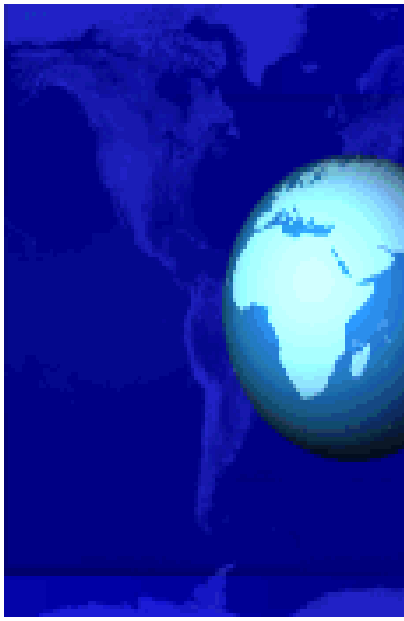
Software Design

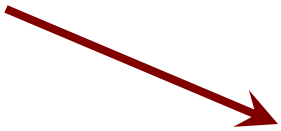


Software Design

- More creative than analysis
- Problem solving activity

WHAT IS DESIGN

'HOW'



Software design document (SDD

Software Design

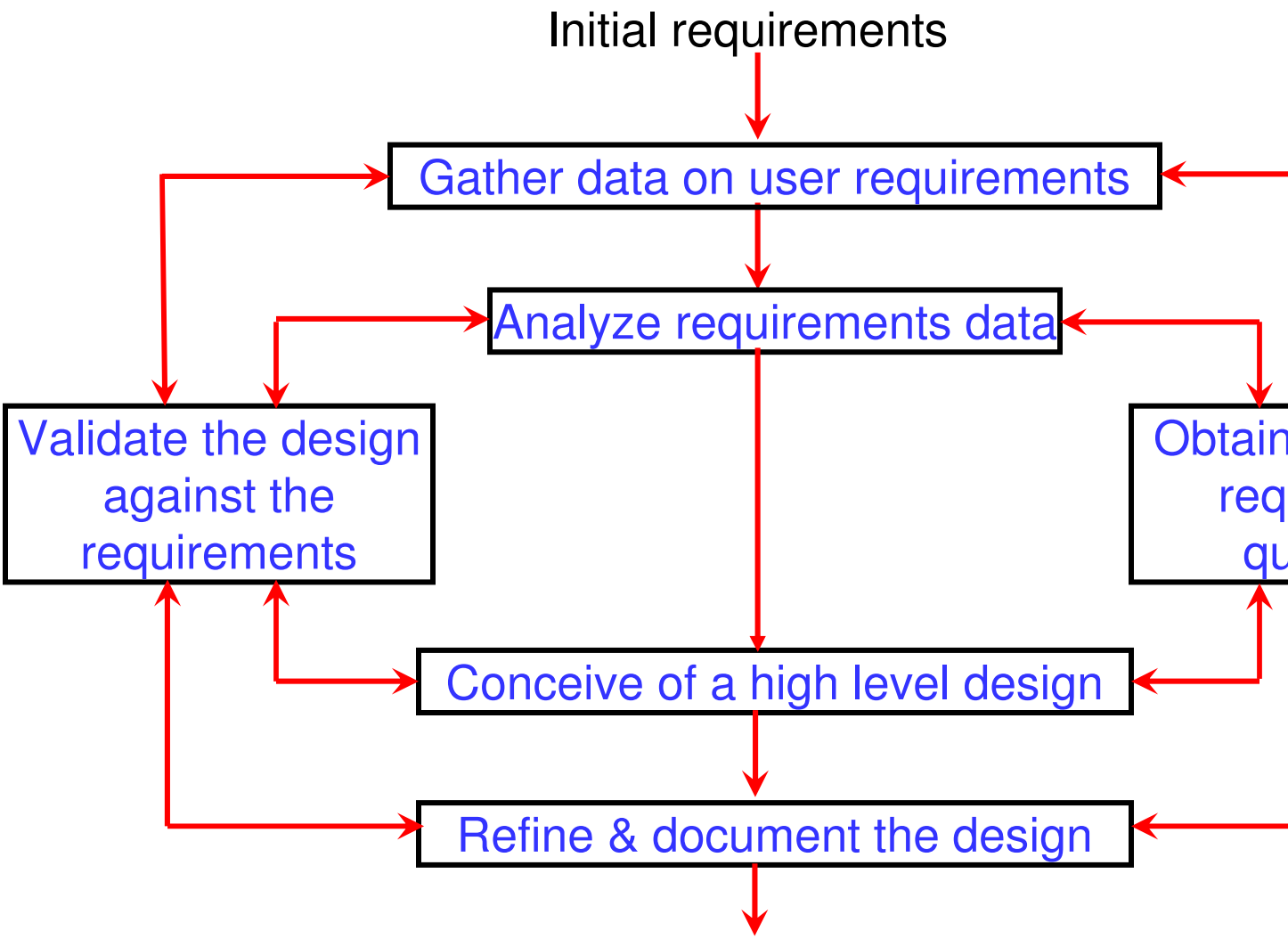
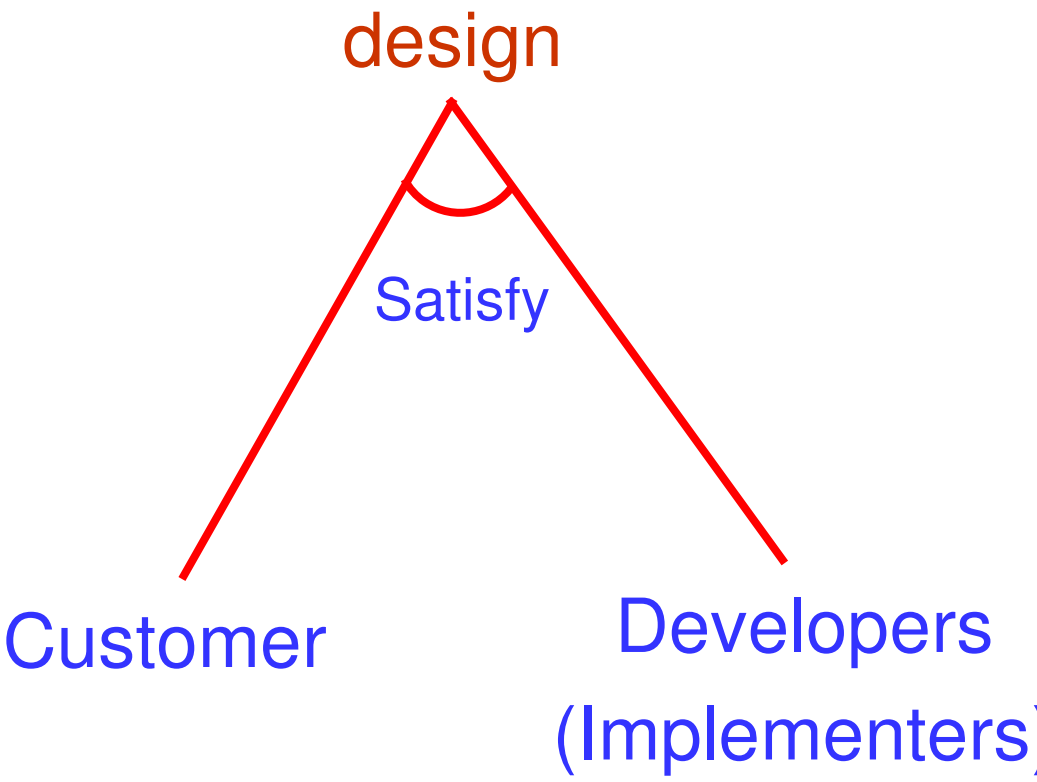


Fig. 1 : Design framework

Software Design



Conceptual Design and Technical Design

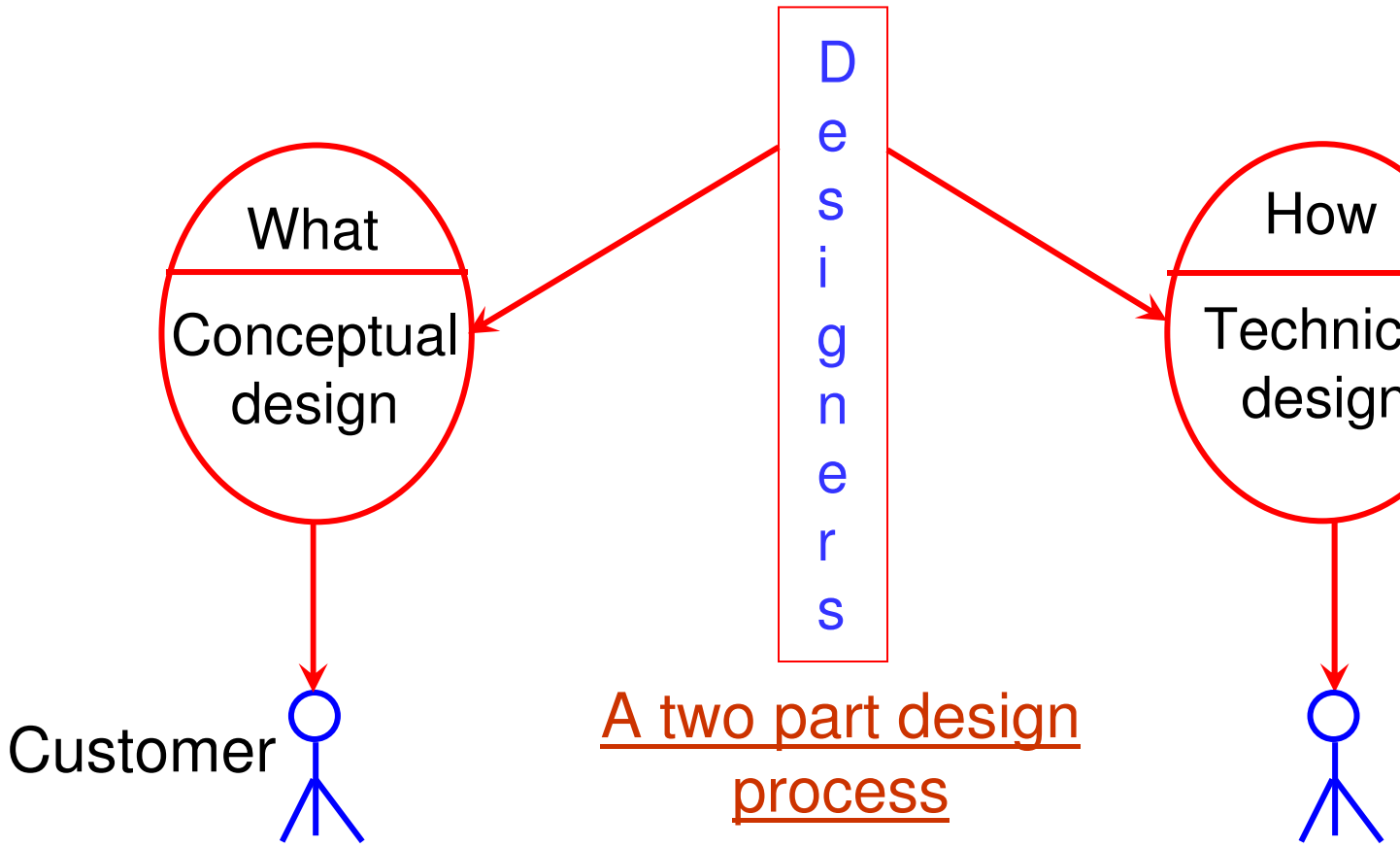


Fig. 2 : A two part design process

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 requirem solution to the customer's problem.

Software Design

The design needs to be

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

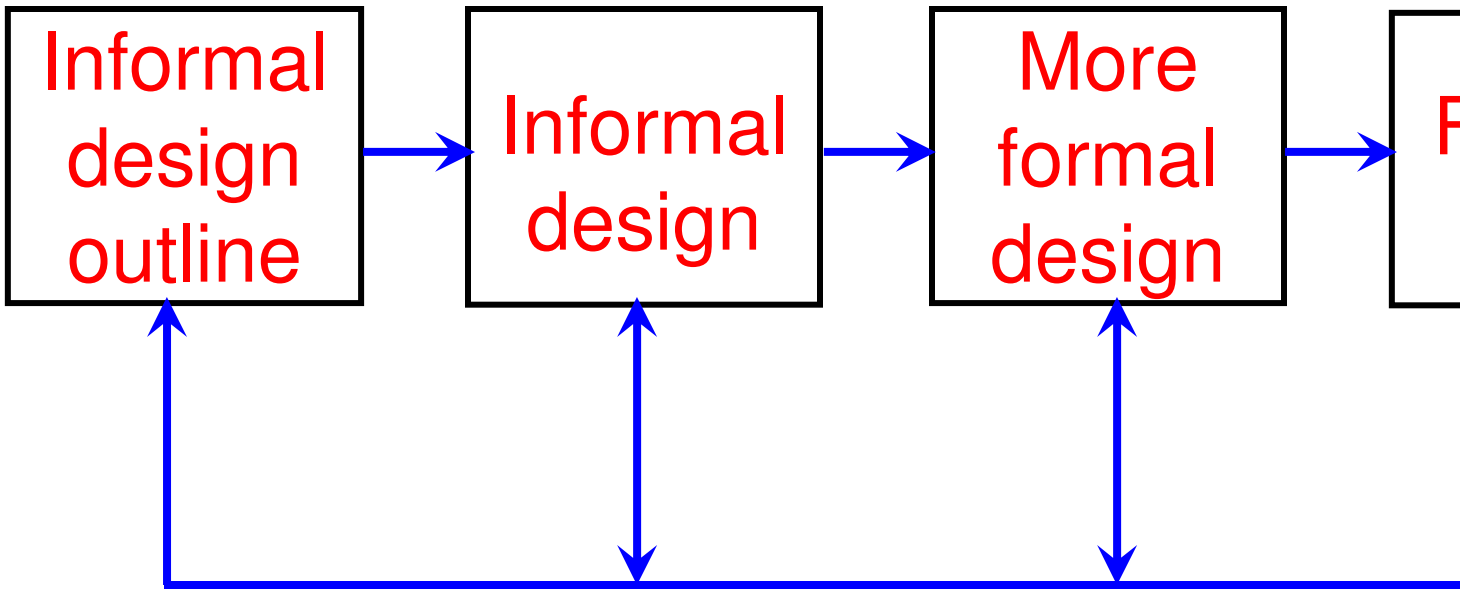


Fig. 3 : The transformation of an informal design to a design.

MODULARITY

There are many definitions of the term module. Rang

i. Fortran subroutine

ii. Ada package

iii. Procedures & functions of PASCAL & C

iv. C++ / Java classes

v. Java packages

vi. Work assignment for an individual program

Software Design

All these definitions are correct. A m
system consist of well defined mana
units with well defined interfaces
the units.

Software Design

Properties :

i. Well defined subsystem

ii. Well defined purpose

iii. Can be separately compiled and stor
library.

iv. Module can use other modules

v. Module should be easier to use than

vi. Simpler from outside than from the i

Software Design

Modularity is the single attribute of softw
allows a program to be intellectually mana

It enhances design clarity, which in tur
implementation, debugging,
documenting, and maintenance of s
product.

