Syllabus:Module : 2: REQUIREMENT ANALYSIS AND DESIGN

1. Describe software Requirement analysis and its needs 2. Describe Requirement specification 3. Describe the desirable characteristics of an SRS 4. Explain structure of an SRS document 5. Explain data flow diagram 6. Explain the role of Software Architecture 7. Describe how to plan for a software project 8. Define software Design 9. Describe software design concepts. 10. Explain function oriented design and its complexity metrics. 11. Explain object oriented design and its complexity metrics 12. Describe detailed design

---

1. Software Requirement Analysis &it's needs

- ✓ The goal of the requirements analysis and specification phase is to clearly understand the customer requirements and to systematically organize the requirement into specification document.
- ✓ The requirements analysis and specification phase starts after feasibility study phase is complete.
- ✓ The requirements analysis and specification phase ends with production and validation of the requirement specification document (SRS).
- ✓ SRS is the final outcome of the requirement and specification phase.
- ✓ Also serves as a contract between customer and development organization.
- ✓ Main activities carried out during this phase is
  1. Requirement gathering & analysis
  2. Requirement specification

Requirements Gathering and Analysis

- ✓ The requirements are almost never available in the form of single document from the customer.
- ✓ The requirements have to be gathered by analyst in several from several sources in bits and pieces.
- ✓ We can broadly divide the requirements gathering and analysis activity into two separate tasks:

1. Requirements Gathering
2. Requirements Analysis

Requirements Gathering

- ✓ Also known as requirement elicitation.
- ✓ The analyst collecting all the information needed for the development of system.
- ✓ Important ways in which analyst gather requirements are the following :
  1. Studying the existing documentation :
  2. Interview: Different users are interviewed to gather the different functionalities required by them.

3. Task Analysis: For each identified task, the analyst tries to formulate the different steps necessary to realize the service in consultation with the users.
4. Scenario Analysis: Different scenarios of a task can occur when the task is invoked under different situations.
5. Form Analysis: Different forms are analyzed to determine the data input to the system and the data that are output from the system.

Requirement Analysis

- ✓ After the requirement gathering is complete the analyst analyses the gathered requirement to clearly understand the exact customer requirement
- ✓ There are 3 main problems in the requirements that analyst need to identify and resolve :

  Anomaly or Ambiguity, Inconsistency, Incompleteness

  1. Anomaly or Ambiguity
  - ✓ Ambiguity in requirement.
  - ✓ When requirement is anomalous several interpretations of the requirements are possible.
  - ✓ Anomaly can lead to development of incorrect system

  2. Inconsistency
  - ✓ Two requirements are said to be inconsistent, if one requirement contradicts the other.

  3. Incompleteness
  - ✓ Some requirements have been overlooked.
  - ✓ Some features must be realized by the customer much later.

2.SOFTWARE REQUIREMENT SPCIFICATION (SRS)

- After the analyst gathered all the information regarding the software to be developed and has removed all the inconsistencies, he starts systematically organize the requirements in the form of SRS document.
- The SRS document usually contains all the user requirements in structured through informal form.
- Important categories of users of the SRS Document and their needs are as follows:

Users, Customers, and Marketing Personnel

- This set of audience in referring to SRS document is to ensure that the system as described will meet their needs.

Software Developers

- Refer SRS to make sure that they are developing exactly what is required by the customer.

Test engineers:

- To ensure that requirement is understandable from functionality point of view, so that they test the software and validate it's working.

User Documentation writers

- Ensure that they understand the features of the product well enough to beable to write user's manual.

Project Engineers

- Ensure that they can estimate the cost of the project easily by referring to the SRS document and that it contains all the required information to plan the project.

Maintenance Engineers

- SRS document helps to understand the functionalities of the system.
- A clear knowledge of functionalities helps them to understand the design and code, also determines what modifications needed for specific purpose.

3.CHARACTERISTICS OF GOOD SRS DOCUMENT

1. **Concise :** SRS document should be concise and at the same time unambiguous, consistent and complete. Irrelevant description reduce readability.
2. **Structured :** It should be well structured for easy understanding and modification.
3. **Traceable** : Verify the results of a phase with the previous phase, to analyse the impact of a change.
4. **Response to undesired Event** :It should characterize acceptable responses to undesired events.
5. **Black Box View** : It should specify only what the system should do and refrain from stating how to do these. Should only specify the external behavior of the system.
6. **Verifiable :** Verifiable means that it should be possible to determine wether or not requirements have been met in implementation.

4.STRUCTURE OF SRS

- The requirements document is devised in a manner that is easier to write, review, and maintain.
- It is organized into independent sections and each section is organized into modules or units.
- Note that the level of detail to be included in the SRS depends on the type of the system to be developed and the process model chosen for its development.
- For example, if a system is to be developed by an external contractor, then critical system specifications need to be precise and detailed.
- Similarly, when flexibility is required in the requirements and where an in-house development takes place, requirements documents can be less detailed.

- ➢ Since the requirements document serves as a foundation for subsequent software development phases, it is important to develop the document in the prescribed manner.
- ➢ For this, certain guidelines are followed while preparing SRS.
- ➢ These guidelines are listed below.

1. **Functionality:** It should be separate from implementation.
2. **Analysis model:** It should be developed according to the desired behaviour of a system. This should include data and functional response of a system to various inputs given to it.
3. **Cognitive model:** It should be developed independently of design or implementation model. This model expresses a system as perceived by the users.
4. **The content and structure** of the **specification:** It should be flexible enough to accommodate changes.
5. **Specification:** It should be robust. That is, it should be tolerant towards incompleteness and complexity.

- ➢ The information to be included in SRS depends on a number of factors, for example, the type of software being developed and the approach used in its development.
- ➢ If software is developed using the iterative development process, the requirements document will be less detailed as compared to that of the software developed for critical systems.
- ➢ This is because specifications need to be very detailed and accurate in these systems.
- ➢ A number of standards have been suggested to develop a requirements document.
- ➢ However, the most widely used standard is by IEEE, which acts as a general framework.
- ➢ This general framework can be customized and adapted to meet the needs of a particular organization.

