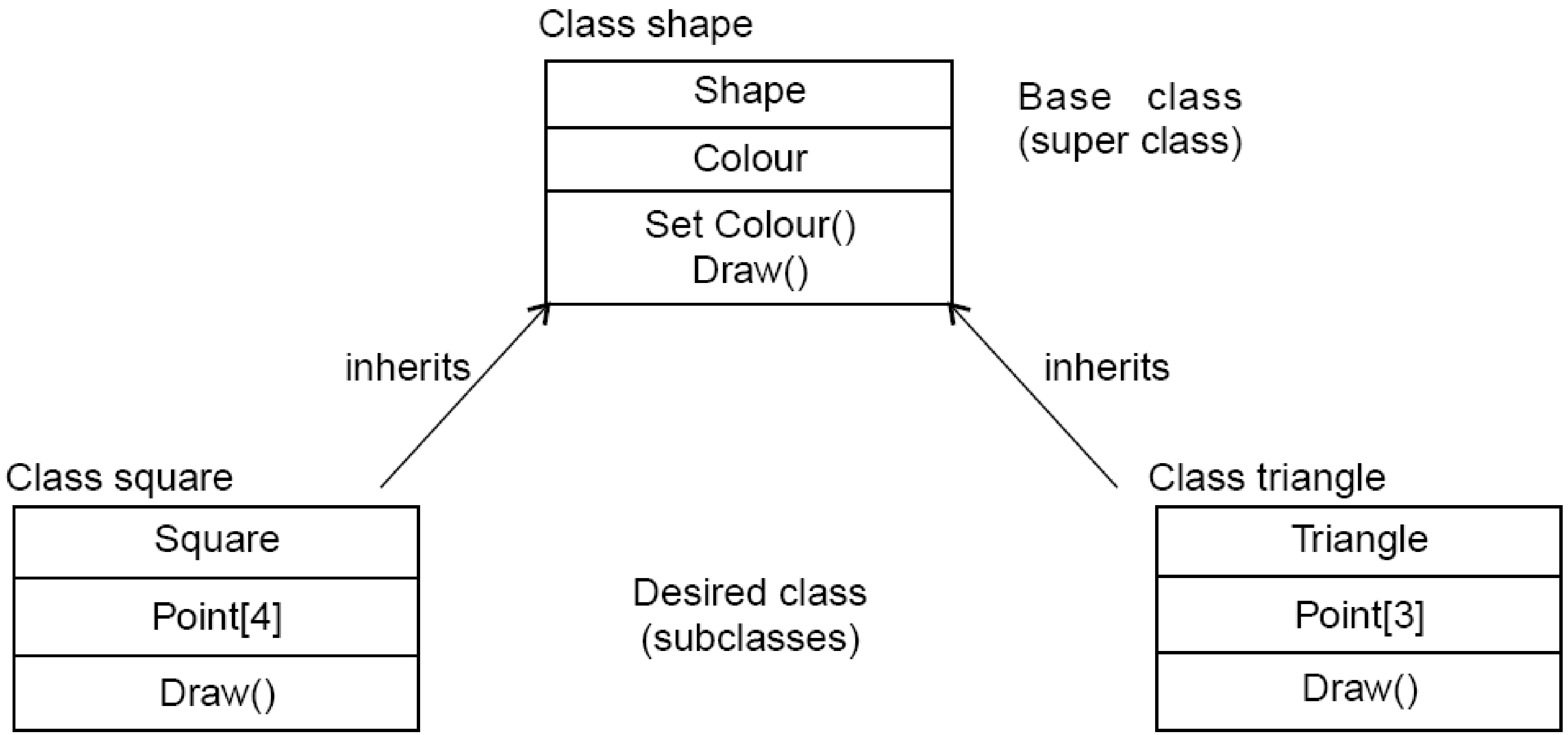


The triangle class

Object Oriented Design

- Now, comparing above two Figures, we can see that there is some difference between triangle and squares classes.
- For example, at a high level of abstraction, we might want to think of a picture as made up of shapes and to draw the picture, we draw each shape in turn. We want to eliminate the irrelevant details: we do not care that one shape is a square and the other is a triangle as long as both can draw themselves.
- To do this, we consider the important parts out of these classes in to a new class called Shape. Fig. 23 shows the results.

Object Oriented Design



Abstracting common features in a new class

This sort of abstraction is called inheritance. The low level classes (known as subclasses or derived classes) inherit state and behavior from this high level class (known as a super class or base class).

Object Oriented Design

Polymorphism

When we abstract just the interface of an operation and leave the implementation to subclasses it is called a polymorphic operation and process is called polymorphism.

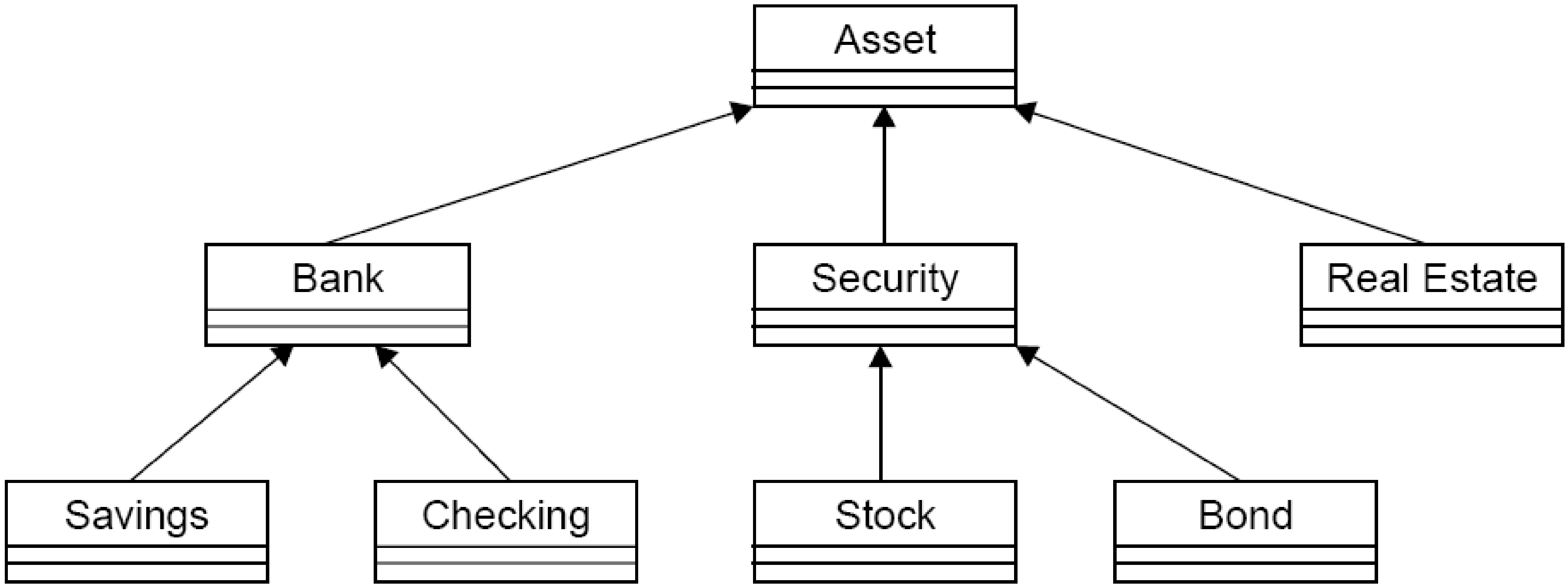
Encapsulation (Information Hiding)

Encapsulation is also commonly referred to as “Information Hiding”. It consists of the separation of the external aspects of an object from the internal implementation details of the object.