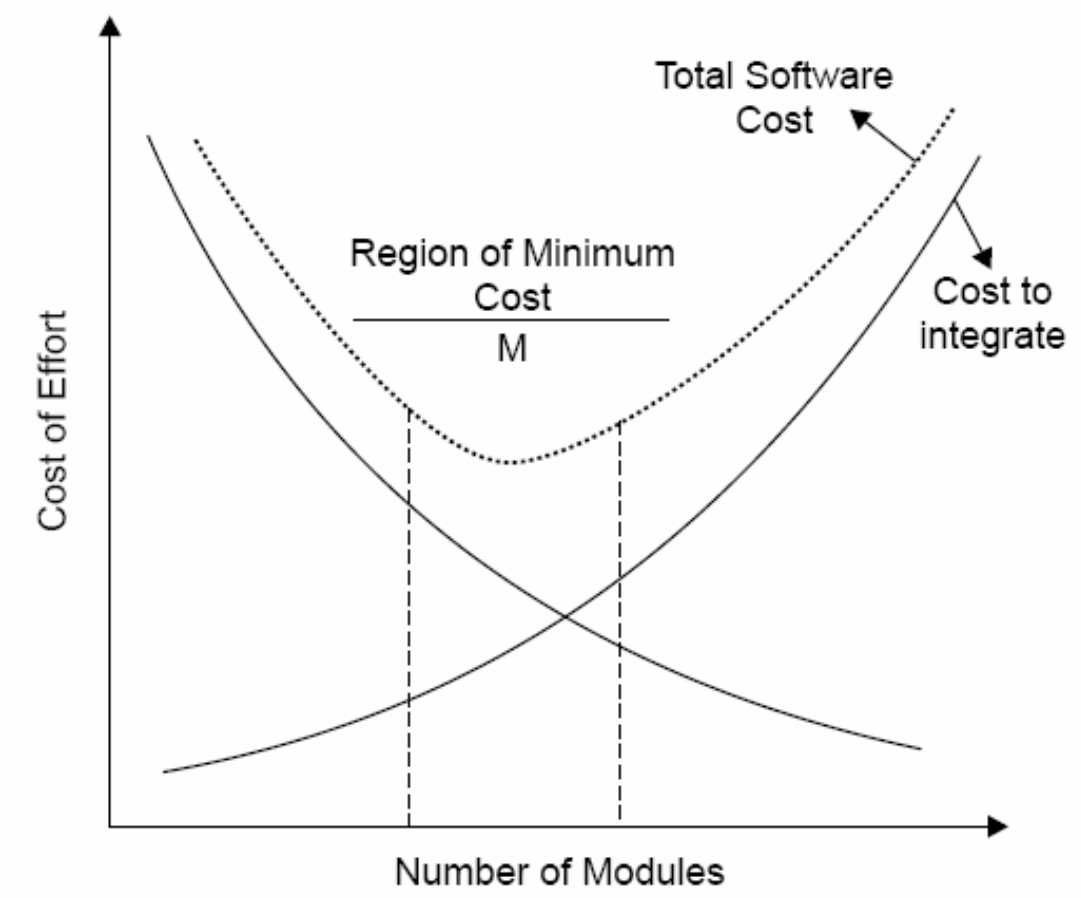
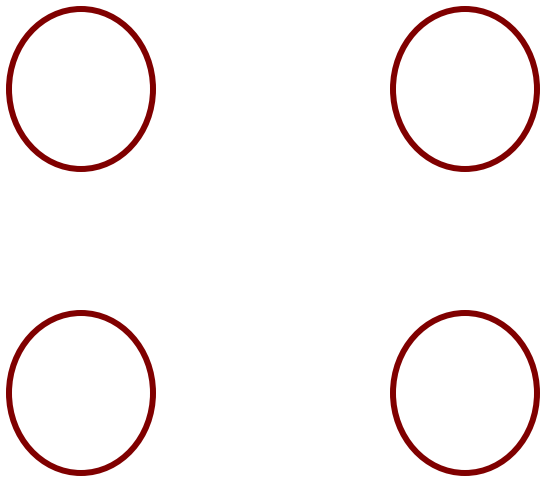


Fig. 4 : Modularity and software cost

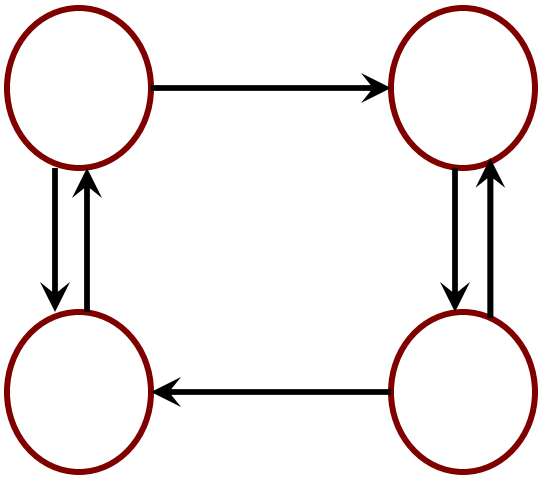
Module Coupling

Coupling is the measure of the de
interdependence between modules.



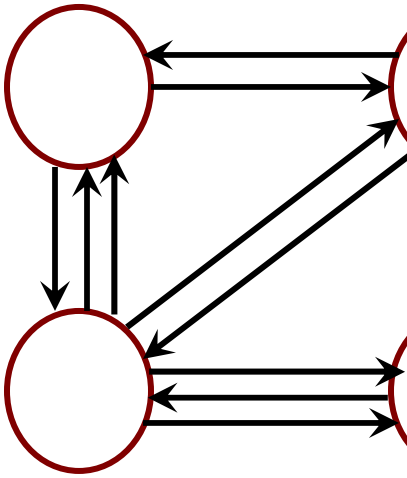
(Uncoupled : no dependencies)

(a)



Loosely coupled:
some dependencies

(B)



Highly coup
many depend

(C)

Fig. 5 : Module coupling

This can be achieved as:

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

Consider the example of editing a student re
'student information system'.

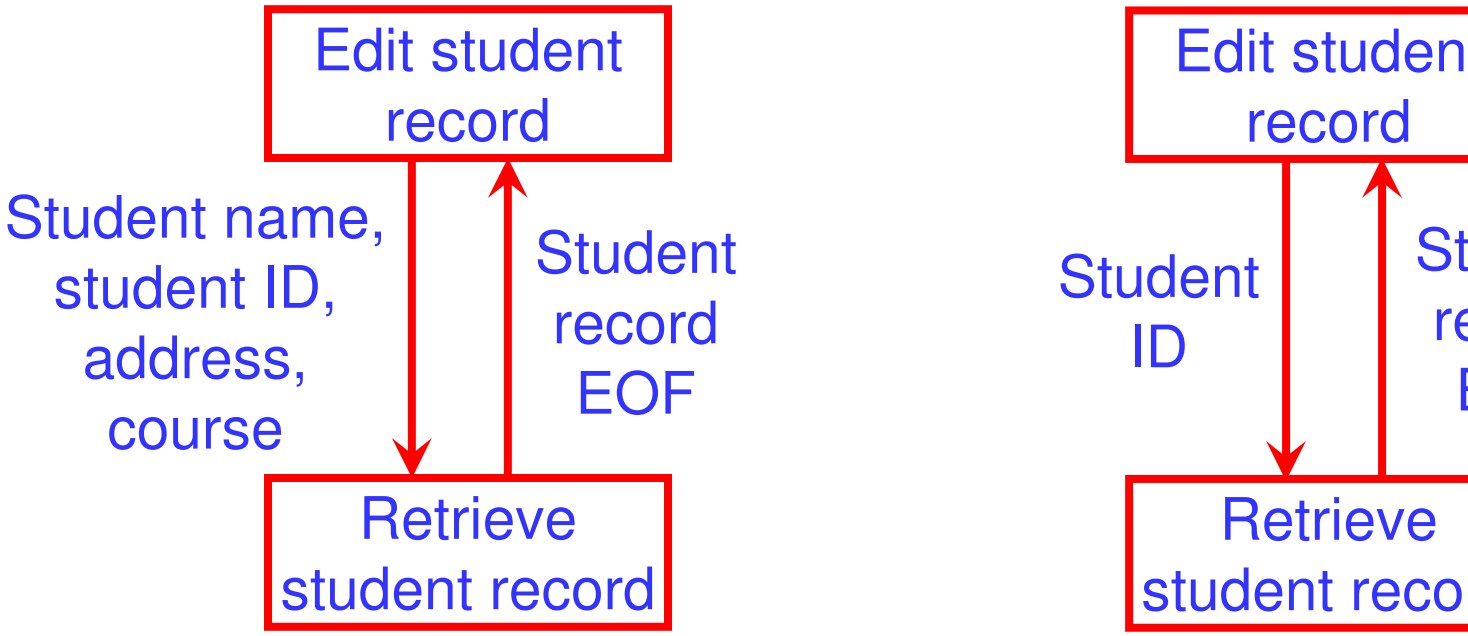


Fig. 6 : Example of coupling

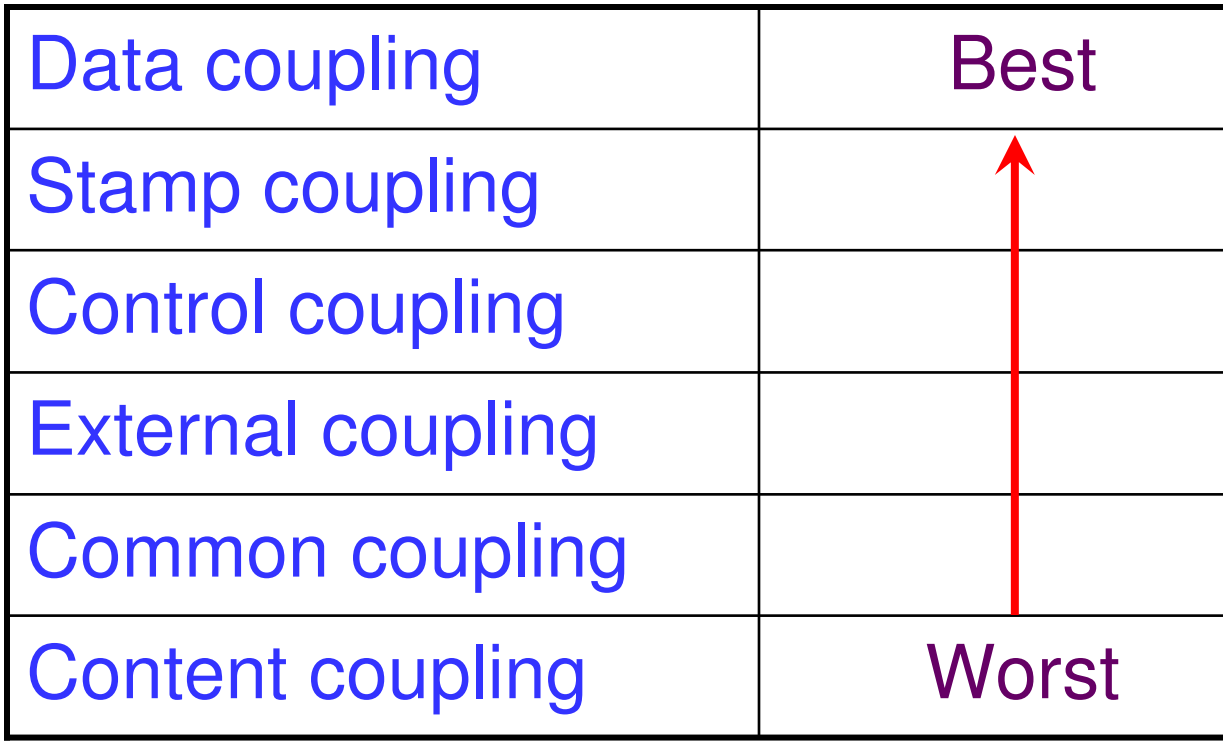


Fig. 7 : The types of module coupling

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

Data coupling

The dependency between module A and B is said coupled if their dependency is based on the communicate by only passing of data. O communicating through data, the two mo independent.

Stamp coupling

Stamp coupling occurs between module A and complete data structure is passed from one module t

Control coupling

Module A and B are said to be control couple
communicate by passing of control information. This
accomplished by means of flags that are set by one m
reacted upon by the dependent module.

Common coupling

With common coupling, module A and module B ha
data. Global data areas are commonly found in pro
languages. Making a change to the common data mea
back to all the modules which access that data to ev
effect of changes.

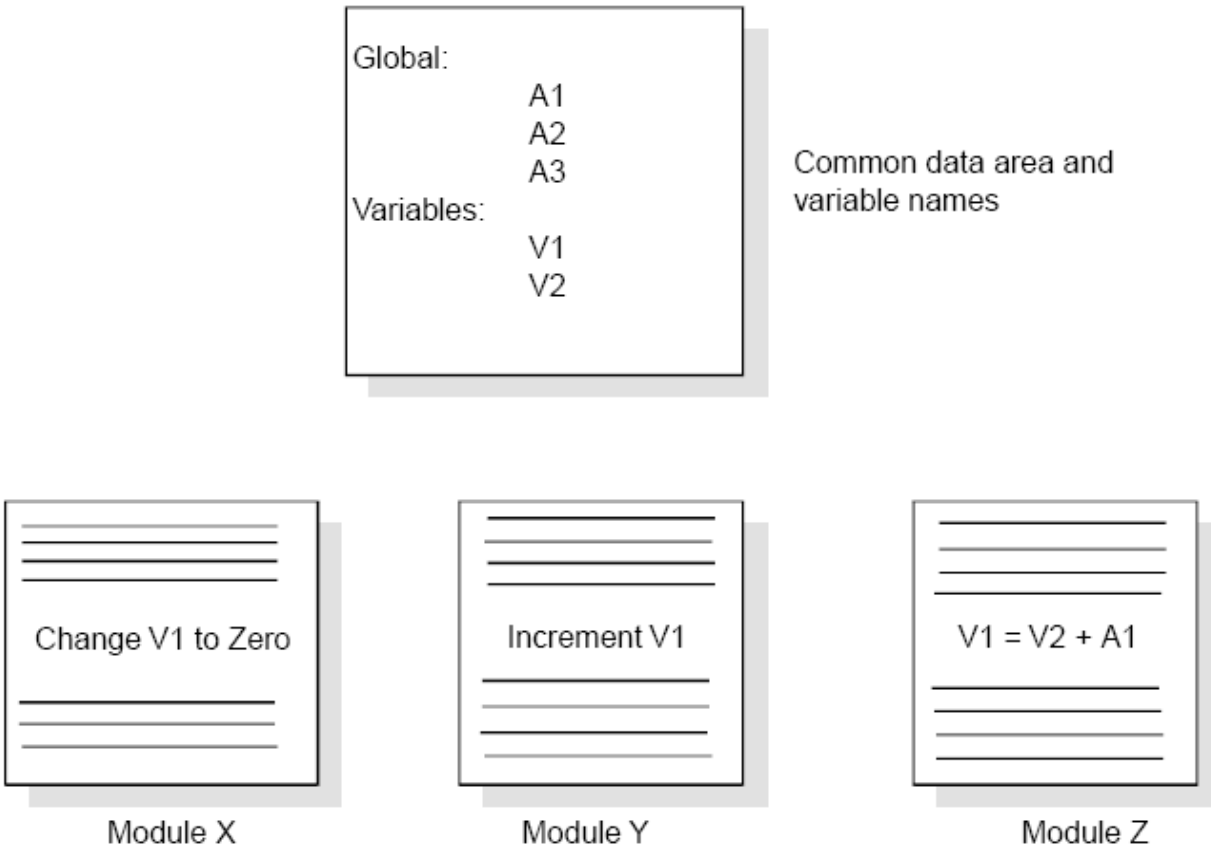


Fig. 8 : Example of common coupling

Content coupling

Content coupling occurs when module A change
module B or when control is passed from one mod
middle of another. In Fig. 9, module B branches int
though D is supposed to be under the control of C.