- ✓ Each SRS fits a certain pattern; thus, it is essential to standardize the structure of the requirements document to make it easier to understand.
- ✓ For this IEEE standard is used for SRS to organize requirements for different projects, which provides different ways of structuring SRS.
- ✓ Note that in all requirements documents, the first two sections are the same.

This document comprises the following sections.

1. **Introduction:** This provides an overview of the entire information described in SRS. This involves purpose and the scope of SRS, which states the functions to be

performed by the system. In addition, it describes definitions, abbreviations, and the acronyms used. The references used in SRS provide a list of documents that is referenced in the document.

2. **Overall description:** It determines the factors which affect the requirements of the system. It provides a brief description of the requirements to be defined in the next section called 'specific requirement'. It comprises the following sub-sections.
3. **Product perspective:** It determines whether the product is an independent product or an integral part of the larger product. It determines the interface with hardware, software, system, and communication. It also defines memory constraints and operations utilized by the user.
4. **Product functions:** It provides a summary of the functions to be performed by the software. The functions are organized in a list so that they are easily understandable by the user:
5. **User characteristics:** It determines general characteristics of the users.
6. **Constraints:** It provides the genera1 description of the constraints such as regulatory policies, audit functions, reliability requirements, and so on.
7. **Assumption and dependency:** It provides a list of assumptions and factors that affect the requirements as stated in this document.
8. **Apportioning of requirements:** It determines the requirements that can be delayed until release of future versions of the system.
9. **Specific requirements:** These determine all requirements in detail so that the designers can design the system in accordance with them. The requirements include description of every input and output of the system and functions performed in response to the input provided. It comprises the following subsections.
10. **External interface:** It determines the interface of the software with other systems, which can include interface with operating system and so on. External interface also specifies the interaction of the software with users, hardware, or other software. The characteristics of each user interface of the software product are specified in SRS. For the hardware interface, SRS specifies the logical characteristics of each interface among the software and hardware components. If the software is to be executed on the existing hardware, then characteristics such as memory restrictions are also specified.
11. **Functions:** It determines the functional capabilities of the system. For each functional requirement, the accepting and processing of inputs in order to generate outputs are specified. This includes validity checks on inputs, exact sequence of operations, relationship of inputs to output, and so on.
12. **Performance requirements:** It determines the performance constraints of the software system. Performance requirement is of two types: static requirements and

dynamic requirements.**Static requirements** (also known as **capacity requirements)** do not impose constraints on the execution characteristics of the system. These include requirements like number of terminals and users to be supported. **Dynamic requirements** determine the constraints on the execution of the behaviour of the system, which includes response time (the time between the start and ending of an operation under specified conditions) and throughput (total amount of work done in a given time).

13. **Logical database of requirements:** It determines logical requirements to be stored in the database. This includes type of information used, frequency of usage, data entities and relationships among them, and so on.
14. **Design constraint:** It determines all design constraints that are imposed by standards, hardware limitations, and so on. Standard compliance determines requirements for the system, which are in compliance with the specified standards. These standards can include accounting procedures and report format. Hardware limitations implies when the software can operate on existing hardware or some pre-determined hardware. This can impose restrictions while developing the software design. Hardware limitations include hardware configuration of the machine and operating system to be used.
15. **Software system attributes:** It provide attributes such as reliability, availability, maintainability and portability. It is essential to describe all these attributes to verify that they are achieved in the final system.
16. **Organizing Specific Requirements:** It determines the requirements so that they can be properly organized for optimal understanding. The requirements can be organized on the basis of mode of operation, user classes, objects, feature, response, and functional hierarchy.
17. **Change management process:** It determines the change management process in order to identify, evaluate, and update SRS to reflect changes in the project scope and requirements.
18. **Document approvals:** These provide information about the approvers of the SRS document with the details such as approver's name, signature, date, and so on.
19. **Supporting information:** It provides information such as table of contents, index, and so on. This is necessary especially when SRS is prepared for large and complex projects

…………………………………………………………………………………….

Contents of SRS Document (or) Important Categories of Customer Requirements

- The different user requirements can be categorized into following categories:

1. Functional Requirements
2. Non-functional Requirements
3. Goals of Implementation

1. Functional Requirements

- Functional requirements discuss the functionalities required by the users from the system.
- It clearly described each function along with the corresponding input and output data set.
- First we have to identify the *high level* function of the system.
- After that it can be split into smaller sub requirements.
- *High level* function is one using which the user can get some useful piece of work done.
- A high level function usually requires a series of interactions between the system and one or more users.

Identifying the functional Requirements:

- Each high level requirement characterized way of system usage (service invocation) by some user to perform some meaningful piece of work.
- First identify the different types of users who might use the system and then identify the different services expected from the software by different types of users.

Documentation of functional requirement

- Once all the high level requirement have been identified, they can be documented.
- A function can be documented by identifying the state at which the data is to be input to the system, its input data domain, the output data domain and the type of processing to be carried on the input data to obtain the output data.
- It has several sub-requirements corresponding to the different user interactions.
- These different interaction sequences capture the different scenarios.

Example : Withdraw cash from ATM

R.1. **Withdraw cash**

Description : The withdraw function cash function first determines the type of account that the user has and the account number from which the user wishes to with draw the cash. It checks the balance to determine the whether the requested amount is available in the account.If enough balance is available, it outputs the required cash, otherwise it generates an error message.

R.1.1. Select withdraw amount option

Input : "withdraw amount "option

*Output :*User prompted to enter the account type

R.1.2. Select Account Type

Input **:**User Option from anyone of the following : Saving/checking/ Deposit

Output : Prompt to enter amount

R.1.3. Get required Amount

*Input :*Amount to be withdrawn in integer values greater than 100andlessthan 10,000 in multiples of 100.

*Output :*the requested cash and printed transaction statement.

Processing : The amount is debited from the user's account if sufficient balance is available otherwise an error message is displayed.