Hierarchy

Hierarchy involves organizing something according to some particular order or rank. It is another mechanism for reducing the complexity of software by being able to treat and express sub-types in a generic way.

Object Oriented Design



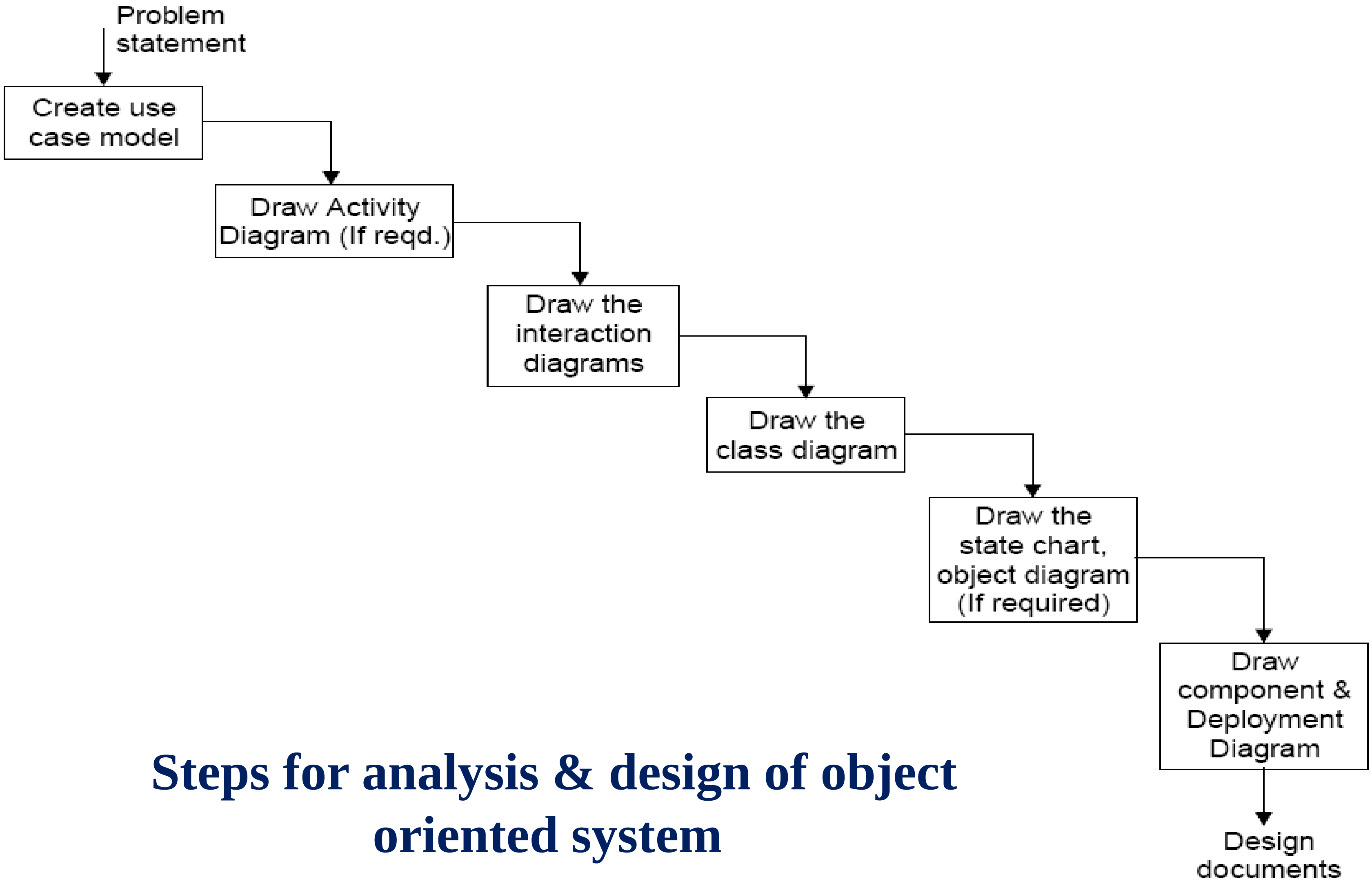
Hierarchy

Object Oriented Design

Steps to Analyze and Design Object Oriented System

There are various steps in the analysis and design of an object oriented system are given in following Figure.

Object Oriented Design



Steps for analysis & design of object oriented system

Object Oriented Design

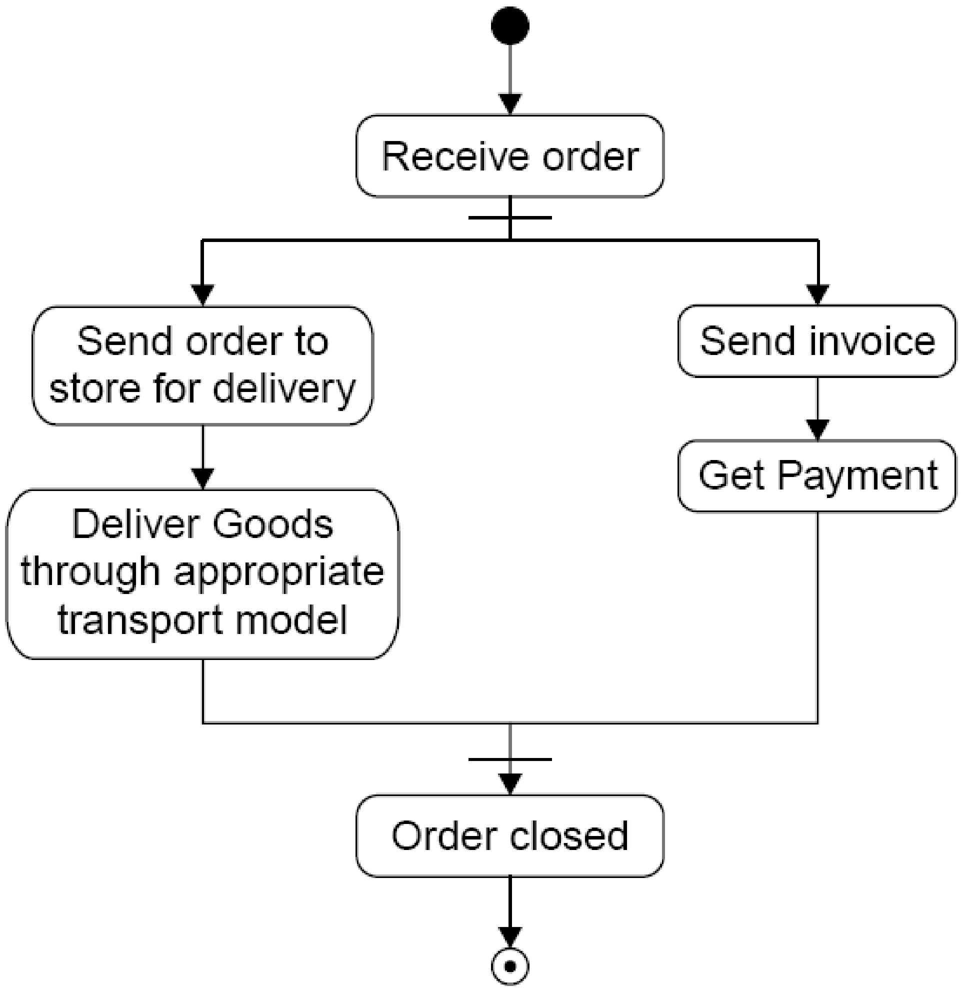
Create use case model

First step is to identify the actors interacting with the system. We should then write the use case and draw the use case diagram.