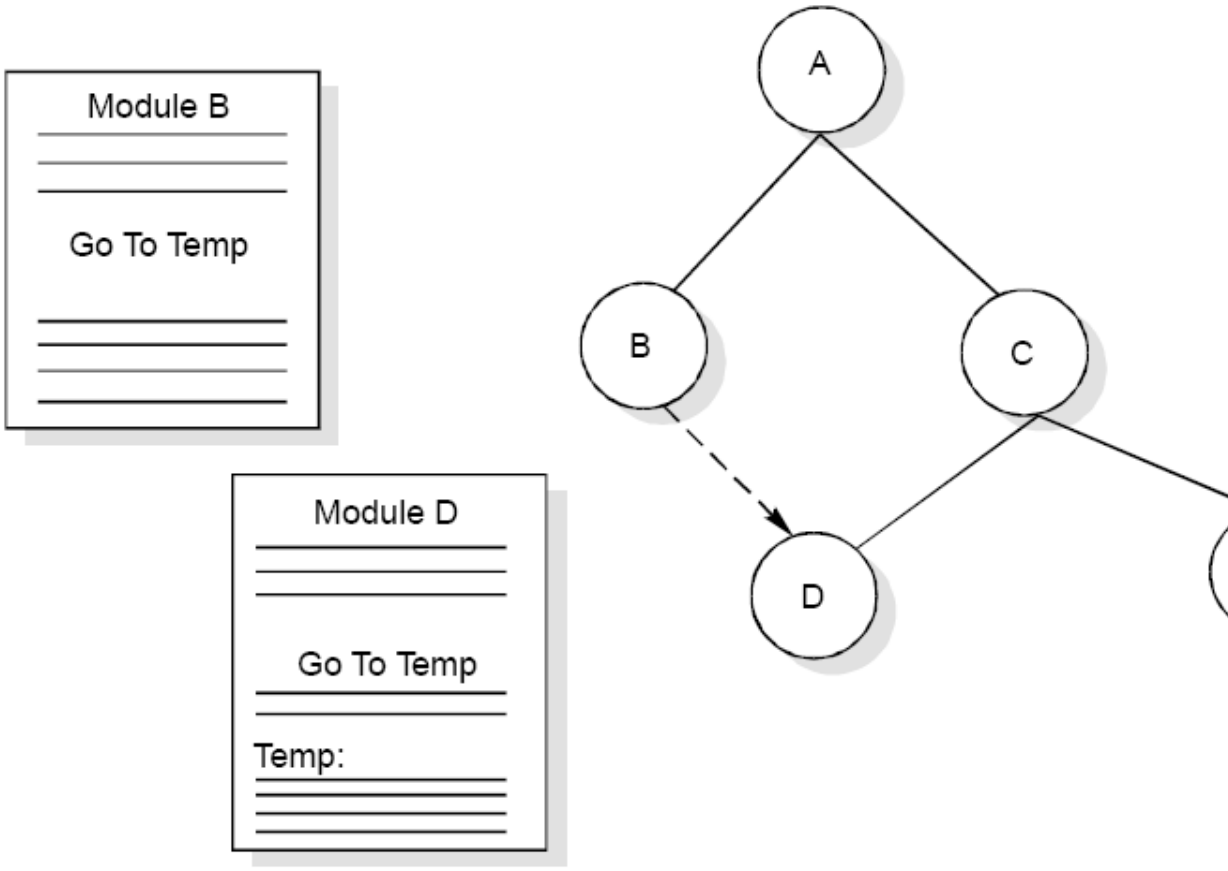


Fig. 9 : Example of content coupling

Module Cohesion

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

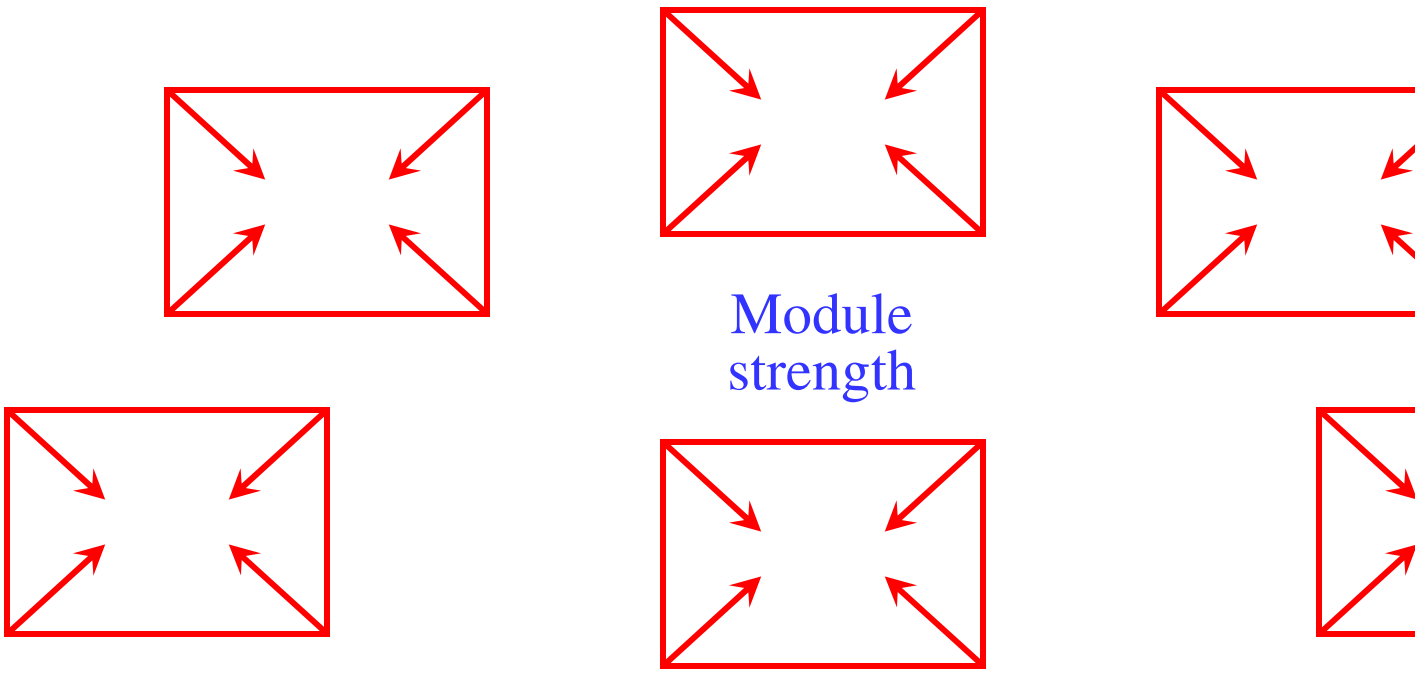


Fig. 10 : Cohesion=Strength of relations within mo

Types of cohesion

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

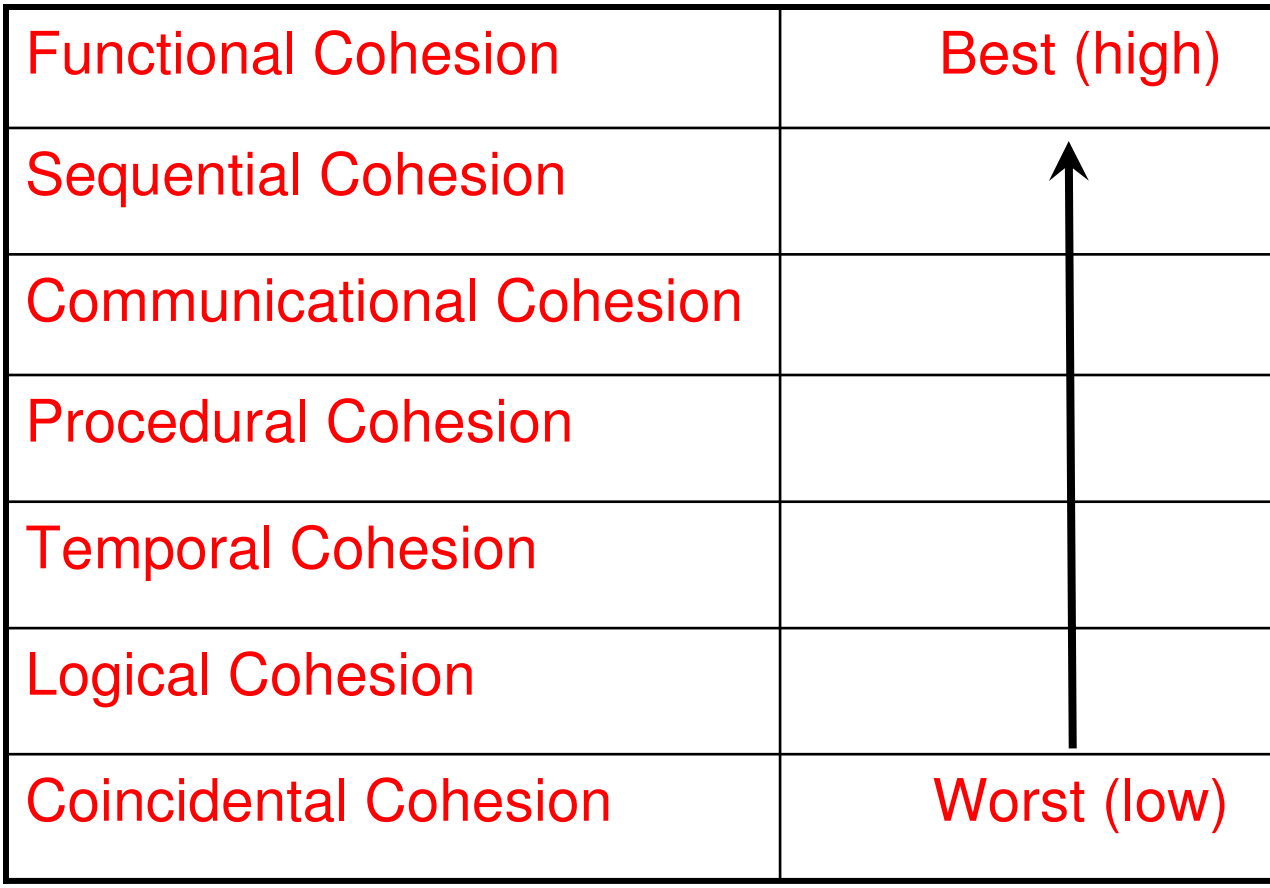


Fig. 11 : Types of module cohesion

Functional Cohesion

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

Sequential Cohesion

- Module A outputs some data which forms the input to the reason for them to be contained in the same proce

Procedural Cohesion

- Procedural Cohesion occurs in modules whose although accomplish different tasks yet have been because there is a specific order in which the tasks completed.

Temporal Cohesion

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

Logical Cohesion

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

Coincidental Cohesion

- Coincidental cohesion exists in modules tha
  instructions that have little or no relationship to one a

Relationship between Cohesion & Coupling

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

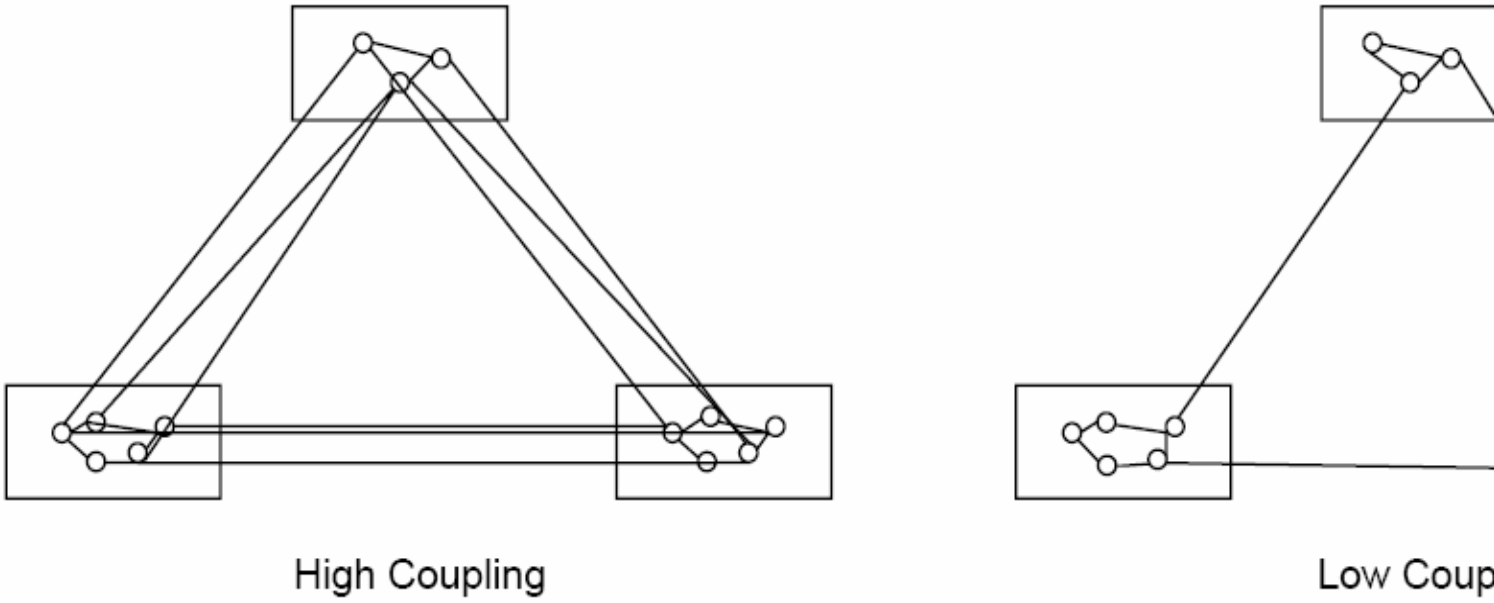


Fig. 12 : View of cohesion and coupling

STRATEGY OF DESIGN

A good system design strategy is to organize the progra
in such a way that are easy to develop and latter
Structured design techniques help developers to deal w
and complexity of programs. Analysts create instruct
developers about how code should be written and ho
code should fit together to form a program. It is impor
reasons:

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

Bottom-Up Design

These modules are collected together in the form of a “lib

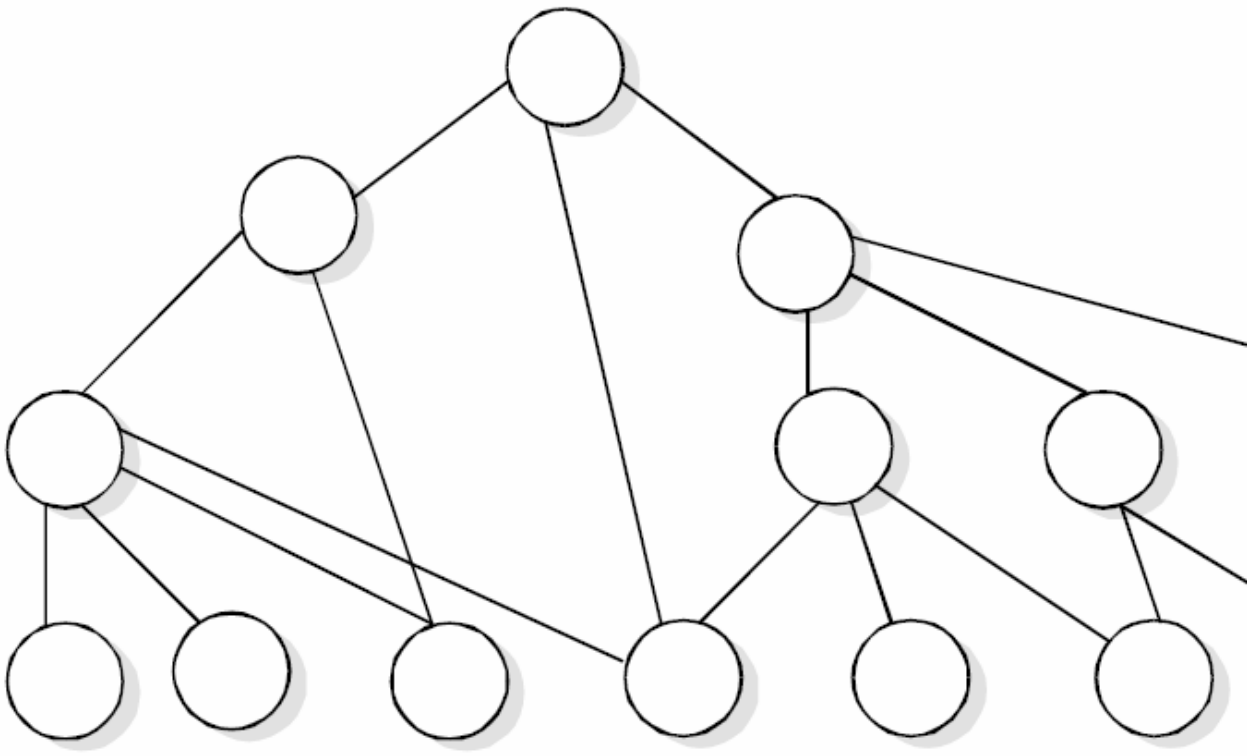


Fig. 13 : Bottom-up tree structure

Top-Down Design

A top down design approach starts by identifying the ma
of the system, decomposing them into their lower level r
iterating until the desired level of detail is achieved. This
refinement; starting from an abstract design, in each ste
is refined to a more concrete level, until we reach a lev
more refinement is needed and the design can be i
directly.

Hybrid Design

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

- To permit common sub modules.
- Near the bottom of the hierarchy, where the intuitior and the need for bottom-up testing is greater, becaus more number of modules at low levels than high level
- In the use of pre-written library modules, in particula modules.

FUNCTION ORIENTED DESIGN

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

Software Design

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

```
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" modul
ate" module and a "print" module. Now we consider the "print" module and is giv

```
        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 t
level of our programming language. At that point, we can
structure of our program as a tree of refinement as in desi
structure as shown in fig. 14.

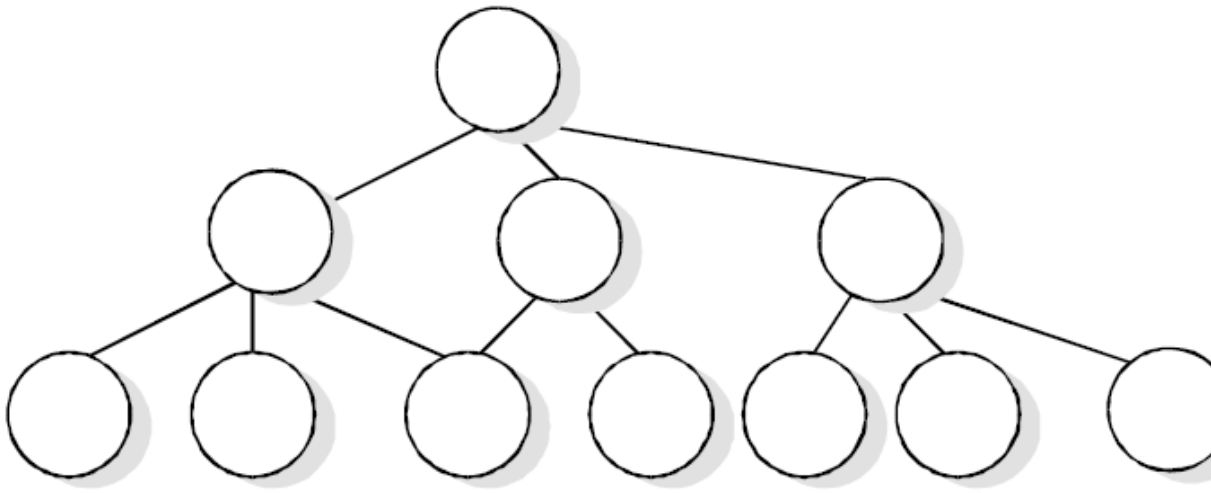


Fig. 14 : Top-down structure

Software Design

If a program is created top-down, the modules become very
As one can easily see in top down design structure, each m
by at most one other module, its parent. For a module, l
must require that several other modules as in design reusa
as shown in fig. 15.