2. **Non functional Requirements**
    - The non-functional requirement deals with characteristics of a system that cannot be expressed as functions.
    - Non-functional requirements address aspects concerning
    - The non-functional requirements include reliability issues accuracy of results , constraints on the system implementation,maintainability, portability, usability, timing and throughput..
    - The non-functional requirements are non-negotiable obligations that must be supported by the system.

3. **Goals of Implementation**

- The goals of implementation part of the SRS document offers some general suggestions regarding development.
- The developers may take these suggestions into account if possible.
- The goals of implementation section might document issues such as revisions to the system functionalities that may be required in the future; new devices supported in future reusability issues etc.
- Not checked by the customer for conformance at the time of acceptance testing

Organization of the SRS Document

SRS document can be organized as follows:

- ➢ Introduction

- Goals of implementation
- Functional requirements
- Non-functional requirements
- Behavioural description

5.DATA FLOW DIAGRAM (DFD)

- Also known as bubble chart.
- Represent a system in terms of the input data to the system, various processing carried out on those data, and the output data generated by the system.
- DFD model uses very limited number of primitive symbols to represent functions performed by a system and the data flow among these functions.

Primitive Symbols Used for constructing DFD

1. **Function symbol:** A function is represented using a circle. This symbol is called a process or a *bubble*.

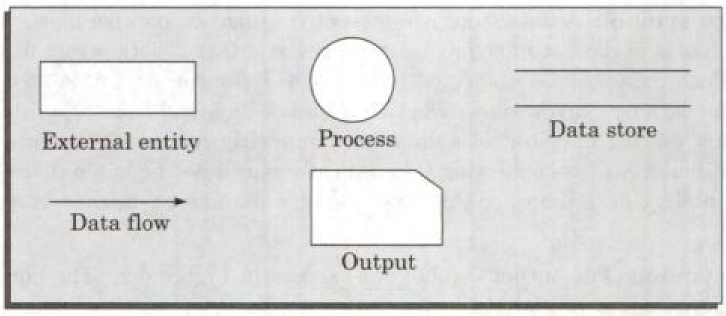


Figure 6.2: Symbols used for designing DFDs.

2. **External entity symbol:** An external entity such as a librarian, a library member, etc. is represented by a rectangle. The external entities are essentially those physical entities external to the software system which interact with the system by inputting data to the system or by consuming the data produced by the system. In addition to the human users, the external entity symbols can be used to represent external hardware and software such as another application software that would interact with the software being modelled.

3. **Data flow symbol:** A directed arc (or an arrow) is used as a data flow symbol. A data flow symbol represents the data flow occurring between two processes or between an external entity and a process in the direction of the data flow arrow.

4.Data store symbol :a data store is represented using 2 parallel lines.it represents a logical file or a physical file on disk.

5.Output symbol : The output symbol is used when a hard copy is produced.

Synchronous and asynchronous operations

- If two bubbles are directly connected by a data flow arrow, then they are synchronous
- They operate at same speed.

- If two bubbles are connected through a data store, then the speed of operation of the bubbles are independent.

Data Dictionary

- It lists the purpose of all data items and the definition of all composite data items in terms of their component data items.
- There will be several DFD but has a single data dictionary.

Advantages of data Dictionary

1. Provides standard terminology for all relevant data.
2.Helps the developers to determine the definition of different data structures.
3. The data dictionary helps to perform impact analysis

Data Definition :

- Composite data items can be defined in terms of primitive data items using the following operators.

1. + : Denotes composition of two data items eg: a+b represents a and b
2. [, ,] : represents selection eg: [a,b] means either a occurs or b occurs
3. () ; represent optional data eg : a+(b) means either a or a+b occurs
4. {} : represent iterative data definition eg : {name}5 represent 5 name data
5. = : represents equivalence
6. /* */ : considered as comment

Developing DFD Model of a system

- DFD model represents how much each input is transformed to its corresponding output data through a hierarchy of DFDs.
- The DFD model of a problem consists of many DFD's and a single data dictionary.
- Top level DFD is called a level 0 DFD or context diagram.
- At each successive lower levels DFDs, more and more details are gradully introduced.
- Lower DFD's are constructed by the decomposition of the higher level functions.

Context Diagram

- Most abstract (highest level)data flow representation of a system.
- Entire system as a single bubble.
- Bubble is annotated as name of the software system being developed.
- Only bubble in the DFD model where noun is used for naming.
- Purpose of the context diagram is to capture the context of the system.

Level 1 DFD

- Contains 3 to 7 bubbles.
- To develop the level 1 DFD , examine high level function requirements in SRS document .

- Contains functions between 3 to 7 means , it can directly represent as bubbles.
- Contains more than 7 means , some of them has to be combined to make maximum of 7 bubbles.
- If it is less than 3, then some functions should split to get minimum of 3 functions.

Decomposition

- Each bubble in the DFD represents a function performed by the system.
- The bubbles are decomposed into subfunctions at the successive levels of DFDmodel.
- Decomposition of a bubble is also known as *factoring* or *exploding* a bubble.

Systematic steps to develop DFD :

1. Construction of Context diagram

Examine the SRS document to determine :

a) Different high level functions that system needs to perform
b) Data input to every high level functions
c) Data output from every high level function

2. Construction of level 1 Diagram

Examine SRS document

a) Minimum 3 functions, if less split the functions
b) Maximum 7, if greater , combine to get 7 functions.

3. Construction of lower level diagrams

a) Decompose each high level function into sub functions through the following set of activities:
   ✓ Identify the Different subfunctionsof high level functions
   ✓ Identify Data input to sub functions
   ✓ Identify Data output to sub functions
   ✓ Identify the interactions among these subfunctions

b) Recursively repeat step 3(a) for each subfunction until a subfunction can be represented by using a simple algorithm.