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. Following Figure shows the activity diagram processing an order to deliver some goods.

Object Oriented Design



Activity diagram

Object Oriented Design

Draw the interaction diagram

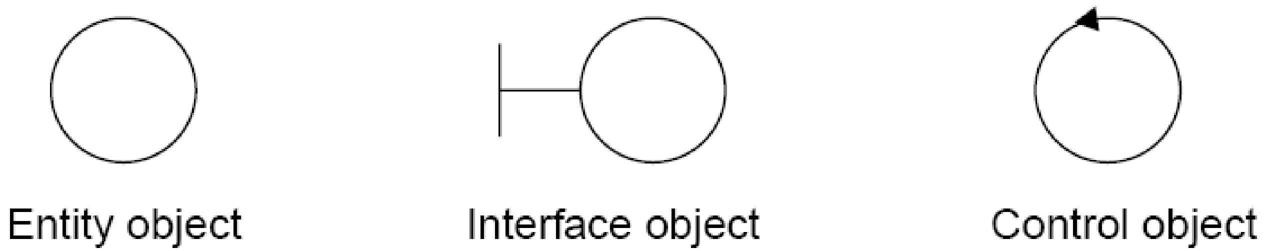
An interaction diagram shows an interaction, consisting of a set of objects and their relationship, including the messages that may be dispatched among them. Interaction diagrams address the dynamic view of a system.

Steps to draws interaction diagrams are as under:

- ❖ **Firstly, we should identify that the objects with respects to every use case.**
- ❖ **We draw the sequence diagrams for every use case.**
- ❖ **We draw the collaboration diagrams for every use case.**

Object Oriented Design

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



Object types

Object Oriented Design

Draw the class diagram

The class diagram shows the relationship amongst classes. There are four types of relationships in class diagrams.

- **Associations** 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.

Object Oriented Design

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

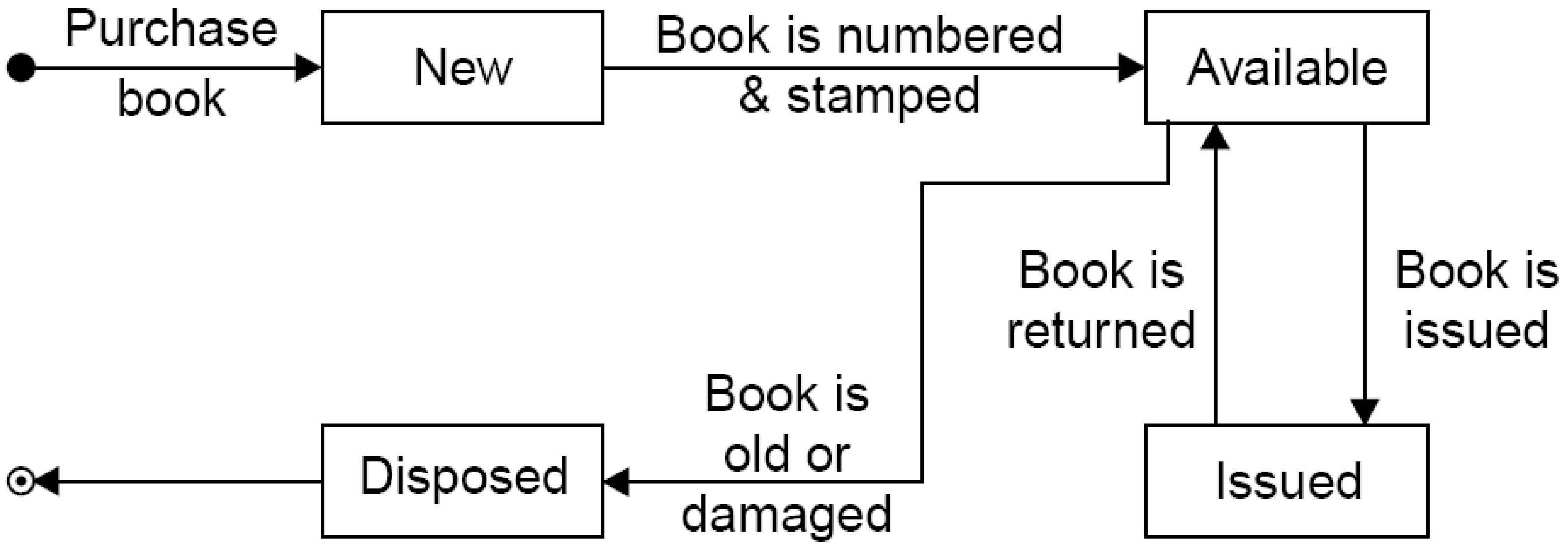
- **Aggregations** are stronger form of association. An aggregation is a relationship between a whole and its parts.

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

Object Oriented Design

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 following diagram.



Transition chart for "book" in a library system

Object Oriented Design

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.
- Deployment Diagram Captures relationship between physical components and the hardware.

Software Design

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.

Software Design

- 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.

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.