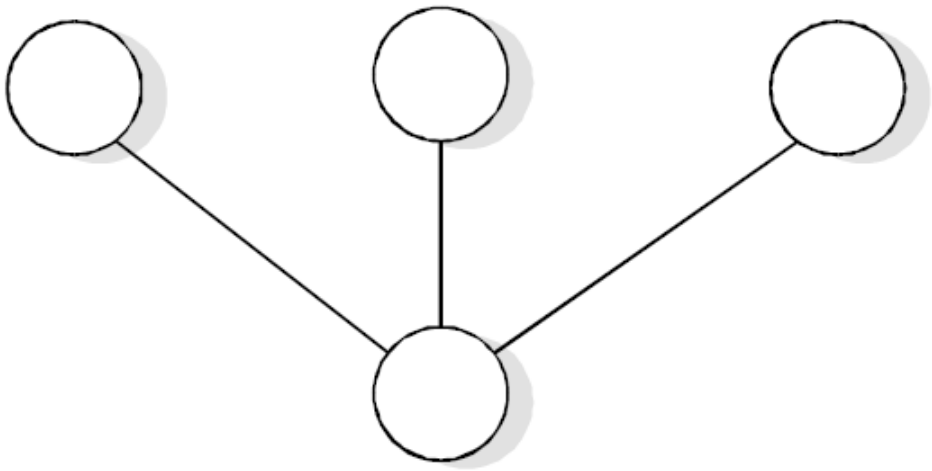


Fig. 15 : Design reusable structure

Design Notations

Design notations are largely meant to be used during
of design and are used to represent design or desig
For a function oriented design, the design can be
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 functionality is known to the user without the knowledg design.

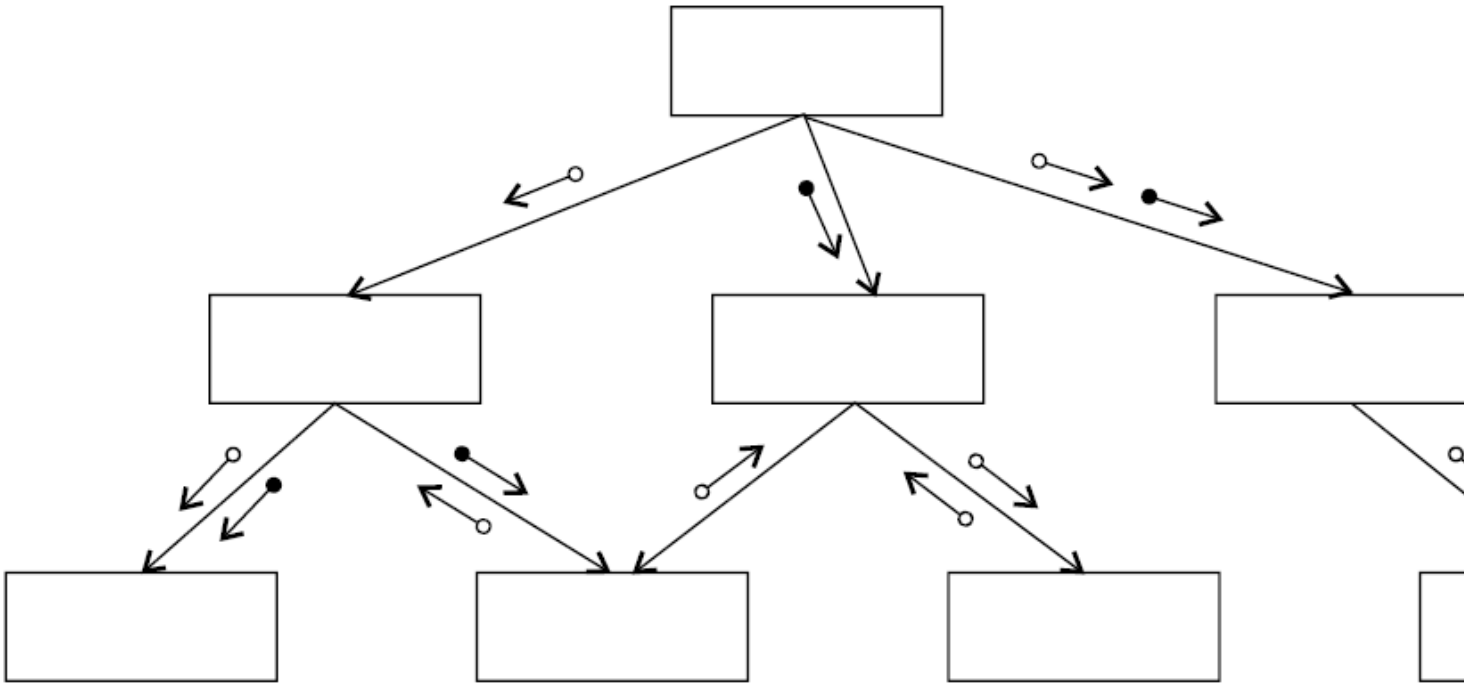


Fig. 16 : Hierarchical format of a structure cha

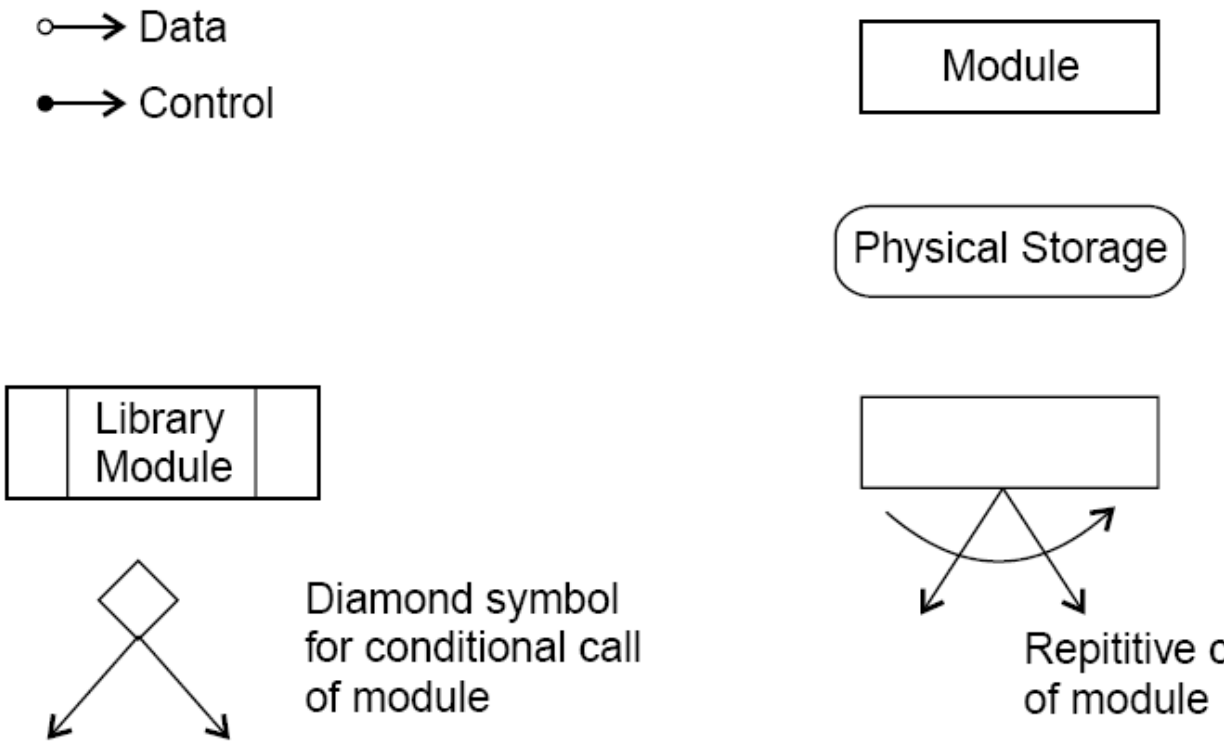


Fig. 17 : Structure chart notations

A structure chart for "update file" is given in fig. 18.

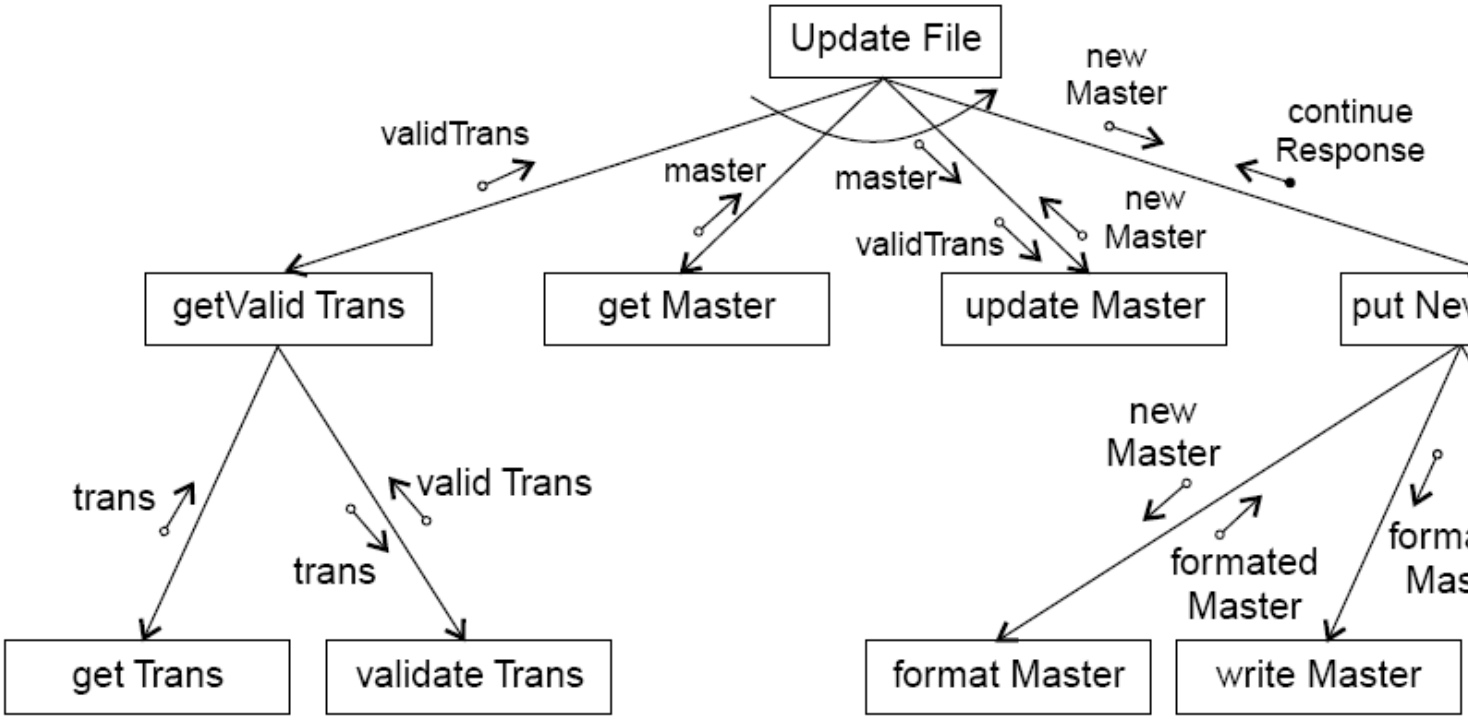


Fig. 18 : Update file

A transaction centered structure describes a system that
number of different types of transactions. It is illustrated i

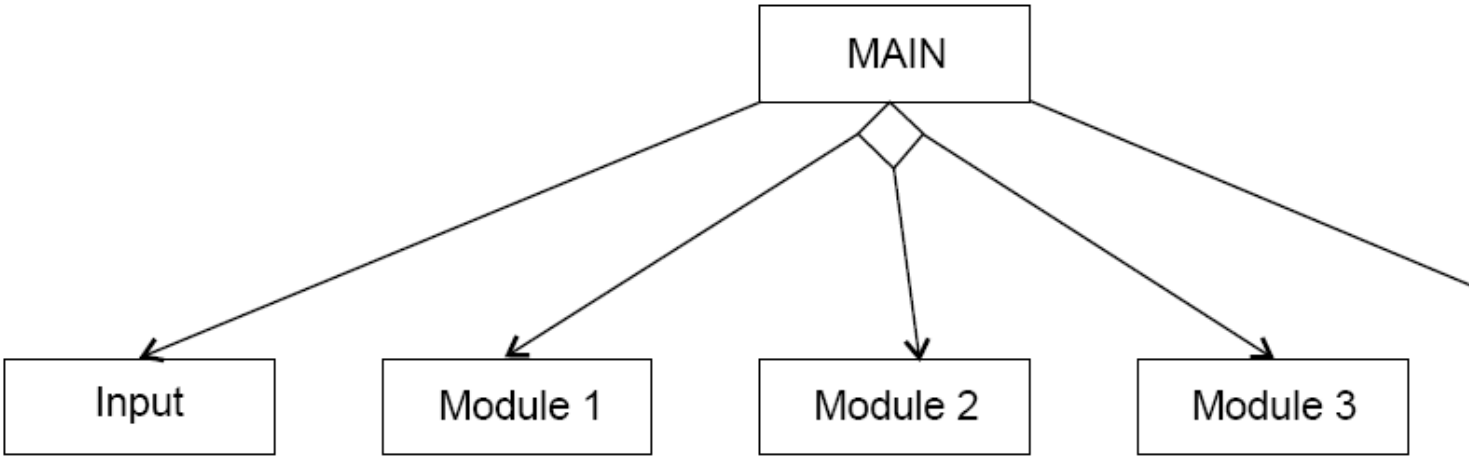


Fig. 19 : Transaction-centered structure

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

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

Pseudocode

Pseudocode notation can be used in both the preliminary
design phases.

Using pseudocode, the designer describes system ch
using short, concise, English language phrases that are s
key words such as It-Then-Else, While-Do, and End.

Functional Procedure Layers

- Function are built in layers, Additional notation specify details.
- Level 0
  - Function or procedure name
  - Relationship to other system components (e.g which system, called by which routines, etc.)
  - Brief description of the function purpose.
  - Author, date

Software Design

- Level 1
  - Function Parameters (problem variables, types, etc.)
  - Global variables (problem variable, type, sharing information)
  - Routines called by the function
  - Side effects
  - Input/Output Assertions

- Level 2
  - Local data structures (variable etc.)
  - Timing constraints
  - Exception handling (conditions, responses, even
  - Any other limitations
- Level 3
  - Body (structured chart, English pseudo code, tables, flow charts, etc.)

IEEE Recommended practice for softwar descriptions (IEEE STD 1016-1998)

- Scope

An SDD is a representation of a software system that is used for communicating software design information.

- References

i. IEEE std 830-1998, IEEE recommended p software requirements specifications.

ii. IEEE std 610.12-1990, IEEE glossary o engineering terminology.

Software Design

- Definitions

i. **Design entity.** An element (Component) of a d structurally and functionally distinct from other e that is separately named and referenced.

ii. **Design View.** A subset of design entity attribute that is specifically suited to the needs of a soft activity.

iii. **Entity attributes.** A named property or charac design entity. It provides a statement of fact abou

iv. **Software design description (SDD).** A repres software system created to facilitate analysi implementation and decision making.

Software Design

- Purpose of an SDD

The SDD shows how the software system will be s satisfy the requirements identified in the SRS. It is b translation of requirements into a description of th structure, software components, interfaces, and data ne the implementation phase. Hence, SDD becomes the b the implementation activity.

- Design Description Information Content
  - Introduction
  - Design entities
  - Design entity attributes

Software Design

The attributes and associated information items are de
following subsections:

a) Identification

b) Type

c) Purpose

d) Function

e) Subordinates

f) Dependenci

g) Interface

h) Resources

i) Processing

j) Data

➢ Design Description Organization

Each design description writer may have a different v
are considered the essential aspects of a software
organization of SDD is given in table 1. This is one of
ways to organize and format the SDD.

A recommended organization of the SDD into sepa
views to facilitate information access and assimilation
table 2.