Numbering of Bubbles :

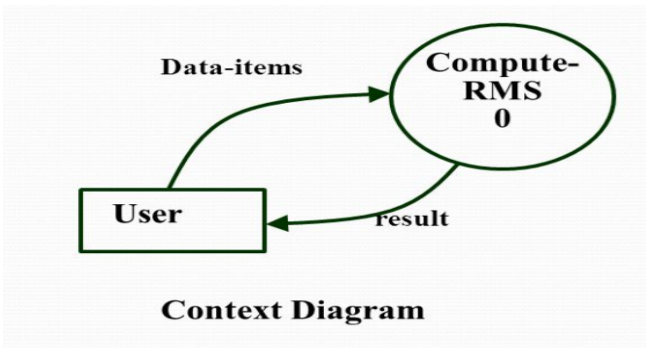
- Should give unique numbers to the bubbles.
- Bubble at context level is usually number as 0.
- Level 1 is numbered as 0.1,0.2,0.3 etc.
- When bubble ‘x’ is composed , the its child is labeled as x.1,x.2 etc.

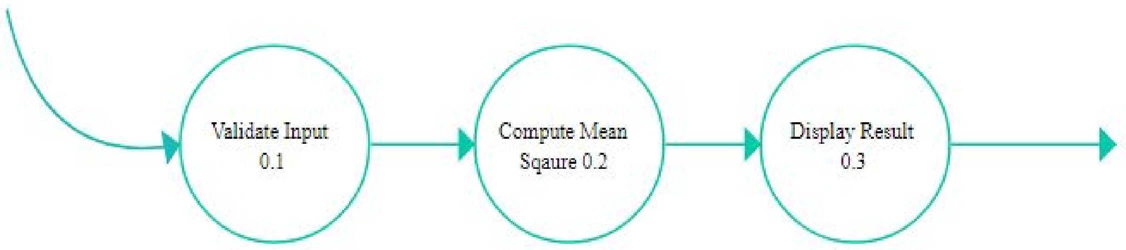
Balancing DFD

- The data flow into or out of a bubble must match the data flow at the next level of DFD. This known as *balancing DFD.*

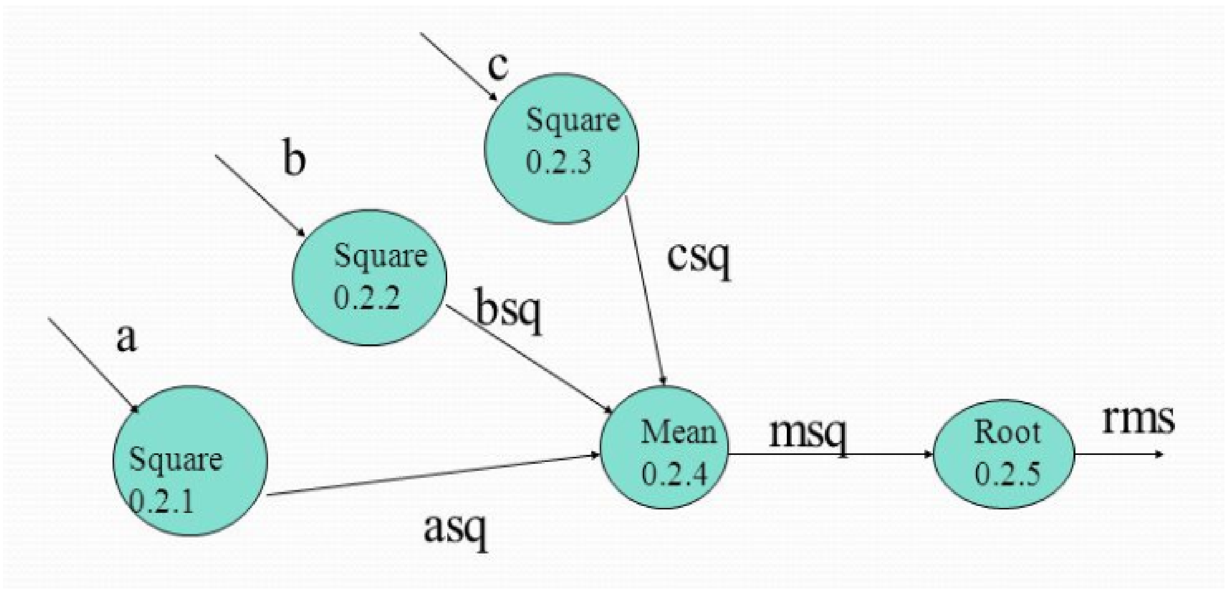
Eg**: RMS CALCULATING SOFTWARE**

- Rms system should read 3 integral values from user in the range of -1000 to 1000 and would determine the root mean square of three inputs and display it.





LEVEL-1 DFD



LEVEL-2 DFD

Data Dictionary :
Data-items : { integer}3
rms : float
a: integer
b: integer
c: integer
asq :integer
bsq : integer
csq: integer
msq:integer

6.EXPLAIN THE ROLE OF SOFTWARE ARCHITECTURE

- Responsibilities of a **Software Architect**. A **software architect** is responsible for creating or selecting the most appropriate **architecture** for a system (or systems), such that it suits the business needs, satisfies stakeholder requirements, and achieves the desired results under given constraints

7. DESCRIBE HOW TO PLAN FOR A SOFTWARE PROJECT

- Project planning is undertaken and completed even before any development activity starts.
- Consist of following activities
  1. Estimation : The following project attributes have to be estimated.
     i. *Cost* How much is it going to cost to develop the software?
     ii. *Duration* How long is it going to take to develop the product?
     iii. *Effort* How much effort would be required to develop the product?
  2. Scheduling : Schedules for manpower and resources were developed.
  3. Staffing : Staff organization and staff plans have to be made.
  4. Risk Management : Risk identification, analysis and planning have to be done.
  5. Miscellaneous Plans : Other plans such as quality assurance, configuration management, etc have to be done.
- Observe that size estimation is the first activity.
- Size is the most fundamental parameter based on which all other estimates are made.
- Based on size estimation, the effort required to complete the project and duration are estimated.
- Based on the effort estimation the cost of the project is computed.
- The estimated cost forms the negotiations with customer made.

8. DEFINE SOFTWARE DESIGN

Software Design

- The activities needed to transform the SRS Document into design document.
- **Software design** is the process of implementing **software** solutions to one or more sets of problems.
- One of the main components of **software design** is the **software** requirements analysis (SRA).
- SRA is a part of the **software** development process that lists specifications used in **software** engineering.