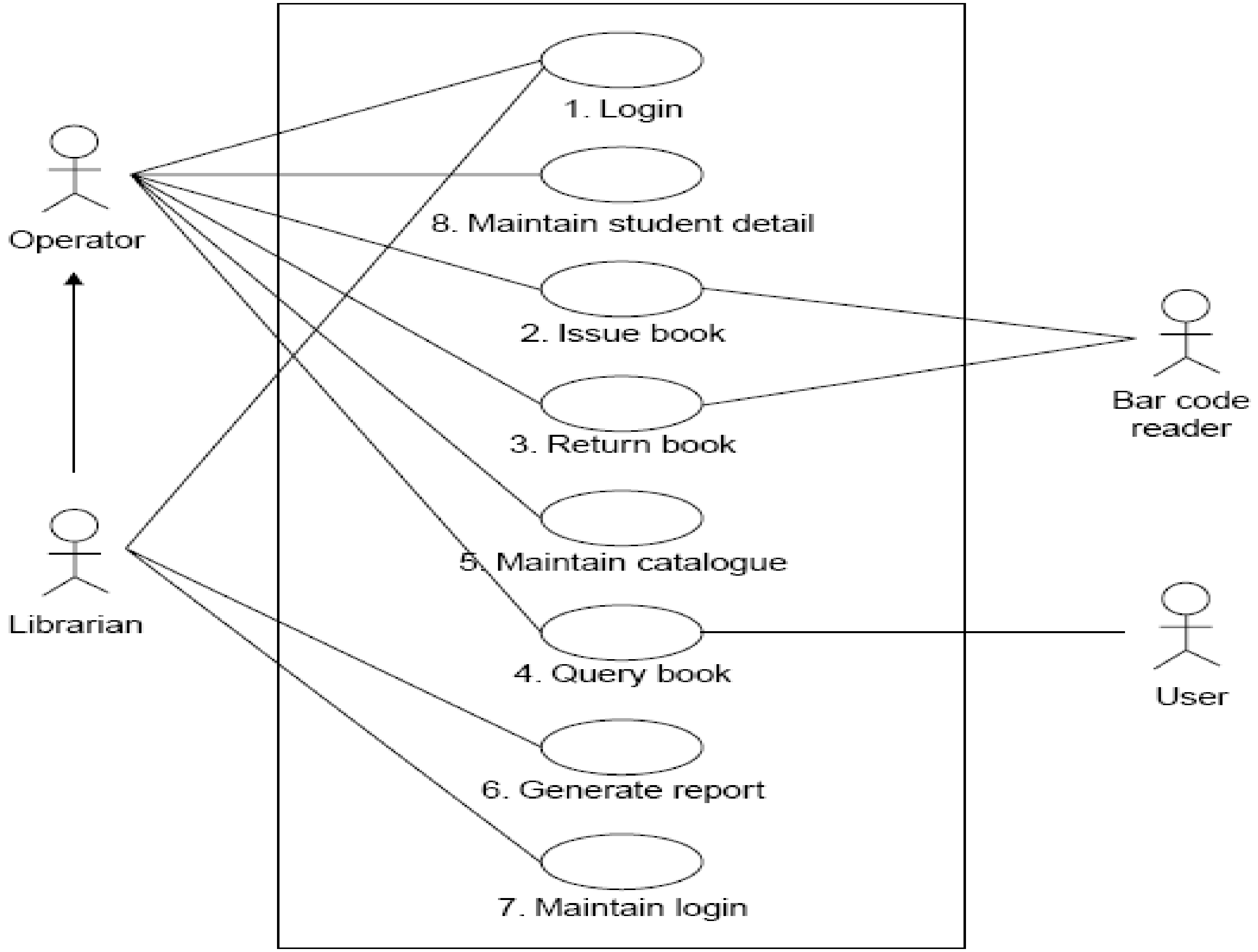
Software Design

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.

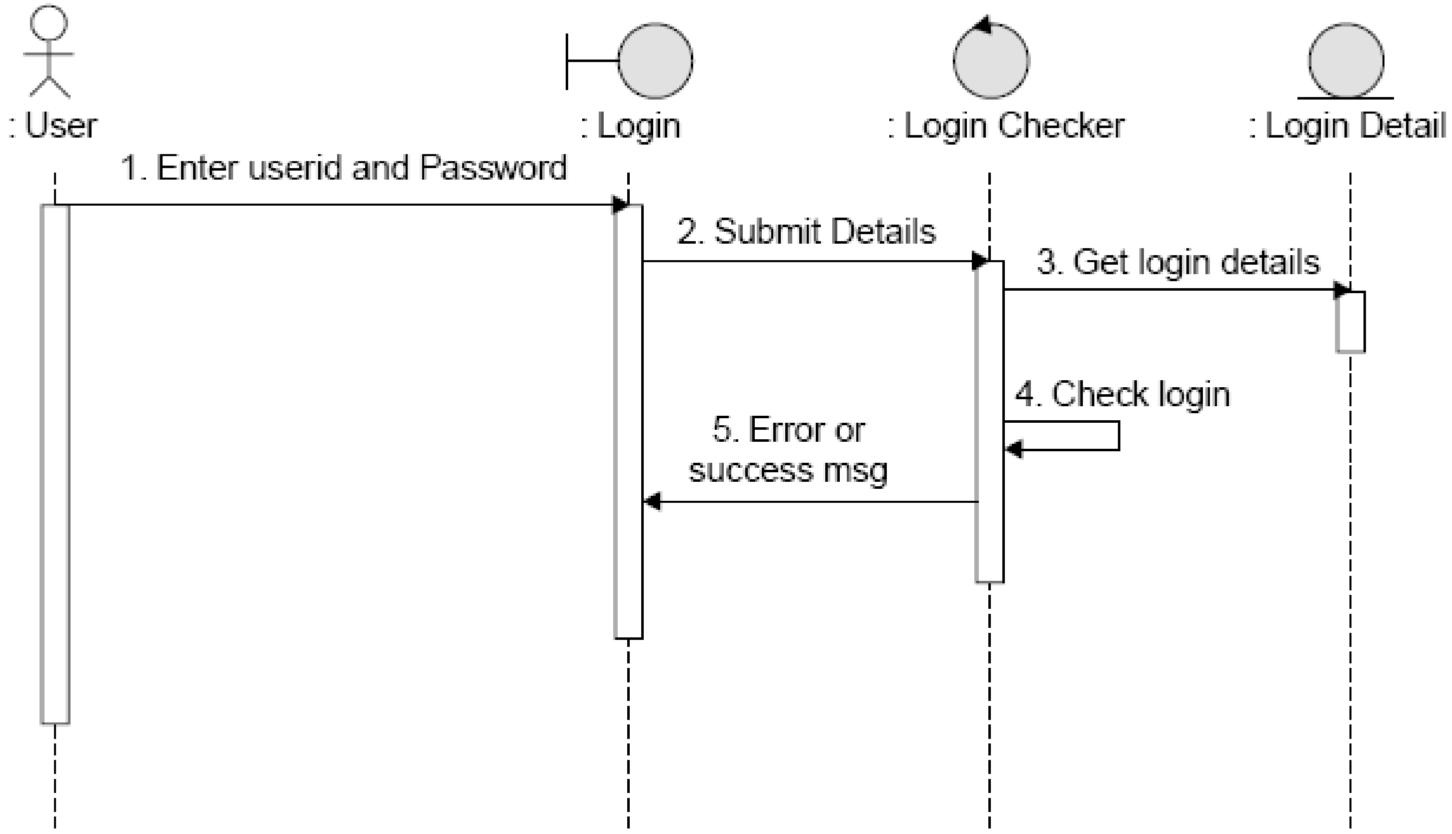
The system should also be able to generate reports regarding the details of the books available in the library at any given time. The corresponding printouts for each entry (issue/return) made in the system should be generated. Security provisions like the ‘login authenticity should be provided. Each 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