Software Design

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions and acronyms
2. References
3. Decomposition description
   - 3.1 Module decomposition
     - 3.1.1 Module 1 description
     - 3.1.2 Module 2 description
   - 3.2 Concurrent Process decompostion
     - 3.2.1 Process 1 description
     - 3.2.2 Process 2 description
   - 3.3 Data decomposition
     - 3.3.1 Data entity 1 description
     - 3.3.2 Data entity 2 description

Software Design

4. Dependency description
   - 4.1 Intermodule dependencies
   - 4.2 Interprocess dependencies
   - 4.3 Data dependencies
5. Interface description
   - 5.1 Module Interface
     - 5.1.1 Module 1 description
     - 5.1.2 Module 2 description
   - 5.2 Process interface
     - 5.2.1 Process 1 description
     - 5.2.2 Process 2 description
6. Detailed design
   - 6.1 Module detailed design
     - 6.1.1 Module 1 detail
     - 6.1.2 Module 2 detail
   - 6.2 Data detailed design
     - 6.2.1 Data entry 1 detail
     - 6.2.2 Data entry 2 detail

Ta
Organ
S

Design View	Scope	Entity attribute	re
Decomposition description	Partition of the system into design entities	Identification, type purpose, function, subordinate	Hierarc decomp natural
Dependency description	Description of relationships among entities of system resources	Identification, type, purpose, dependencies, resources	Structu flow di transac
Interface description	List of everything a designer, developer, tester needs to know to use design entities that make up the system	Identification, function, interfaces	Interfa parame
Detail description	Description of the internal design details of an entity	Identification, processing, data	Flow c

Table 2: Design views

Object Oriented Design

Object oriented design is the result of focusing attentior function performed by the program, but instead on the d to do manipulated by the program. Thus, it is orthogona oriented design.

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

➢ Basic Concepts

Object Oriented Design is not dependent on a implementation language. Problems are modeled us Objects have:

- Behavior (they do things)
- State (which changes when they do things)

The various terms related to object design are:

i. Objects

The word “Object” is used very frequently and conve
meaning in different circumstances. Here, meaning is an e
save a state (information) and which offers a number o
(behavior) to either examine or affect this state. A
characterized by number of operations and a state which
the effect of these operations.

ii. Messages

Objects communicate by message passing. Messages c
identity of the target object, the name of the requested o
any other operation needed to perform the function. Messa
implemented as procedure or function calls.

iii. Abstraction

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

iv. Class

In any system, there shall be number of objects. Some o
may have common characteristics and we can group
according to these characteristics. This type of grouping is
class. Hence, a class is a set of objects that share a comm
and a common behavior.

We may define a class “car” and each object that repr
becomes an instance of this class. In this class “car”, In
Maruti, Indigo are instances of this class as shown in fig. 2

Classes are useful because they act as a blueprint for o
want a new square we may use the square class and sim
particular details (i.e. colour and position) fig. 21 shows
represent the square class.

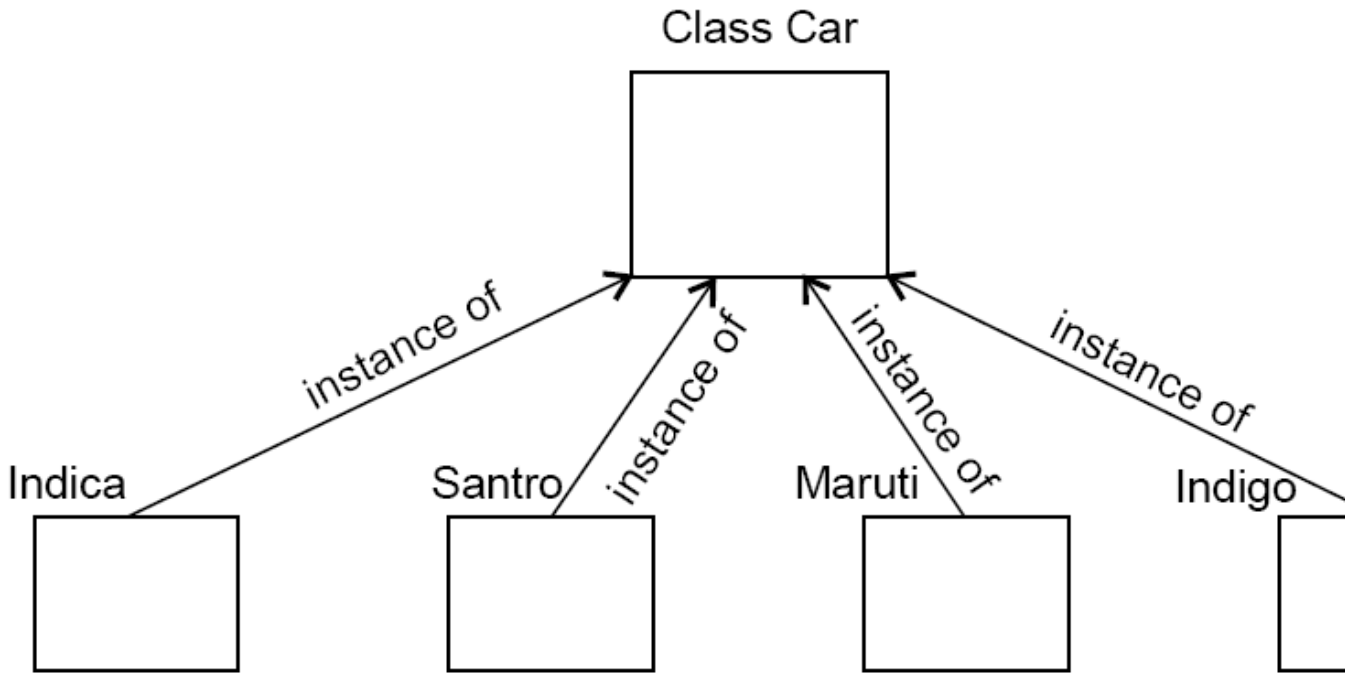


Fig.20: Indica, Santro, Maruti, Indigo are all instances of the

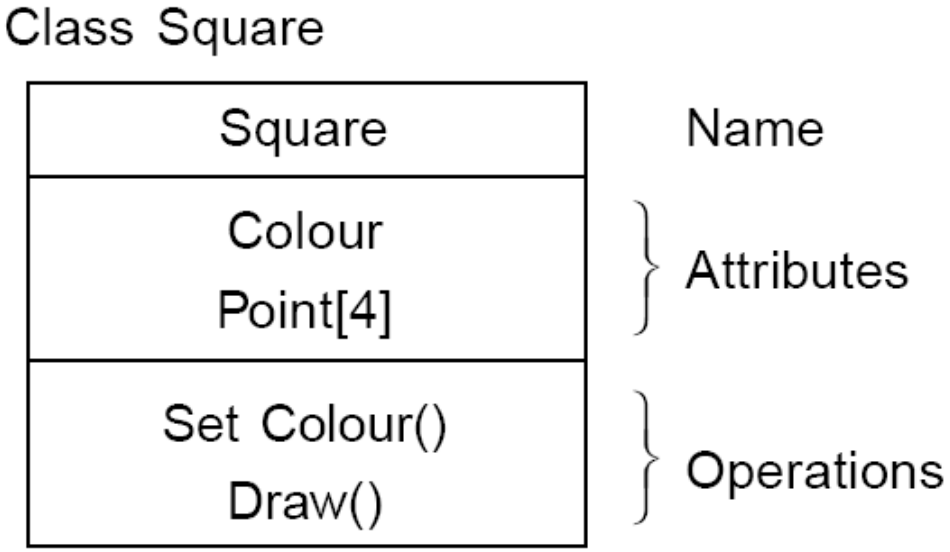


Fig. 21: The square class

v. Attributes

An attributes is a data value held by the objects in a class
class has two attributes: a colour and array of points. Ea
has a value for each object instance. The attributes ar
second part of the class as shown in fig. 21.

vi. Operations

An operation is a function or transformation that may be
by objects in a class. In the square class, we have two op
colour() and draw(). All objects in a class share the same
An object "knows" its class, and hence the right implemen
operation. Operation are shown in the third part of t
indicated in fig. 21.

vii. Inheritance

Imagine that, as well as squares, we have triangle class. F
the class for a triangle.

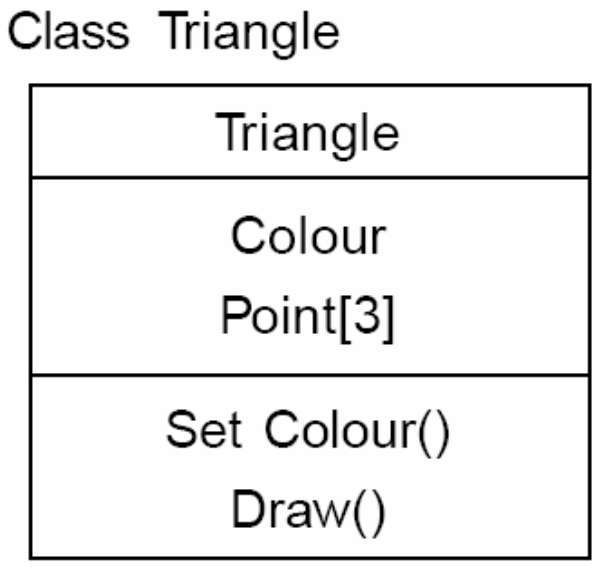


Fig. 22: The triangle class

Now, comparing fig. 21 and 22, we can see that the difference between triangle and squares classes.

For example, at a high level of abstraction, we might want picture as made up of shapes and to draw the picture, w shape in turn. We want to eliminate the irrelevant details care that one shape is a square and the other is a triangl both can draw themselves.

To do this, we consider the important parts out of these cl new class called Shape. Fig. 23 shows the results.

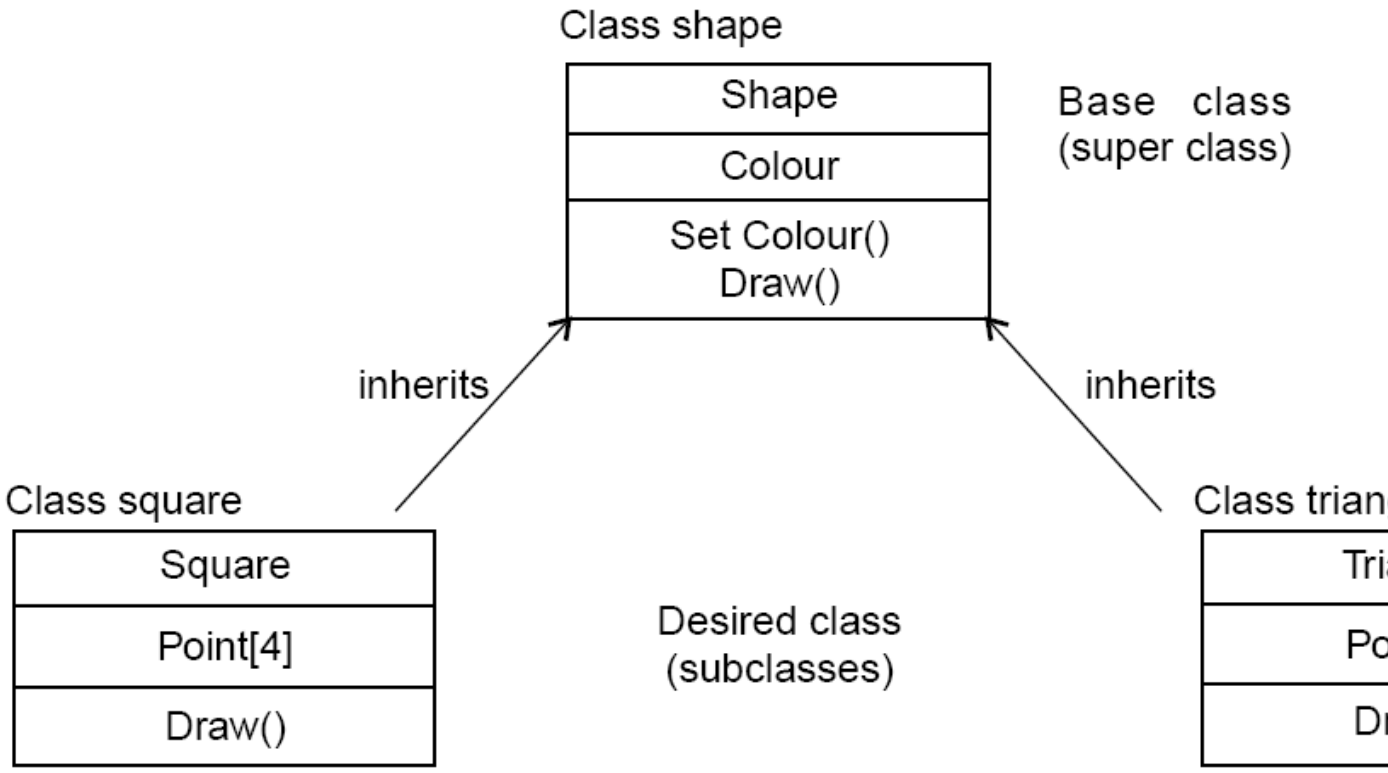


Fig. 23: Abstracting common features in a new clas

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

viii. Polymorphism

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

ix. Encapsulation (Information Hiding)

Encapsulation is also commonly referred to as “Informatio
consists of the separation of the external aspects of an ob
internal implementation details of the object.

x. Hierarchy

Hierarchy involves organizing something according to son
order or rank. It is another mechanism for reducing the c
software by being able to treat and express sub-types in a

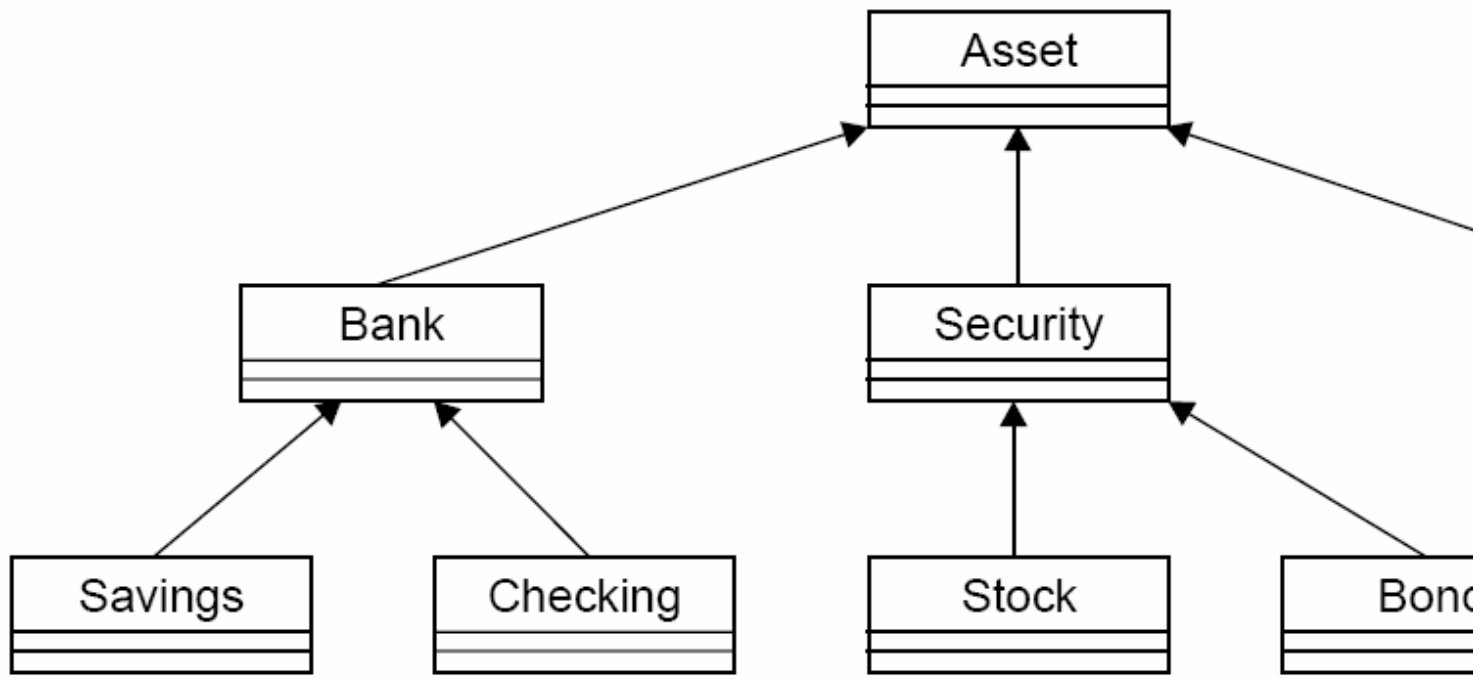


Fig. 24: Hierarchy

- Steps to Analyze and Design Object Oriented Syst

There are various steps in the analysis and design o
oriented system and are given in fig. 25