9. DESCRIBE SOFTWARE DESIGN CONCEPTS.

Outcome of Design Process

- **Different modules required :** –Different modules should be clearly identified
- **Control relationships Among Modules :-** Due to function calls between the two modules.
- **Interfaces Among different modules :-** identifies the exact data exchange between the modules
- **Data Structures of the Individual Module :-** Finding suitable data structures for the data to be stored in a module.
- **Algorithms required to implement the individual modules :-**Algorithms for the various modules are carefully designed and documented.

Classification of Design Activities

- Good software design is arrived by using a series of steps called the design activities
- Depending on the order in which various design activities are performed, classified into 2 important stages :

Preliminary design(high level design) ,Detailed design

- Through high level design, a problem is decomposed into a set of modules, the control relationships among various modules identified, and also the interfaces among various modules are identified.
- The outcome of highlevel design is called the *program structure* or the *software architecture.*
- Many notations are used to represent a high level design.
- A notation that is widely being used for procedural development is structure chart.
- Another technique is UML being used in object oriented designs.
- Once the high level design is complete, the detailed design is undertaken.

- During detailed design, each module is examined carefully to design its data structures and the algorithms.
- The outcome of detailed design stage is usually documented in the form of a module specification(MSPEC) document.

Characteristics of a good Software Design

- ✓ Correctness : a good design should first of all be correct.
- ✓ Understandability : Easily understandable.
- ✓ Efficiency : should address resource, time and cost optimization issues.
- ✓ Maintainability : Should be easy to change.
- Understability of a design solution is possibly the most important issue to be considered while judging the goodness of design.

An Understandable design is modular and layered

- A design solution is understandable, if it is modular and the modules arranged in layers.

Modularity:

- In a modular design the problem has decomposed into set of modules.
- The division of module is based on divide and conquer principle.
- If different modules have either no interactions or little interactions with each other, then each module can be understood separately.
- A design solution is considered to be highly modular, if the different modules in the solution have high cohesion and their inter-module coupling are low.

Layered Design: Layered arrangements of modules

- In a layered design the modules are arranged in a hierarchy of layers(like a tree diagram).
- A layered design can be considered to be implementing control abstraction, since modules at lower layers are unaware of higher layers.
- In a layered design solution, the modules are arranged into several layers based on their call relationships.
- A Module is allowed to call only the modules that are at a lower layer.
- The modules at the intermediate layers offer services to their higher layer by invoking the services of the lower layer module.
- A layered design has control abstraction and is easier to understand and debug.
- In a layered design errors are isolated.
- In a control hierarchy or layered design , a module that controls another module is said to be *superordinate* to it.
- A module controlled by another module is said to be *subordinate* to the controller.

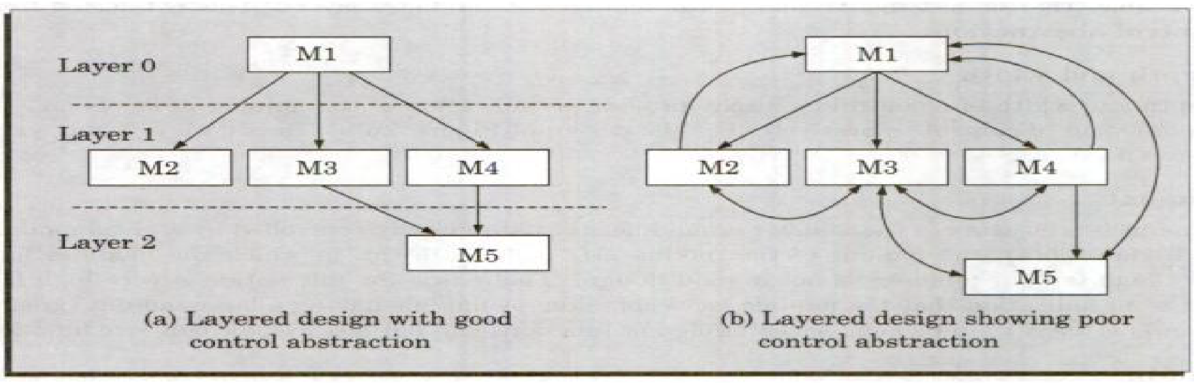

Layer 0
Layer 1
Layer 2
M1
M2
M3
M4
M5
(a) Layered design with good control abstraction
(b) Layered design showing poor control abstraction

Figure 5.6: Examples of good and poor control abstraction.

Visibility : - A module B is said to be visible to another module A, if A directly calls B. thus only the immediately lower layer modules are said to be visible to a module.

Control Abstraction :A module should only invoke the functions of the module modules in higher layers should not be visible.

Depth and width : Depth and width of a control hierarchy provide an indication of the number of layers and the overall span of control respectively.

Fan Out : - Fan out is the measure of the number of modules that are directly controlled by a given module.

Fan In ;- Fan in indicates the number of modules that directly invoke a given module

Approaches to software design (OR) Classification of Design Methodologies

- 2 methods
  1.Function oriented design (procedural oriented design)
  2. Object oriented design

10. EXPLAIN FUNCTION ORIENTED DESIGN AND ITS COMPLEXITY METRICS.

A. FUNCTION- ORIENTED SOFTWARE DESIGN

SA/SD Methodology

Involves carries out 2 activities

1. Structured Analysis(SA)
2. Structured Design(SD)

- During structured analysis,the SRS document is transformed into a DFD(Data Flow Diagram)model.
- During structure design , the DFD model is transformed into structure chart.