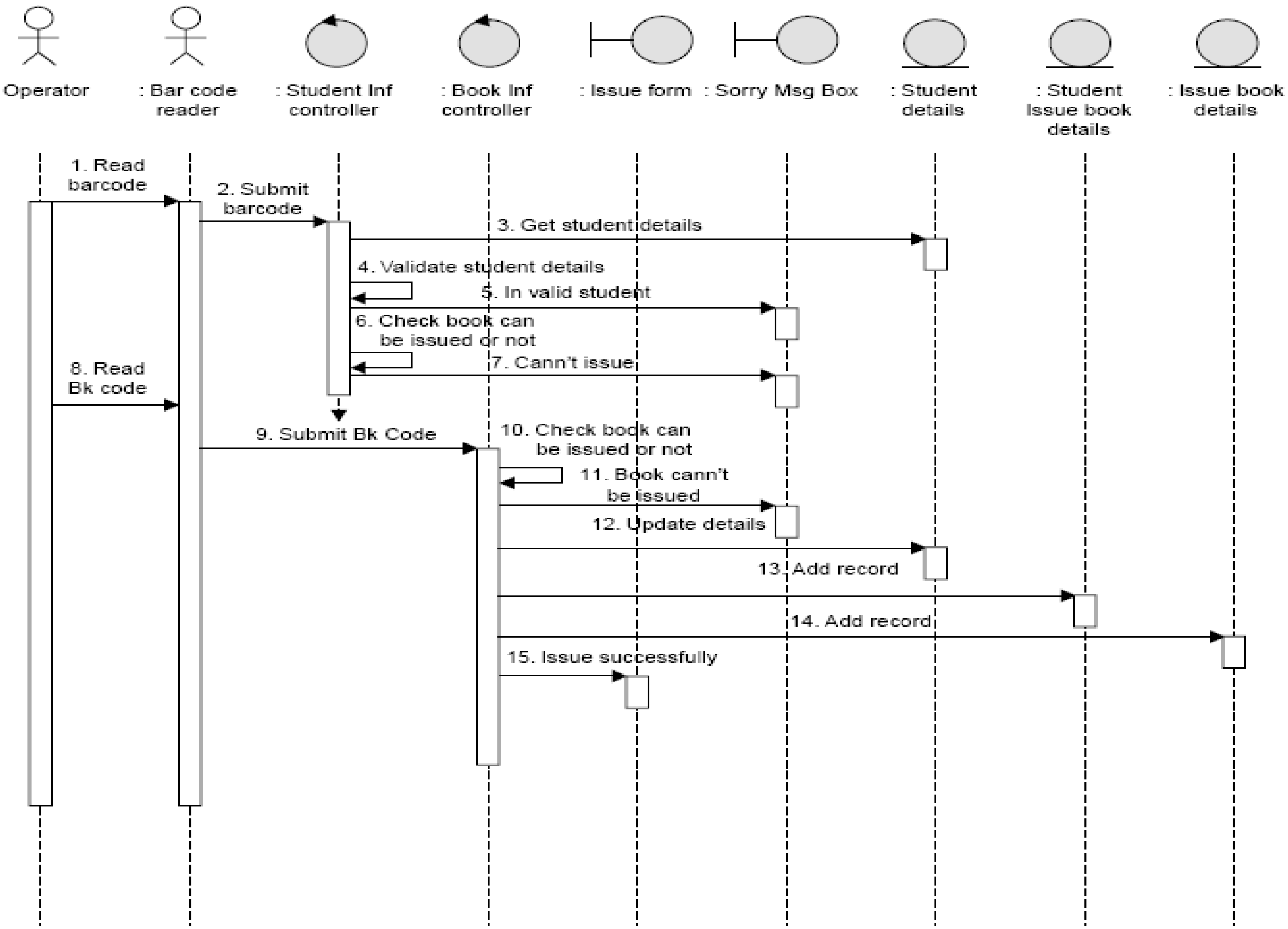
Use case diagram for library management system