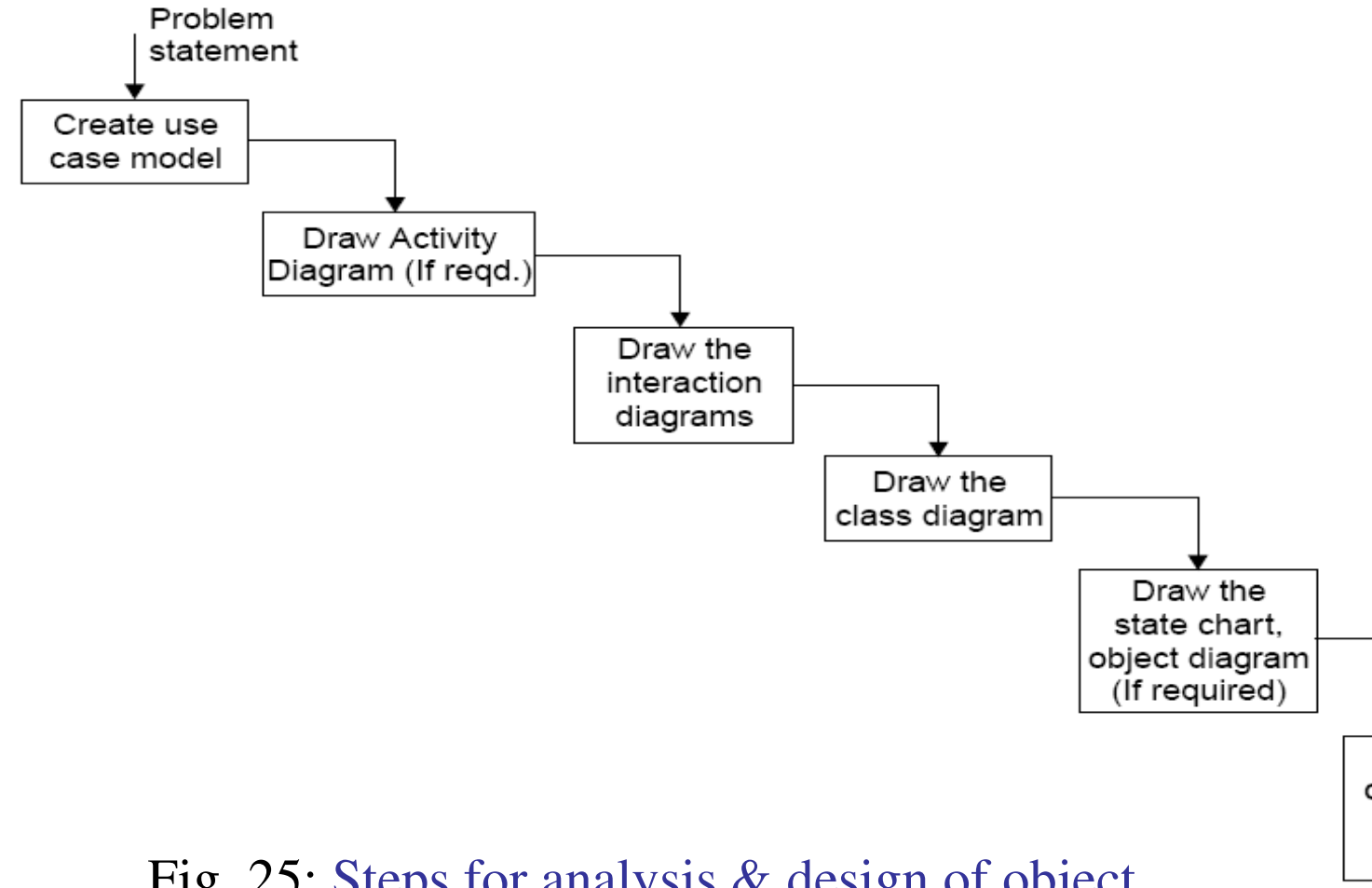


Fig. 25: Steps for analysis & design of object oriented system

i. Create use case model

First step is to identify the actors interacting with the s
should then write the use case and draw the use case diagr

ii. Draw activity diagram (If required)

Activity Diagram illustrate the dynamic nature of a system
the flow of control form activity to activity. An activity re
operation on some class in the system that results in a c
state of the system. Fig. 26 shows the activity diagram pr
order to deliver some goods.

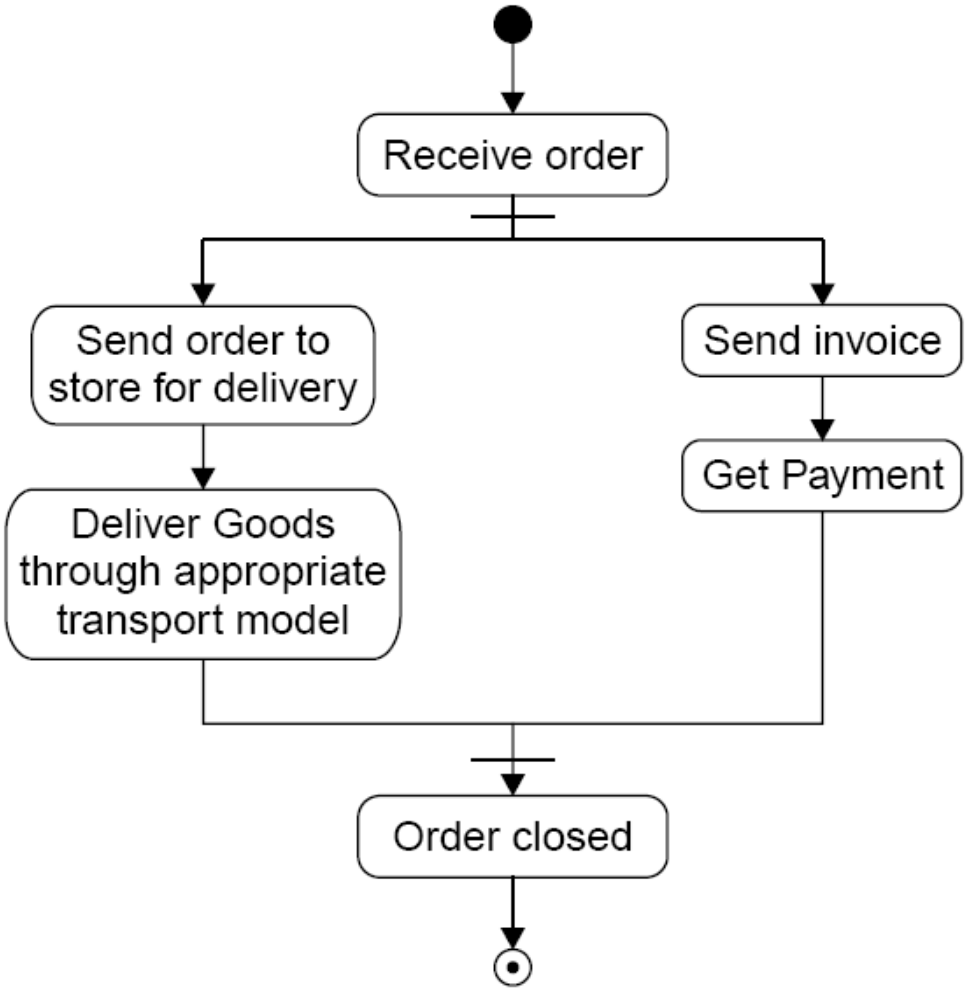


Fig. 26: Activity diagram

iii. Draw the interaction diagram

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

Steps to draws interaction diagrams are as under:

a) Firstly, we should identify that the objects with respec use case.

b) We draw the sequence diagrams for every use case.

d) We draw the collaboration diagrams for every use cas

The object types used in this analysis model are en
interface objects and control objects as given in fig. 27.

Fig. 27: Object types

iv. Draw the class diagram

The class diagram shows the relationship amongst classe
four types of relationships in class diagrams.

a) Association are semantic connection between cla
an association connects two classes, each clas
messages to the other in a sequence or a c
diagram. Associations can be bi-directional or unidi

b) **Dependencies** connect two classes. Dependencie
always unidirectional and show that one class, dep
definitions in another class.

c) **Aggregations** are stronger form of assoc
aggregation is a relationship between a whole and

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

v. Design of state chart diagrams

A state chart diagram is used to show the state space of a
the event that cause a transition from one state to anot
action that result from a state change. A state transition c
"book" in the library system is given in fig. 28.

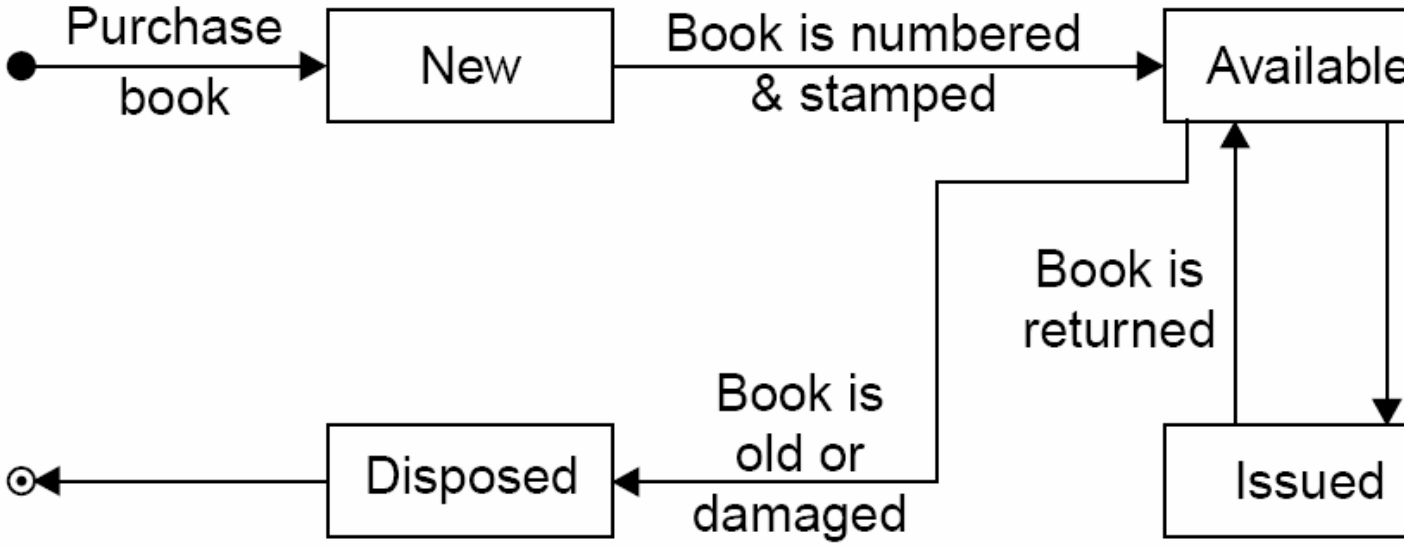


Fig. 28: Transition chart for "book" in a library syste

vi. Draw component and development diagram

Component diagrams address the static implementatio
system they are related to class diagrams in that a compo
maps to one or more classes, interfaces or collaboration.

Deployment Diagram Captures relationship betwee
components and the hardware.

Software Design

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

Issue of Books:

- A student of any course should be able to get books
- Books from General Section are issued to all but
  books are issued only for their respective courses.
- A limitation is imposed on the number of books a s
  issue.
- A maximum of 4 books from Book bank and 3 f
  General section is issued for 15 days only.The soft
  the current system date as the date of issue and calc
  of return.

- A bar code detector is used to save the student as information.
- The due date for return of the book is stamped on th

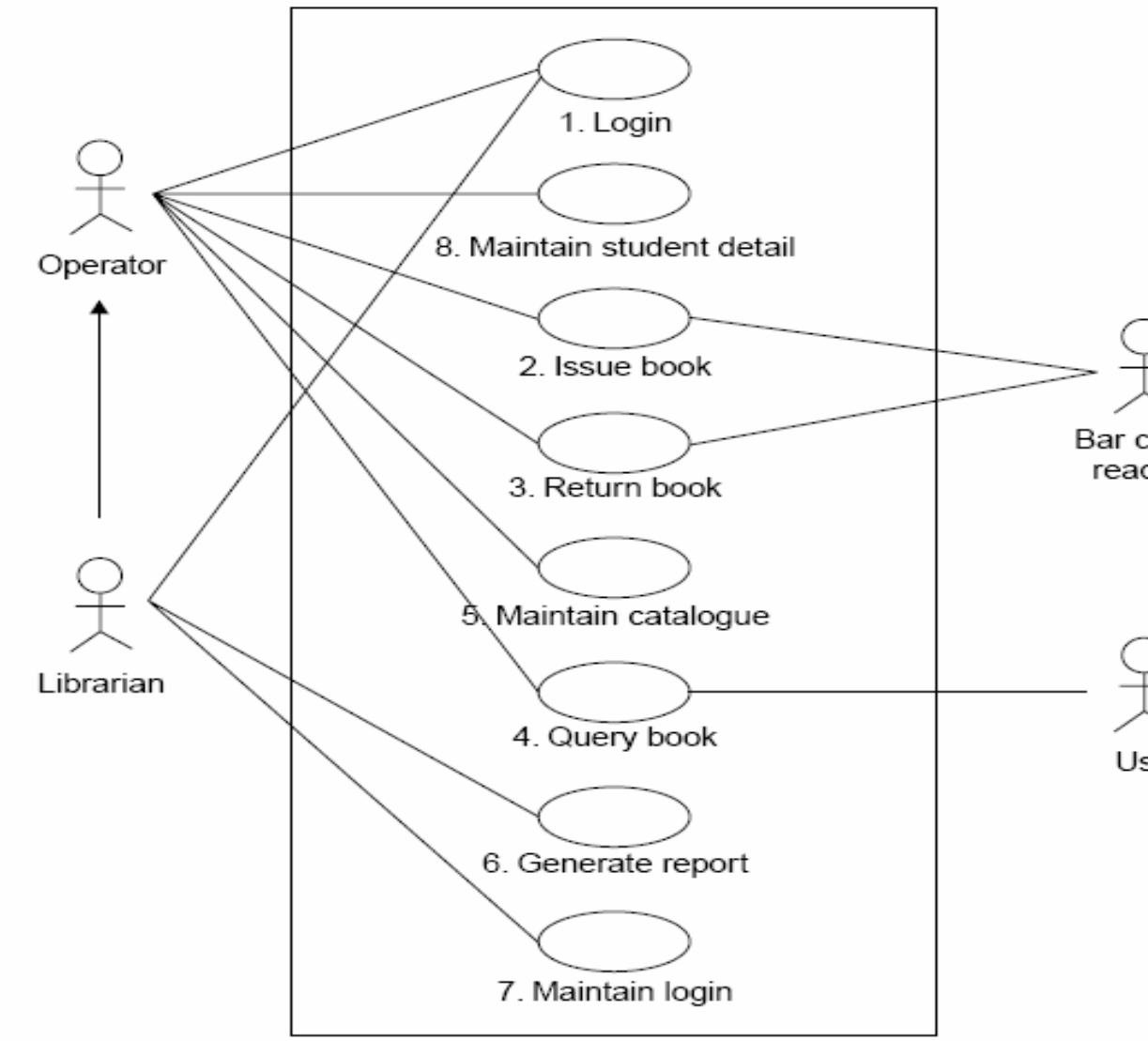
Return of Books:

- Any person can return the issued books.
- The student information is displayed using th detector.
- The system displays the student details on whos books were issued as well as the date of issue and book.
- The system operator verifies the duration for the iss
- The information is saved and the corresponding u place in the database.

Query Processing:

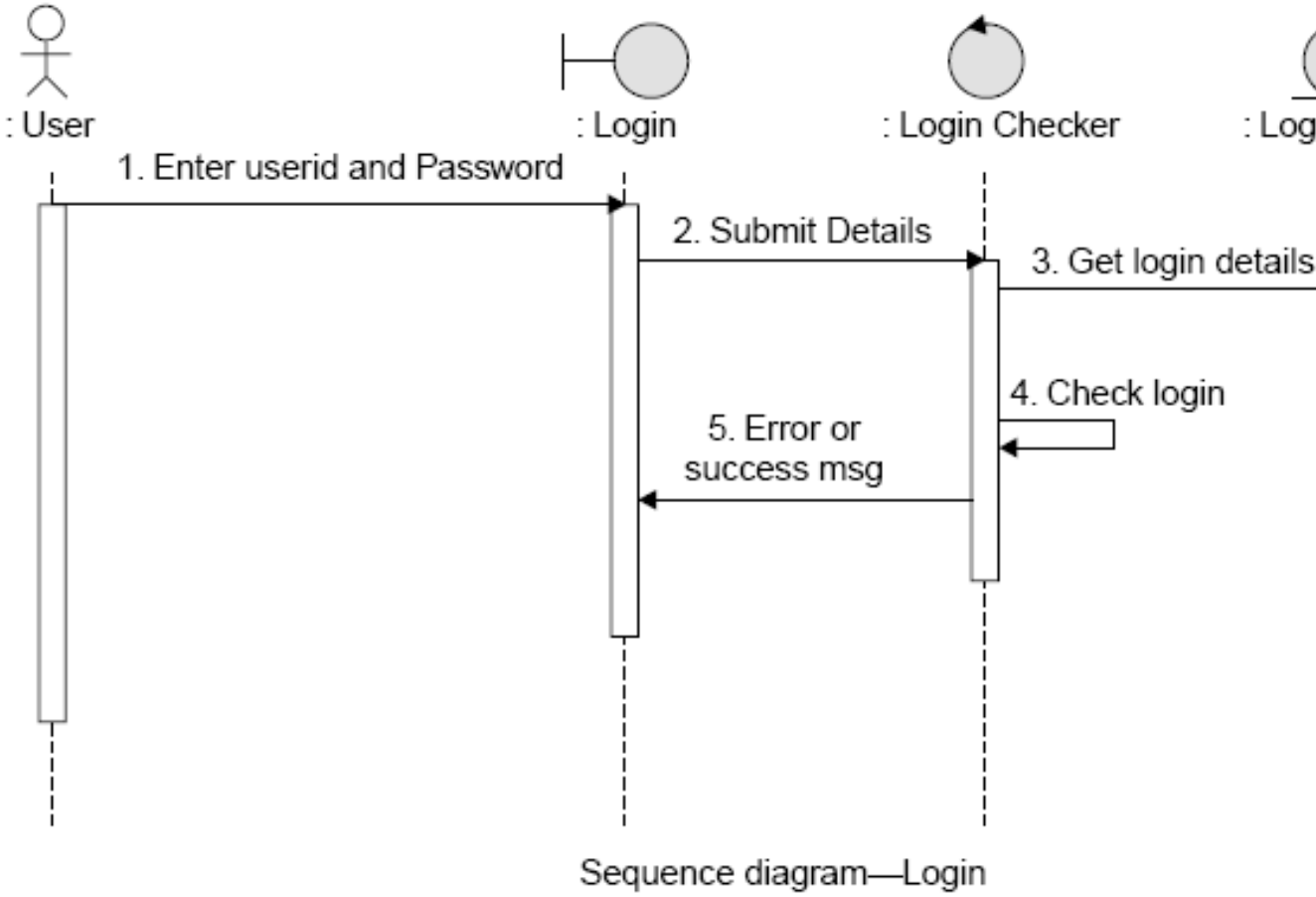
- The system should be able to provide information lil
- Availability of a particular book.
- Availability of book of any particular author.
- Number of copies available of the desired book.