1)Structured Analysis

- Purpose is to capture the detailed structure of the system.
- Based on the following principles :

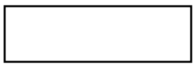
1. Top down decomposition approach
2. Application of divide and conquer principle.
3. Graphical representation of the analysis results using DFDs

2)Structured Design

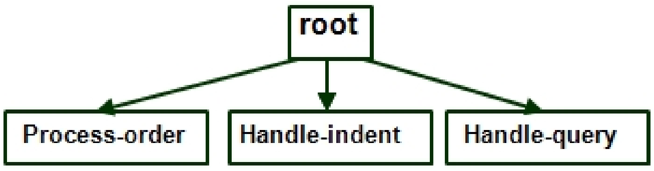
- Theaimofstructureddesignistotransformtheresultsofthestructuredanalysis(i.e.aDFD representationintoastructurechart) into structure chart.
- Representsthesoftwarearchitecture.
- The structure chart representation can be easily implemented using some programming language.
- The basic building blocks using which structure chart are designed are the following:

Building blocks

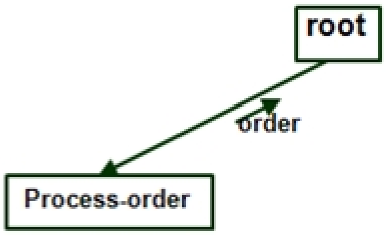
1. **Rectangular boxes** : represents module, annotated with name of the module



2. **Module Invocation Arrows** : during the execution control is passed from one module to the other in the direction of arrow.

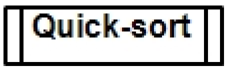


3. **Data Flow arrows** : data passing from one module to another in the direction of arrow



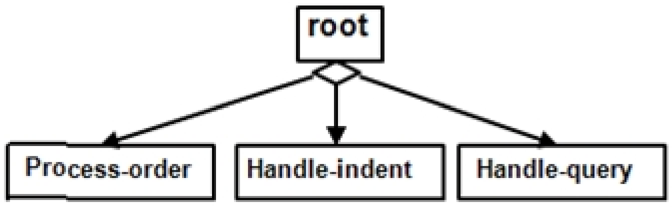
4. **Library Modules :** represents frequently called modules

✓ A rectangle with double sided edges
✓ Simplifies drawing when a module is called by several modules

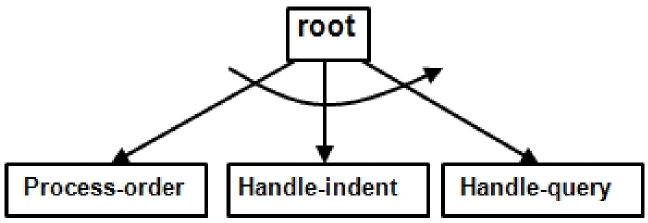


5. **Selection :** Diamond symbol

✓ One module of several modules connected to diamond symbol is invoked depending on the outcome of condition attached with it.



6. **Repetition :**A loop around control flow arrows denotes that the concerned modules are invoked repeatedly.



Transformation of a DFD into a Structure Chart

Two ways:

- ✓ TransformAnalysis
- ✓ TransactionAnalysis

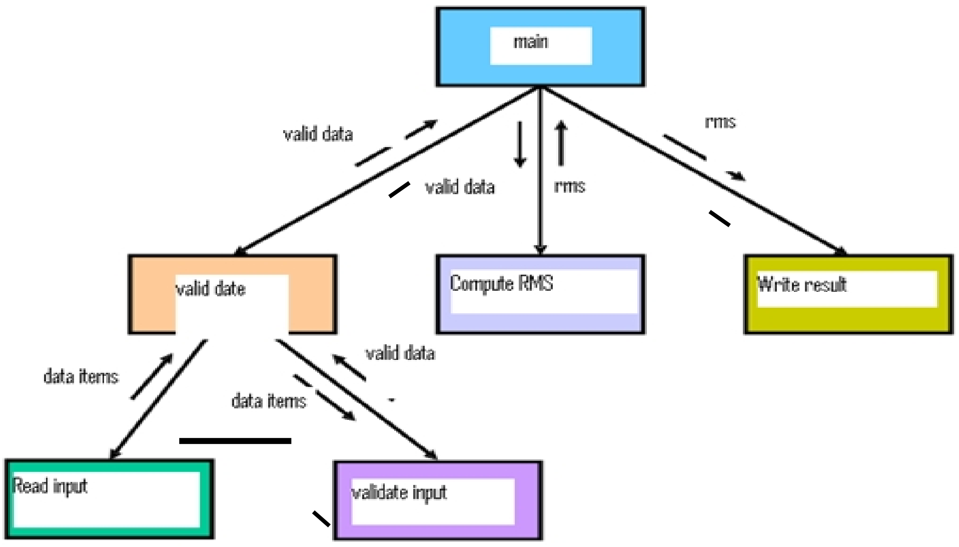
Transform Analysis

- Transformanalysisidentifiestheprimaryfunctionalcomponents(modules)andthehigh levelinputsandoutputsforthecomponents.
- Thefirststepintransformanalysisistodividethe DFD into 3 types of parts:
- ✓ Input
- ✓ Logicalprocessing
- ✓ Output
- TheinputportionoftheDFDincludesprocessesthattransform inputdatafrom physical(e.g. characterfromterminal)tologicalforms(e.g.internaltables,lists,etc.).Eachinputportionis called an *afferent* branch.
- TheoutputportionofaDFDtransformsoutputdatafromlogicaltophysicalform.Each outputportionis called*efferent*branch.TheremainingportionofaDFDiscalled*central transform*

TransactionAnalysis

- Atransaction allowsthe user to perform some meaningful piece of work by using the software.
- Oneofseveral possiblepathsthroughtheDFDistraverseddependingupontheinput dataitem.
- Foreachidentifiedtransaction,tracetheinputdatatotheoutput.Allthetraversedbubbles belong to the transaction.
- Thesebubblesshouldbemappedtothesamemoduleonthestructure chart.

Structure chart of RMS calculator:



12. DESCRIBE DETAILED DESIGN

Detailed Design

- Pseudo code description of the processing and the different data structure are designed for the different modules of the structure chart
- Usually described in the form of Module specification(MSPEC)
- MSPEC written in structured English

11. EXPLAIN OBJECT ORIENTED DESIGN AND ITS COMPLEXITY METRICS

B. OBJECT- ORIENTED SOFTWARE DESIGN

- Viewed as collection of objects.
- Objects are associated with set of functions called methods.
- OOD makes extensive use of principles of data abstraction and decomposition.