Software Design



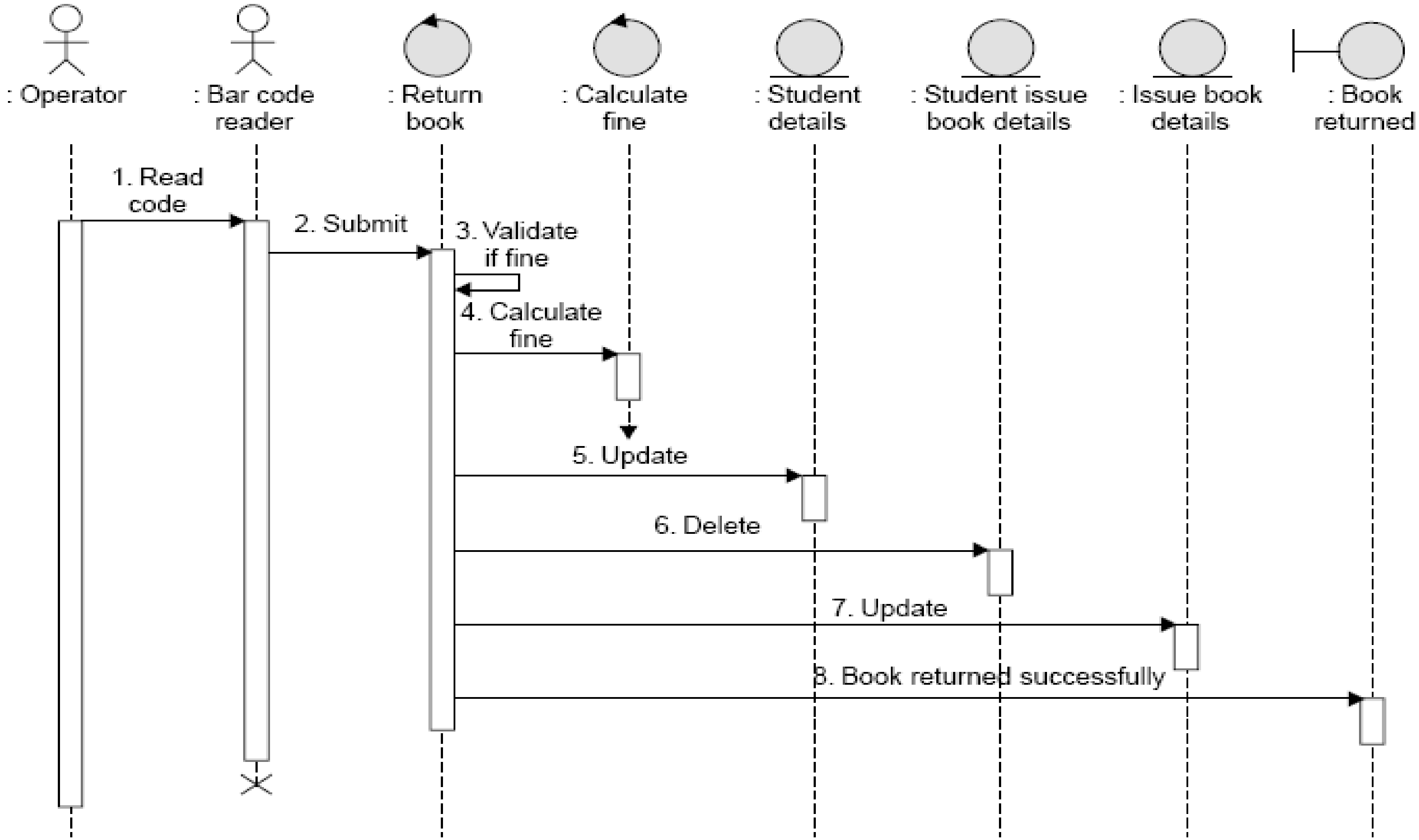
Sequence diagram—Login

Software Design



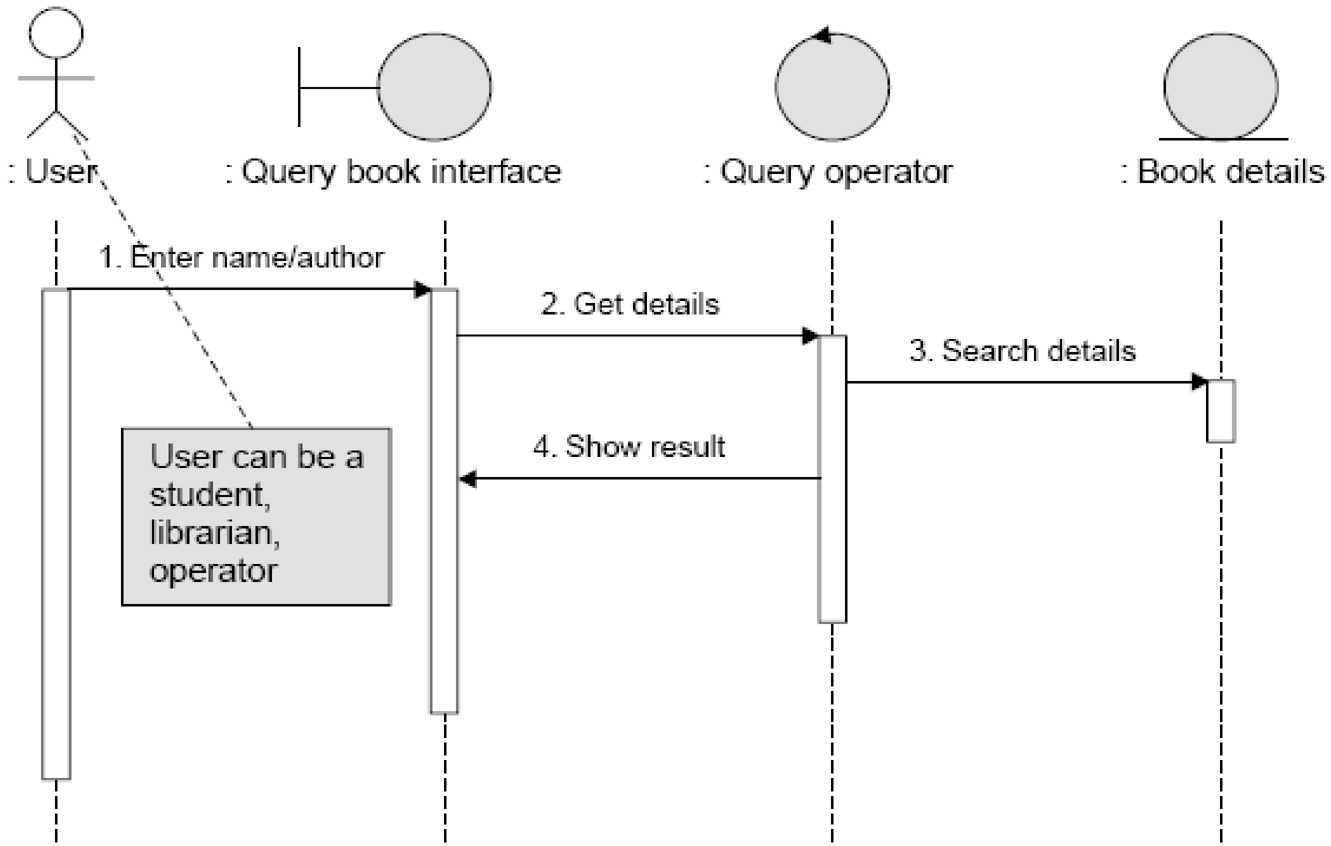
Sequence diagram—issue book

Software Design



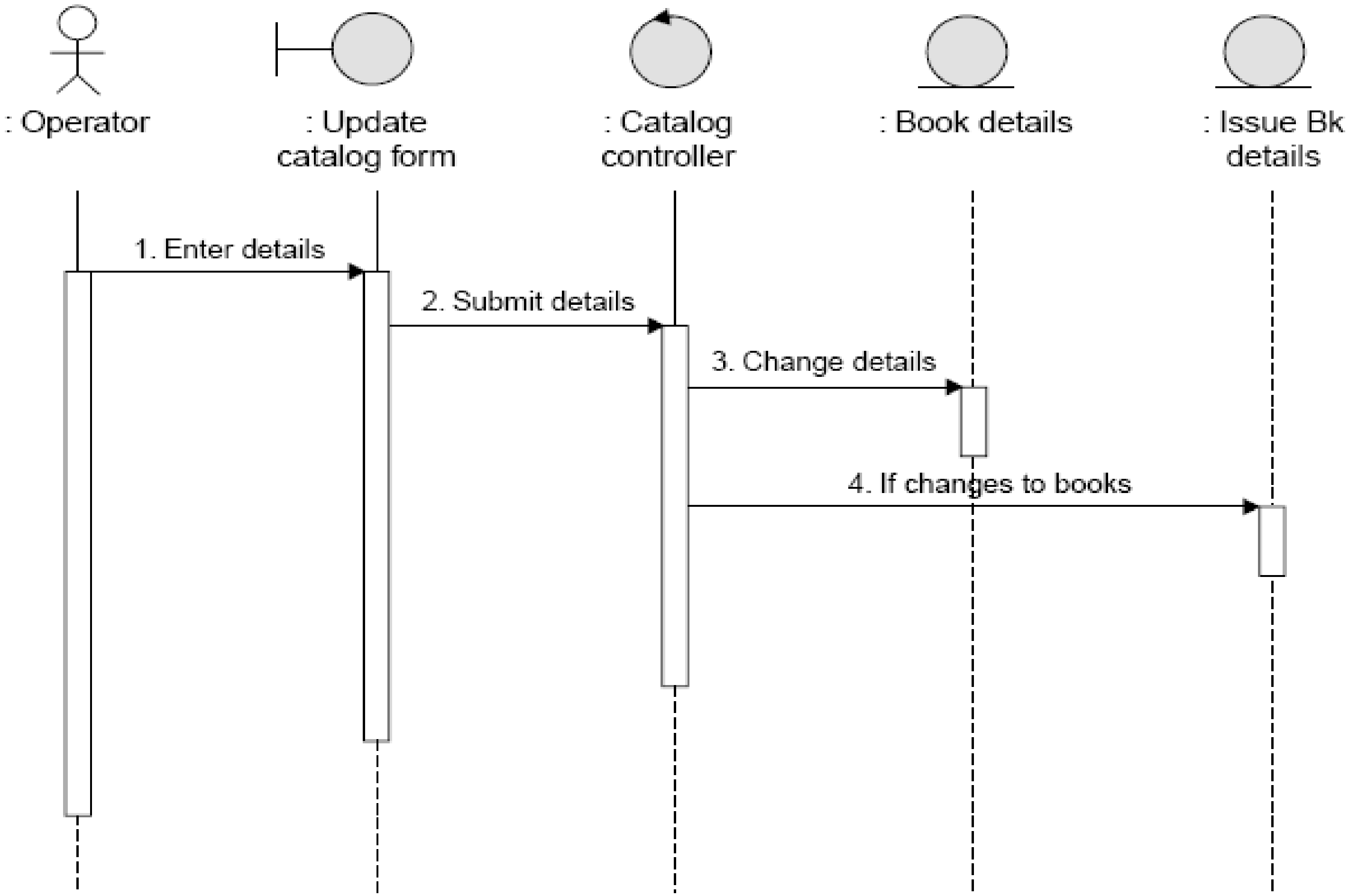
Sequence diagram—return book

Software Design



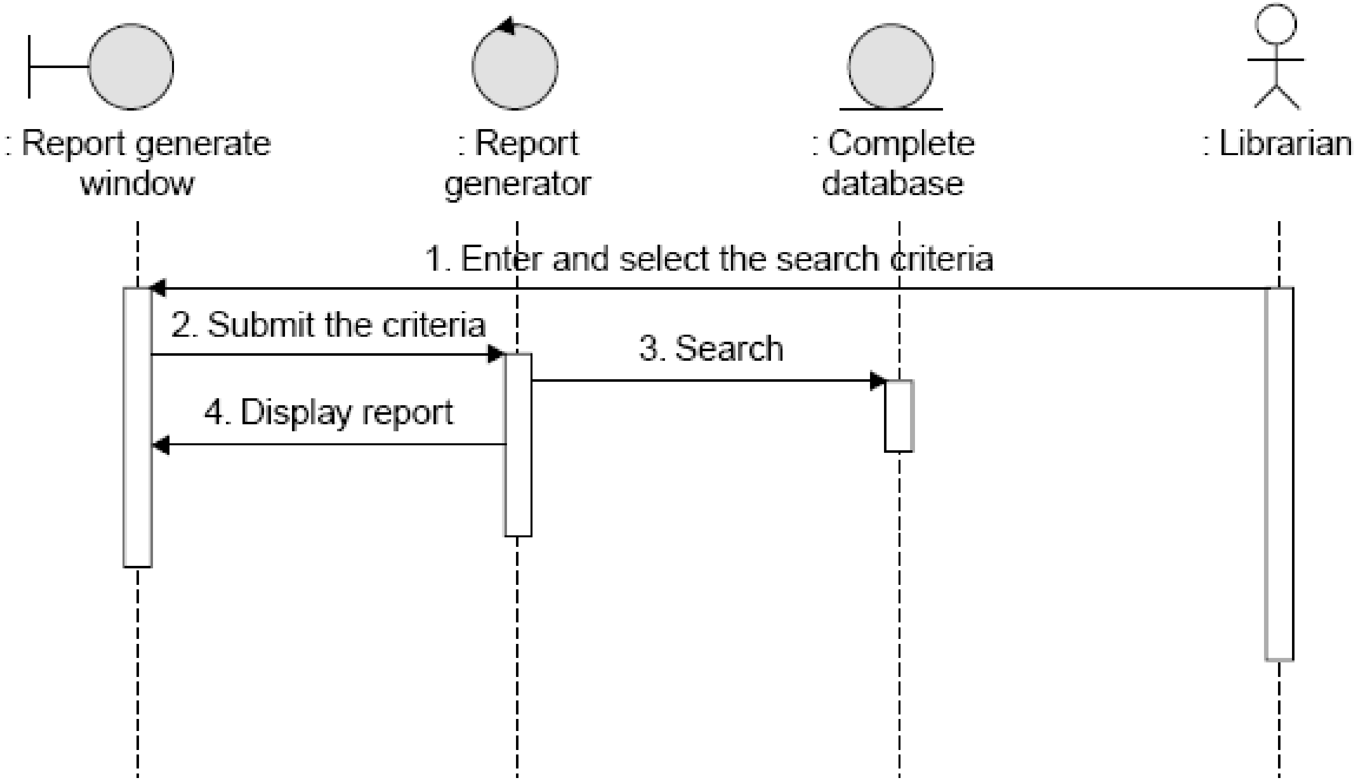
Sequence diagram—query book

Software Design