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

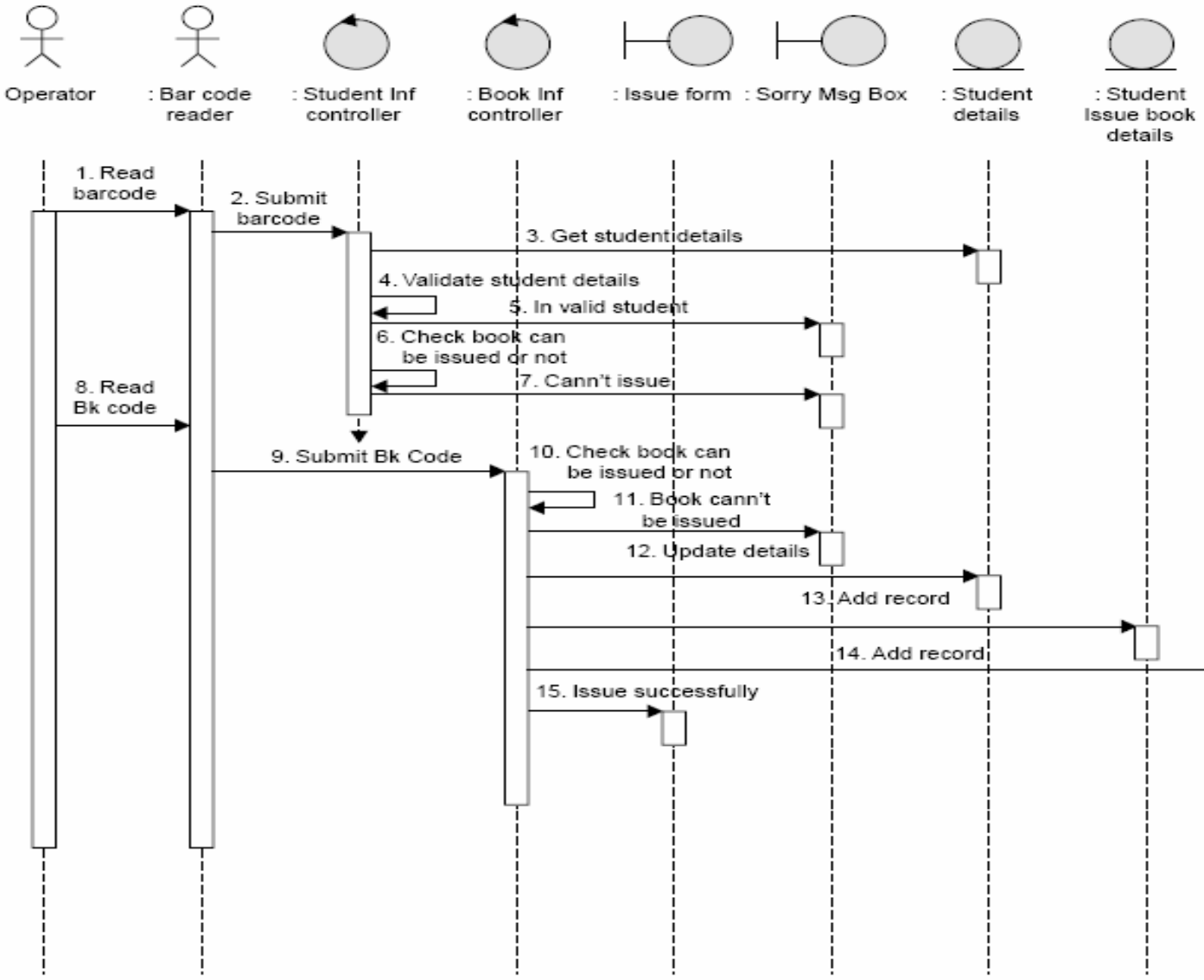


Use case diagram for library management system

Software Design

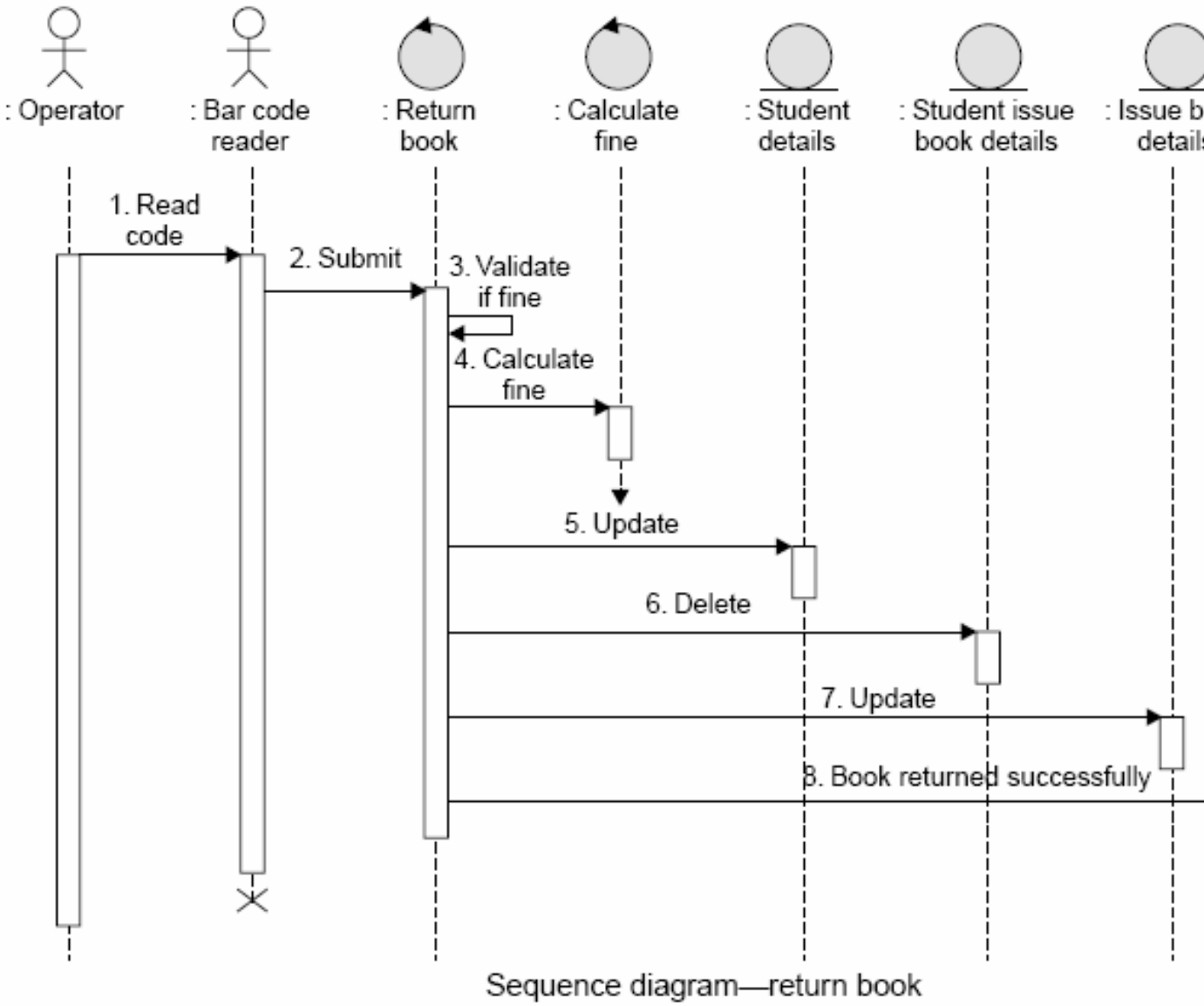


Sequence diagram—Login



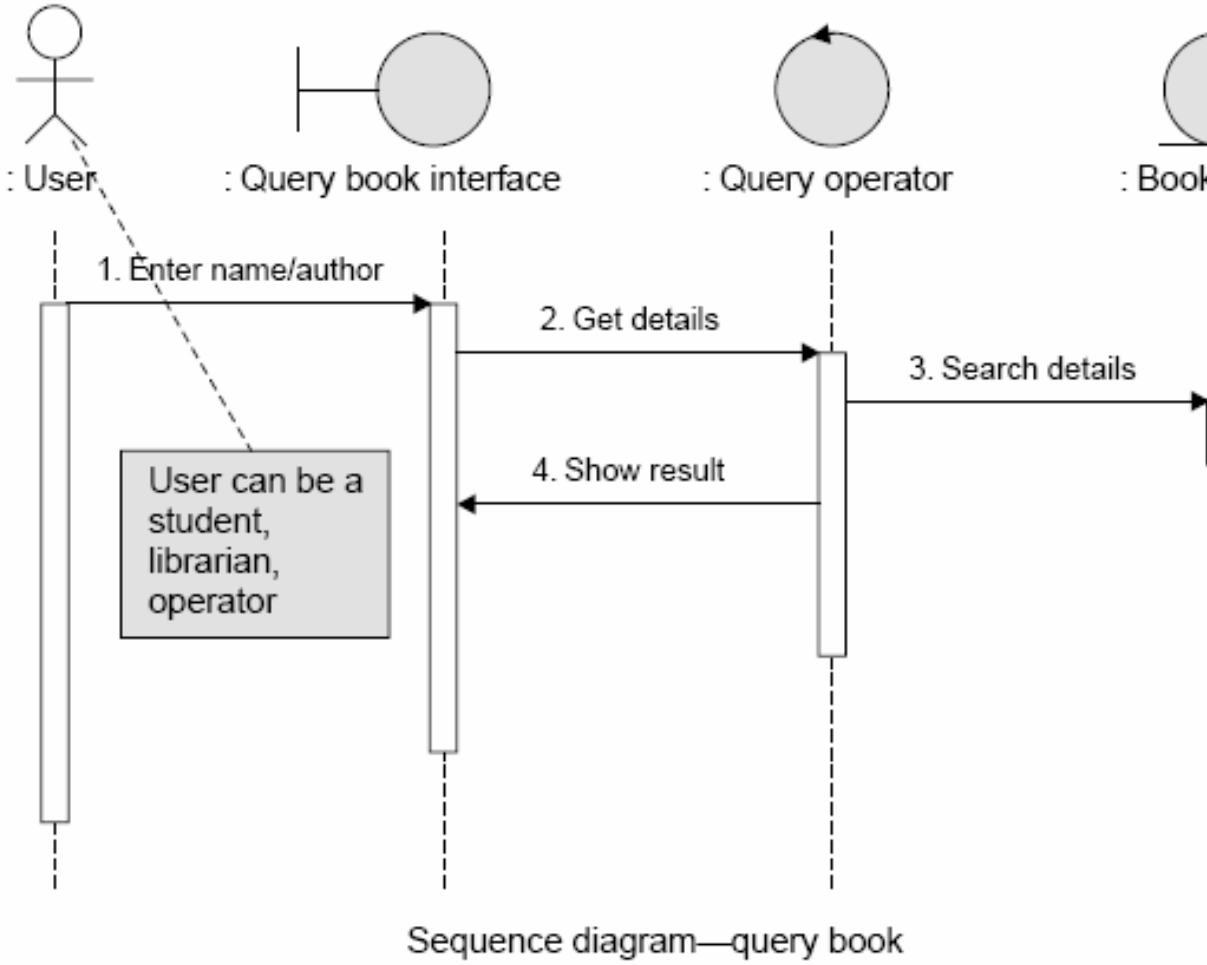
Sequence diagram—issue book

Software Design



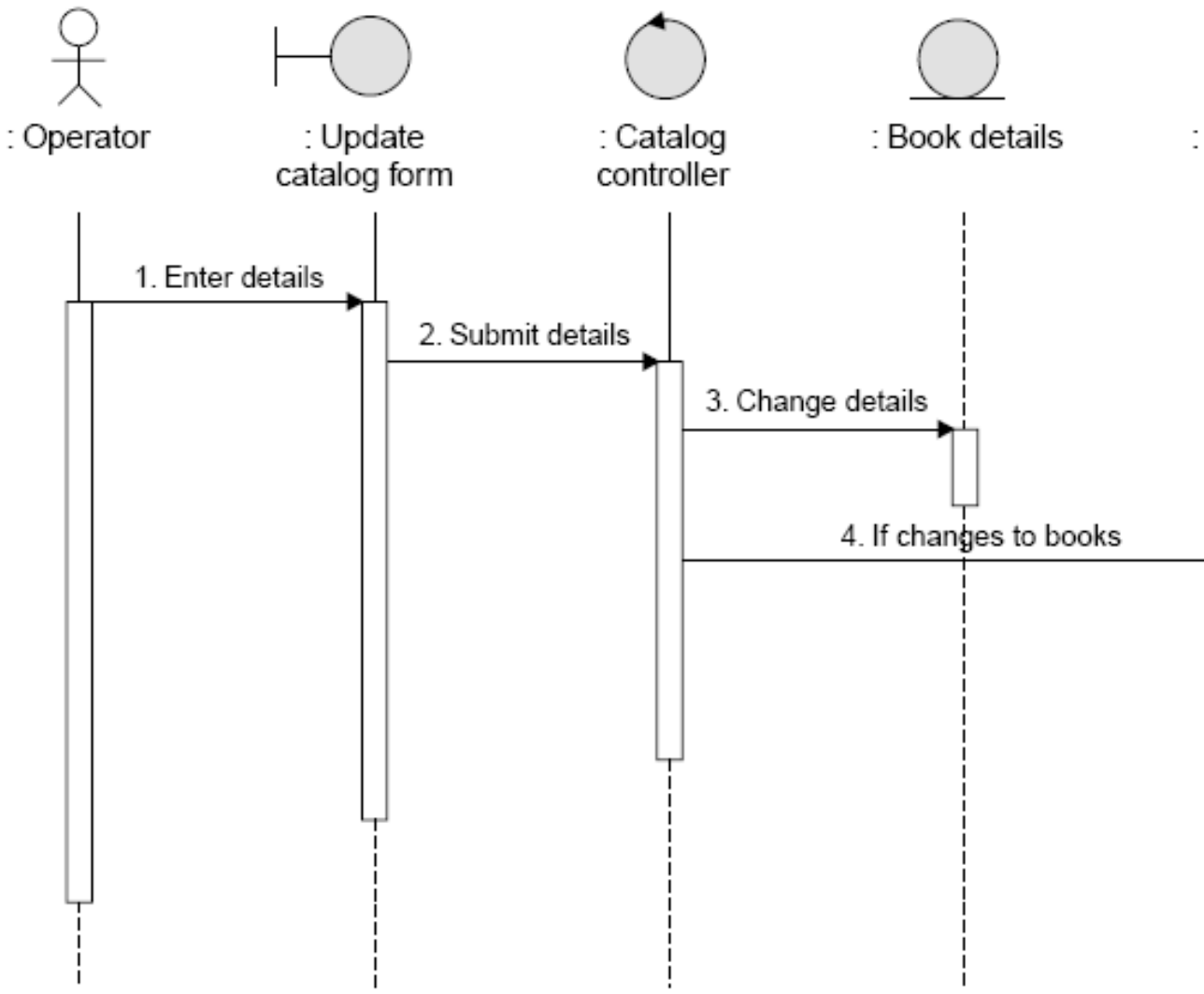
Sequence diagram—return book

Software Design



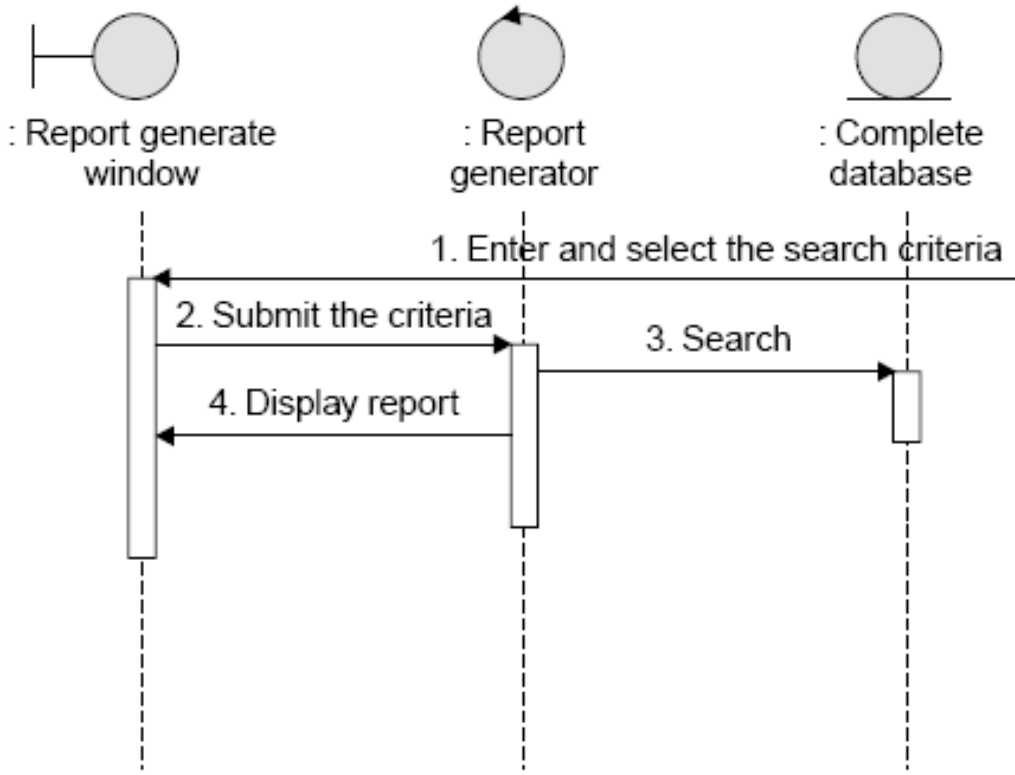
Sequence diagram—query book

Software Design



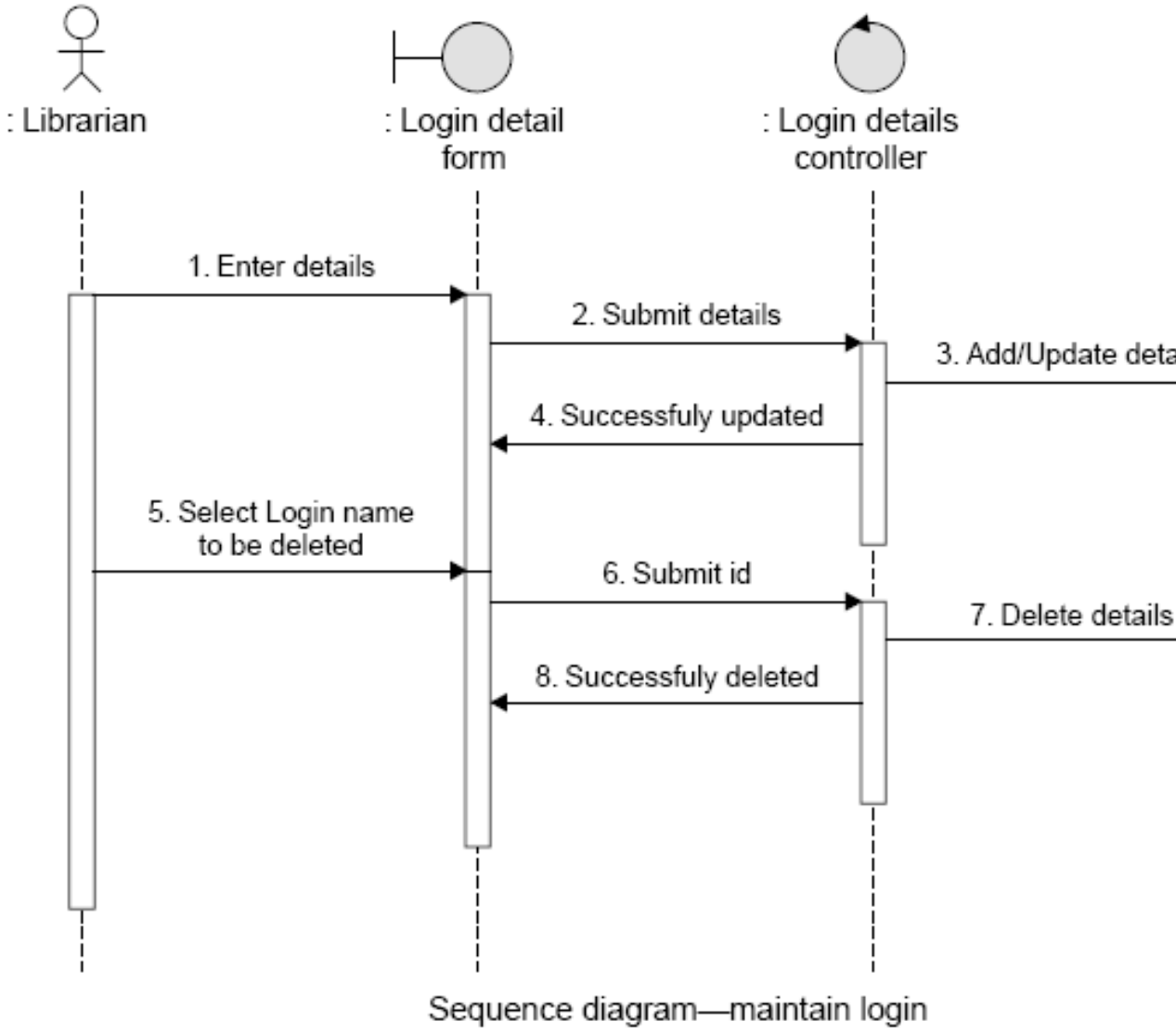
Sequence diagram—maintain catalog

Software Design



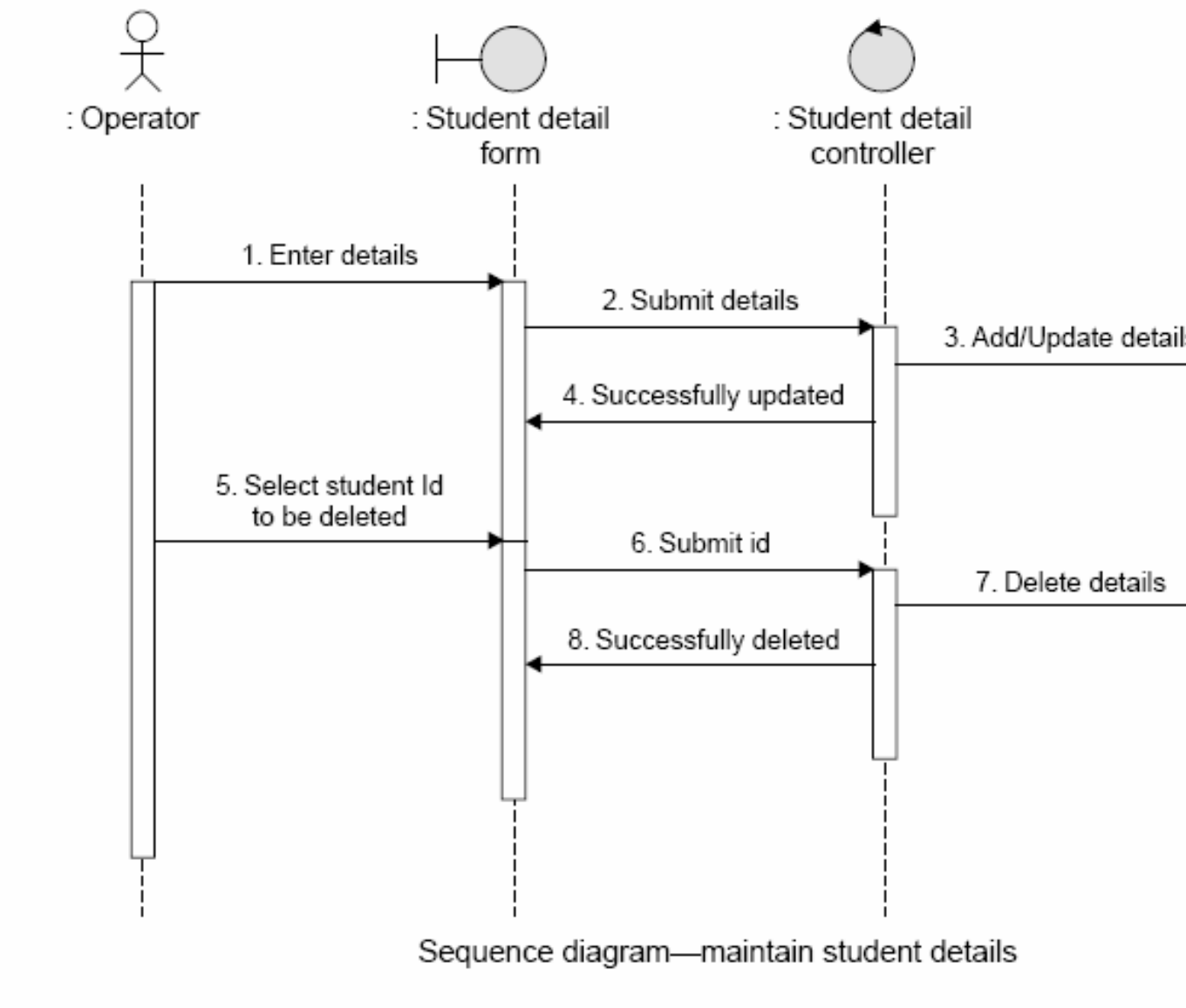
Sequence diagram—generate reports

Software Design



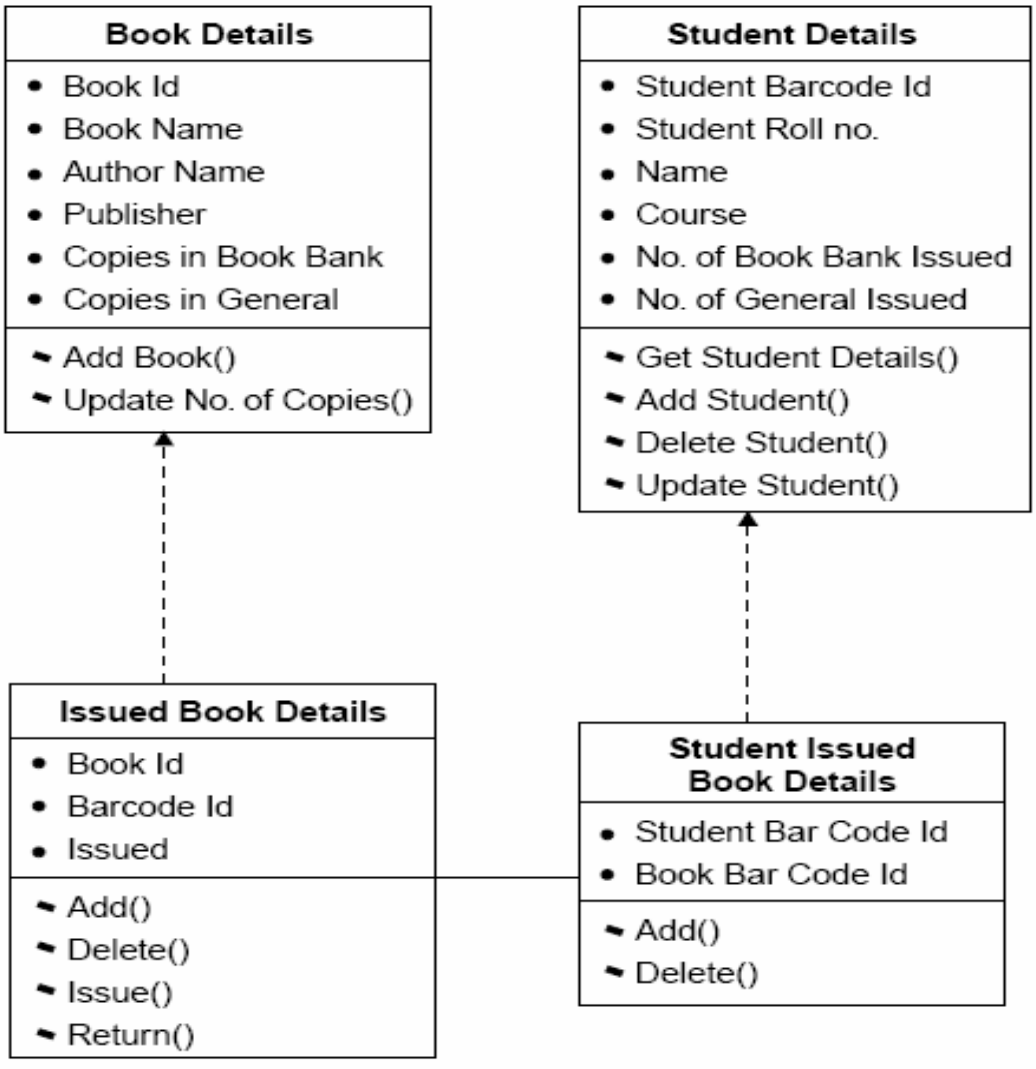
Sequence diagram—maintain login

Software Design



Sequence diagram—maintain student details

Class diagram of entity classes



Class diagram of entity classes

Multiple Choice Questions

Note: Choose most appropriate answer of the following q

5.1 The most desirable form of coupling is

(a) Control Coupling
(b) Data Coupling
(c) Common Coupling
(d) Content Coupling

5.2 The worst type of coupling is

(a) Content coupling
(b) Common coupling
(c) External coupling
(d) Data coupling

5.3 The most desirable form of cohesion is

(a) Logical cohesion
(b) Procedural cohesion
(c) Functional cohesion
(d) Temporal cohesion

5.4 The worst type of cohesion is

(a) Temporal cohesion
(b) Coincidental cohesio
(c) Logical cohesion
(d) Sequential cohesion

5.5 Which one is not a strategy for design?

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

Multiple Choice Questions

5.6 Temporal cohesion means

(a) Cohesion between temporary variables
(b) Cohesion between local variable
(c) Cohesion with respect to time
(d) Coincidental cohesion

5.7 Functional cohesion means

(a) Operations are part of single functional task and are placed in sam
(b) Operations are part of single functional task and are placed in mul
(c) Operations are part of multiple tasks
(d) None of the above

5.8 When two modules refer to the same global data area, they are related

(a) External coupled
(b) Data coupled
(c) Content coupled
(d) Common coupled

5.9 The module in which instructions are related through flow of control i

(a) Temporal cohesion
(b) Logical cohesion
(c) Procedural cohesion