UML (unified modeling language)

- Language for creating models
- Have its own syntax
- UML can be used to document object oriented analysis and design results obtained using any methodology.
- Many notationsare difficult to draw on paper and are best drawn using CASE tools such as Rational Rose,Magic Draw etc.

MODEL

- A model is an abstraction of real problem(or situation) and is constructed by leaving out unnecessary details. This reduces the problem complexity and makes it easy to understand the problem
- The model can be used for the following purposes::
  1. Analysis
  2. Specification
  3. Design
  4. Coding
  5. Visualization
  6. Testing

UML DIAGRAMS

- UML can be used to construct 9 different types of diagrams to view five different views of system.
- If a single model is made to capture all the required perspective, then it would be as complex as the original problem, and would be of very little use.
- UML diagrams can capture the following views (models) of a system:
  - Users' view
  - Structual view
  - Behavioural view
  - Implementation view
  - Environmental view

1. Users' view: This view defines the functionalities made available by the system to its users.
The users' view captures the view of the system in terms of the functionalities offered by the system to its users.

2.**Structural View:**

- Defines the structure of the problem in terms of objects.
- Also called as static model

3. **BehaviouralView:**

- Capture how objects interact with each other in time to realize the system behaviour.
- Dynamic model of the system

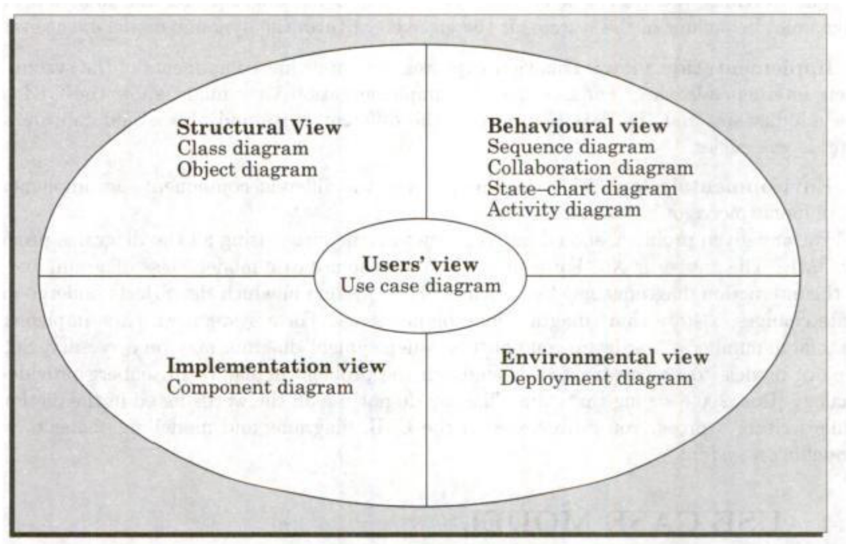


Figure 7.13: Different types of diagrams and views supported in UML.

4. **ImplementationView** :

- Captures important components of the system and their interdependencies.

5.**Environmentalview** :

This view models how different components are implemented on different pieces of hardware.

UML-UNIFIED MODELING LANGUAGE

UML Diagrams

- Diagrams are the heart of UML.
- These diagrams are broadly categorized as structural and behavioral diagrams.
- Structural diagrams -static diagrams like class diagram, object diagram etc.
- Behavioral diagrams- dynamic diagrams like sequence diagram, collaboration diagram etc.

1. USECASE MODEL

- Represent the different ways in which a system can be used by the users.
- Consists ofset of use cases
- Asimplewaytofindalltheusecasesofasystemistoaskthequestion: “Whattheuserscandousingthe system?"
- Usecasescorrespondtothehigh-levelfunctionalrequirements.

Purposeof use cases

- Ausecasetypicallyrepresentsasequenceofinteractionsbetweentheuserandthesystem.

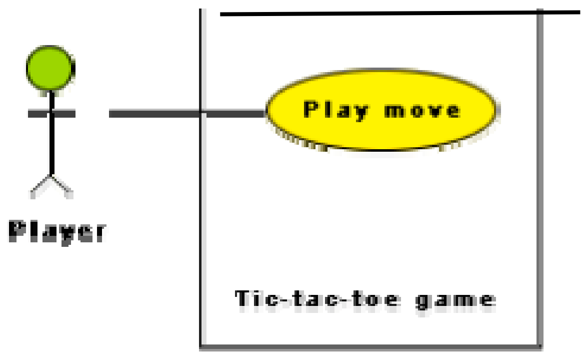
Representationof use cases

- Usecasescanberepresentedbydrawingausecasediagramandwritinganaccompanyingtext elaboratingthedrawing.
- Intheusecasediagram,eachuse caseisrepresentedbyanellipsewiththenameoftheusecasewritteninside theellipse.
- Alltheellipses(i.e.usecases)ofasystemareenclosedwithinarectanglewhichrepresentsthe

systemboundary.

- Thenameofthesystembeingmodeled (such as Library InformationSystem) appears inside the rectangle.
- Thedifferentusersofthesystemarerepresentedbyusingthestickpersonicon.
- Eachstickpersoniconisnormallyreferredtoasanactor.Anactorisaroleplayedbyauserwith respecttothesystemuse.
- Itispossiblethatthesameusermayplaytheroleofmultipleactors.
- Eachactorcanparticipateinoneormoreusecases.
- Thelineconnectingtheactorandtheusecaseiscalledthe communication relationship.
- It indicatesthat the actor makes use of the functionalityprovidedbytheusecase.
- Boththehumanusersandtheexternalsystemscanberepresentedbystickpersonicons.
- Whenastickpersoniconrepresentsanexternalsystem,itisannotatedbythestereotype c o n s t r u c t <<external system>>.