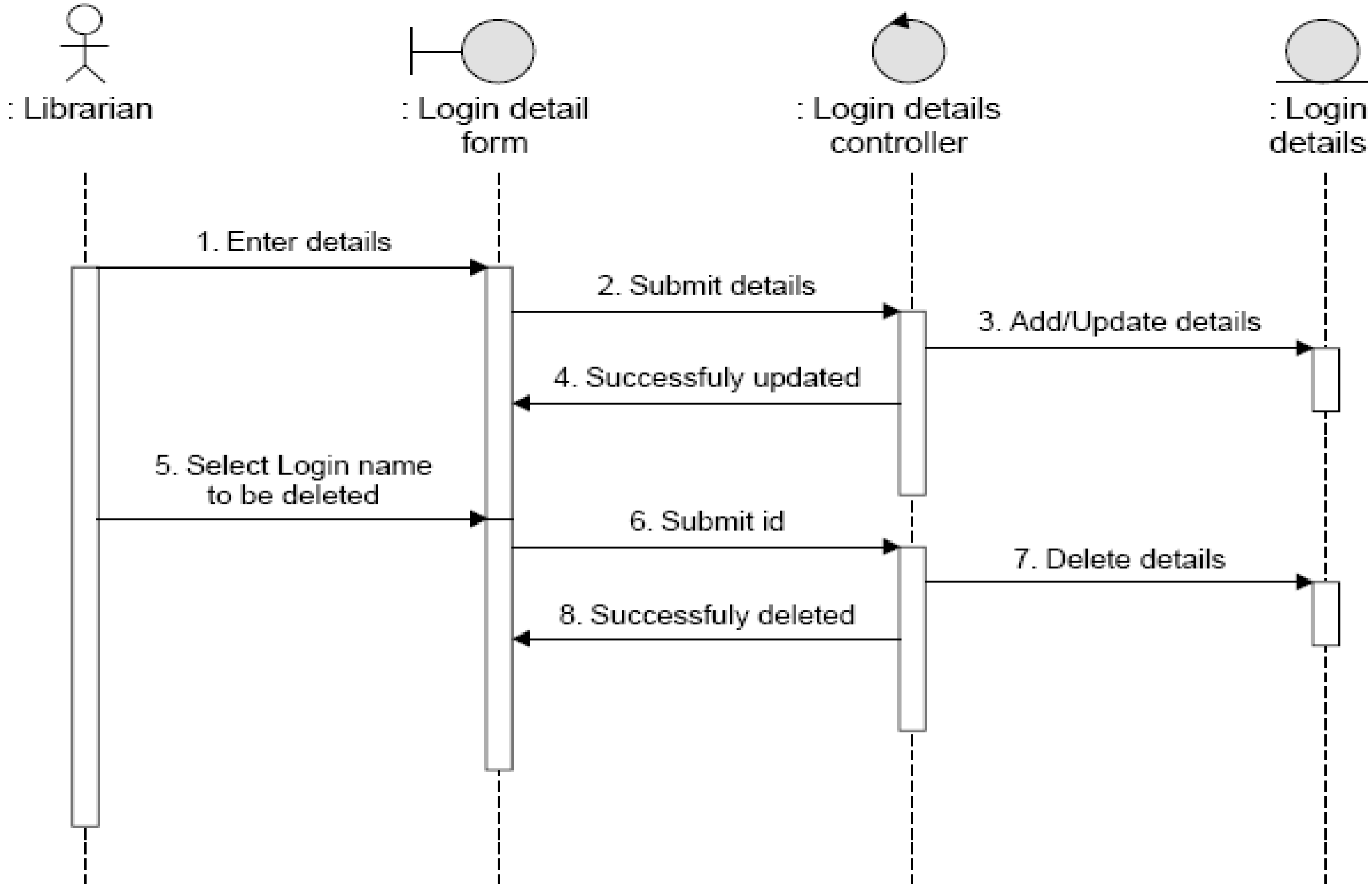
Sequence diagram—maintain catalog

Software Design



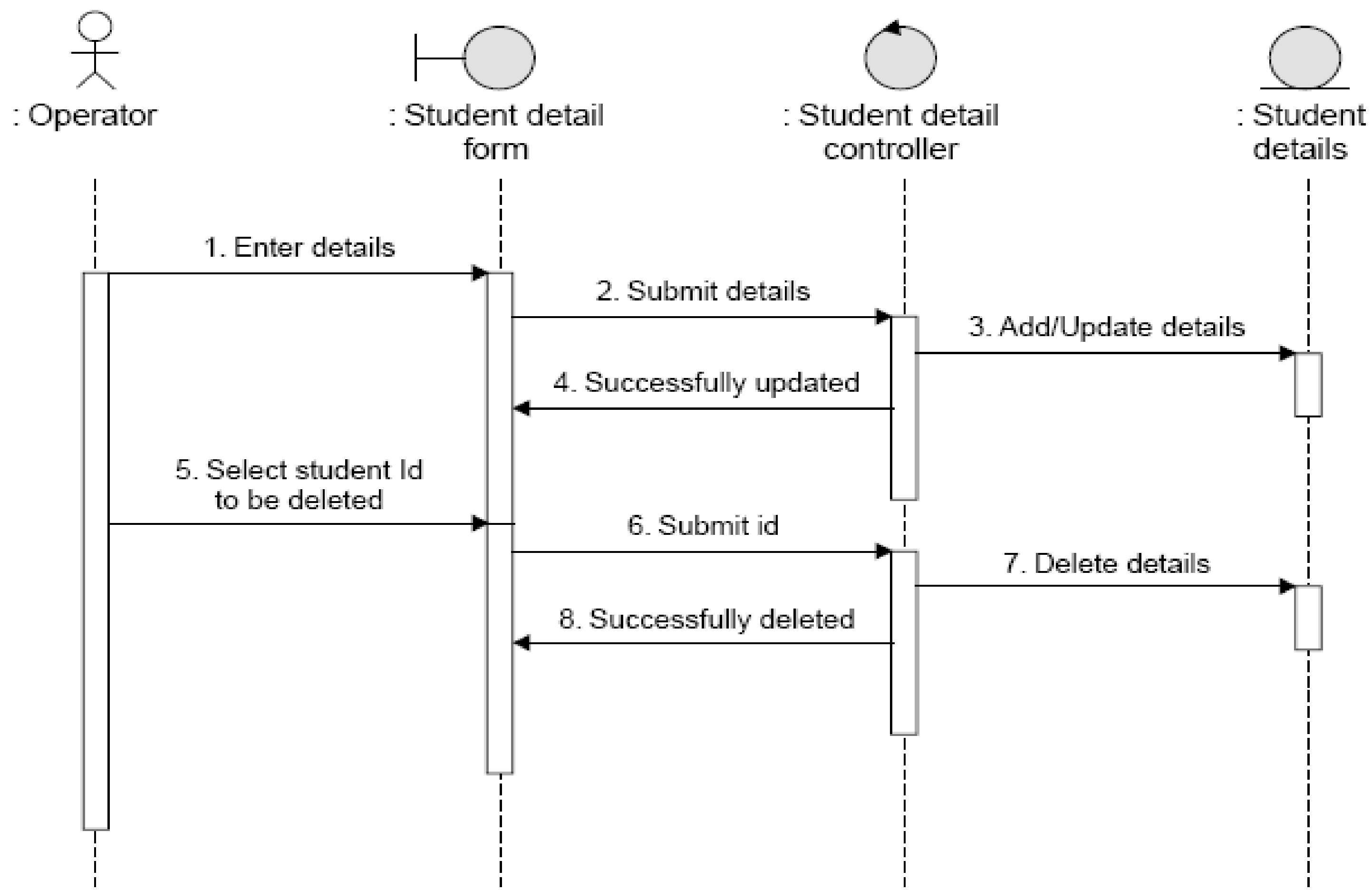
Sequence diagram—generate reports

Software Design



Sequence diagram—maintain login

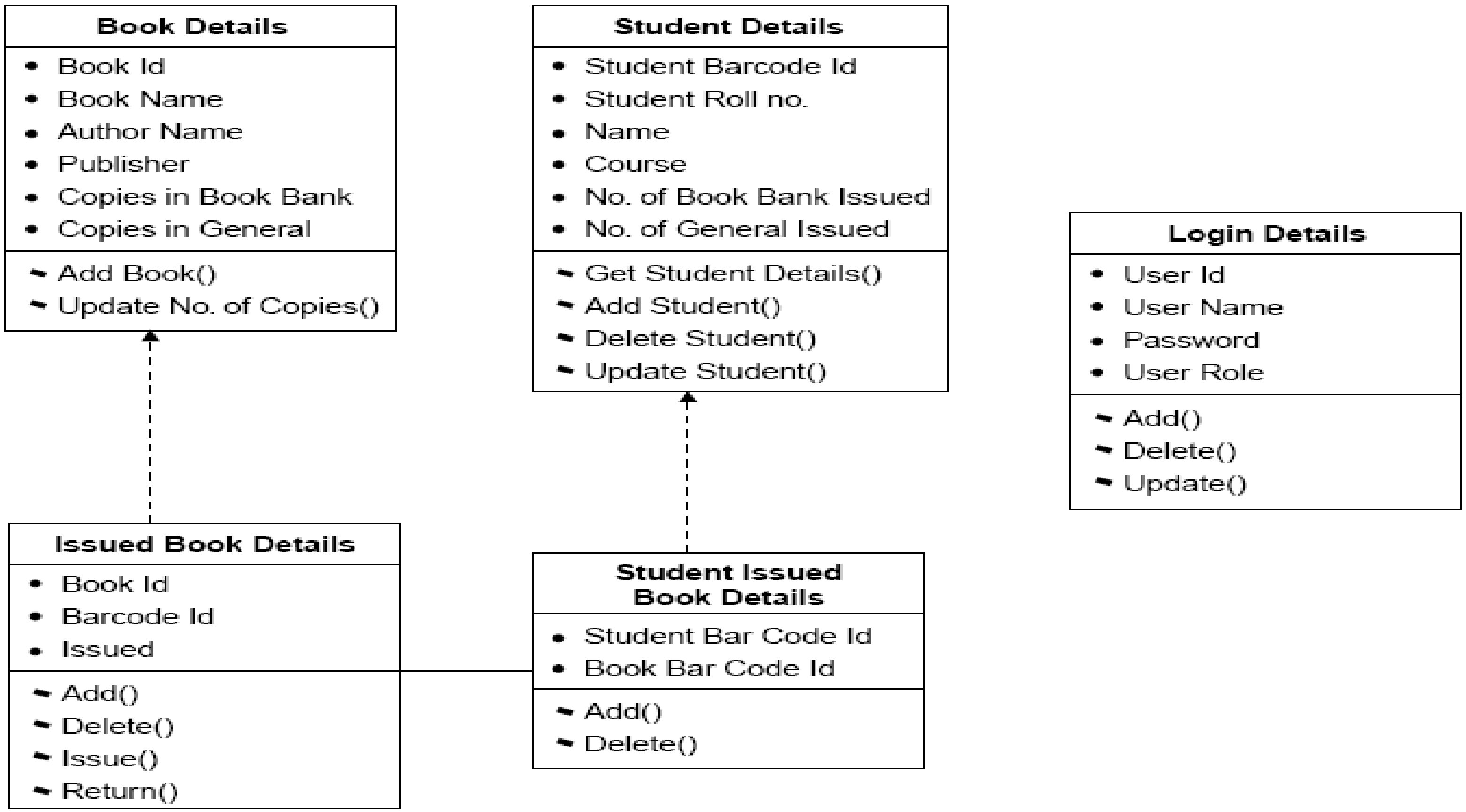
Software Design



Sequence diagram—maintain student details

Software Design

Class diagram of entity classes



Class diagram of entity classes

References

1. K.K Agarwal, Yogesh Singh, "Software Engineering" 3rd edition, New Age International Publishers, 2008
2. Rajib Mall, "Fundamentals of Software Engineering", Prentice-Hall of India.
3. Roger S. Pressman, "Software Engineering: A Practitioner's Approach", Mc Graw Hill Education
4. http://www.tutorialspoint.com