(d) Functional cohesion

Multiple Choice Questions

5.10 The relationship of data elements in a module is called

(a) Coupling
(b) Cohesion
(c) Modularity
(d) None of the above

5.11 A system that does not interact with external environment is calle

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

5.12 The extent to which different modules are dependent upon each o

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

Exercises

5.1 What is design? Describe the difference between conceptu
technical design.

5.2 Discuss the objectives of software design. How do we
informal design to a detailed design?

5.3 Do we design software when we "write" a program?
software design different from coding?

5.4 What is modularity? List the important properties of a modu

5.5 Define module coupling and explain different types of coup

5.6 Define module cohesion and explain different types of cohe

5.7 Discuss the objectives of modular software design. What
of module coupling and cohesion?

5.8 If a module has logical cohesion, what kind of coupling i
likely to have with others?

5.9 What problems are likely to arise if two modules have high

Exercises

5.10 What problems are likely to arise if a module has low cohe

5.11 Describe the various strategies of design. Which design st popular and practical?

5.12 If some existing modules are to be re-used in building a which design strategy is used and why?

5.13 What is the difference between a flow chart and a structure

5.14 Explain why it is important to use different notation software designs.

5.15 List a few well-established function oriented sof techniques.

5.16 Define the following terms: Objects, Message, Abstr Inheritance and Polymorphism.

5.17 What is the relationship between abstract data types and cl

Exercises

5.18 Can we have inheritance without polymorphism? Explain.

5.19 Discuss the reasons for improvement using object-oriented

5.20 Explain the design guidelines that can be used to p
quality" classes or reusable classes.

5.21 List the points of a simplified design process.

5.22 Discuss the differences between object oriented and fun
design.

5.23 What documents should be produced on completion
phase?

5.24 Can a system ever be completely "decoupled"? That is, c
of coupling be reduced so much that there is no coup
modules?