Use case model for tic tac toe game



TextDescription

- Eachellipseontheusecasediagramshouldbeaccompaniedbyatext description.
- The textdescription should define the details of the interactionbetweentheuserandthecomputerandotheraspectsoftheusecase.
- Thefollowingare some ofthe information which may be included in a use case text description

Contact persons: This section liststhepersonnel oftheclient organizationwithwhomtheusecasewasdiscussed,dateandtimeofthemeeting,etc.

Actors: Inadditionto identifying the actors, some information about actorsusingthisusecasewhichmayhelptheimplementationoftheusecasemayberecorded.

Pre-condition: Thepreconditionswoulddescribethestateofthesystem before the use case executionstarts.

Post-condition: Thiscapturesthestateofthesystemaftertheusecasehassuccessfullycompleted.

Non-functional requirements : Constraints such as platform and environment conditions, qualitative statements, responsetimerequirements,etc.

Exceptions, error situations: Errors that are not domain related, such as software errors,neednotbediscussedhere.

Sampledialogs: Theseserveasexamplesillustratingtheusecase.
Specificuser interfacerequirements: Thesecontainspecificrequirementsfortheuserinterfaceoftheusecase.Forexample,itmaycontainformstobeused,screenshots,interactionstyle,etc.
Documentreferences: This part contains reference storelated documents which may be useful to understand the system operation.

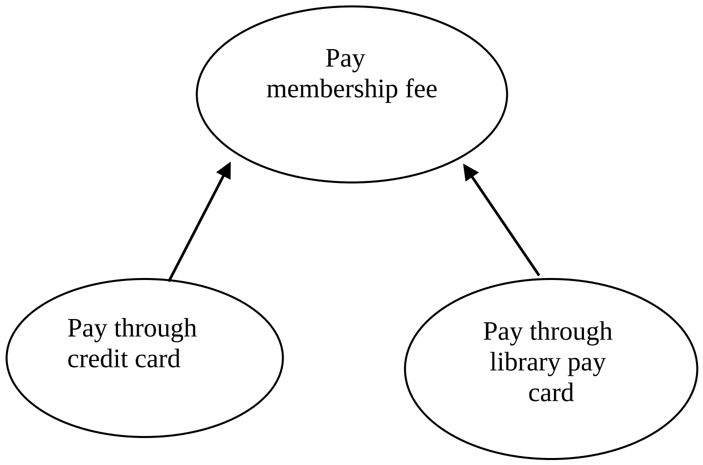
Factoring of use cases

- It is often desirable to factor usecases into component usecases.
- Actually, factoring of usecases are required under two situations. First, complex usecases need to be factored into simpler usecases.
- Secondly, usecases need to be factored whenever there is common behaviour across different use cases
- Factoring would make it possible to define such behaviour only once and reuse it when ever required
- UML offers three mechanisms for factoring of usecases as follows:

Generalization

- Usecase generalization can be used when one usecase that is similar to another, but does something slightly differently or something more.
- Generalization works the same way with use cases as it does with classes.

Representation of use case generalization

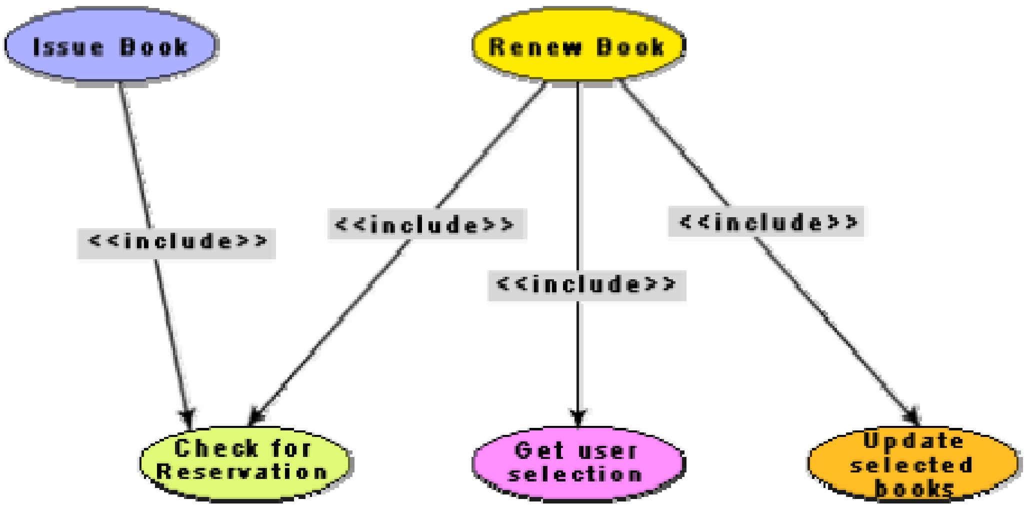


Includes

- The includes relationship in the older versions of UML (priortoUML1.1) was known as the uses relationship.
- The includes relationship involves one usecase including the behaviour of another usecase in its sequence of events and actions.

Representation of use case inclusion

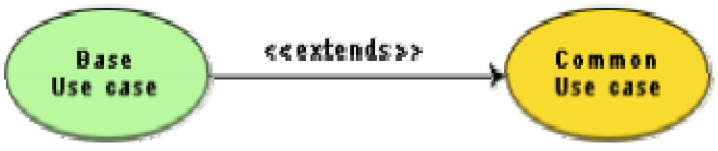
<<include>>



Extends

- Themainideabehindtheextendsrelationshipamongtheusecasesis thatitallowsyoutoshowoptionalsystembehaviour.
- Anoptionalsystem behaviour isextended only under certain conditions.
- The extends relationship is similar to generalization.
- But unlike generalization,theextendingusecase canaddadditional behaviour onlyat an extensionpoint only when certain conditions are satisfied.
- Theextensionpointsarepointswithintheusecasewherevariationto the mainline(normal)actionsequencemayoccur.
- Theextendsrelationshipis normallyusedtocapturealternatepathsorscenarios.

Example of use case extension



Structural view

1. **Class Diagrams**

- Describes the static structure of the system
- Shows system structure rather than how it behaves.
- Comprises a number of class diagrams and their dependencies.

- The main constituents of a class diagram are classes and their relationships:generalization,aggregation, association and various kinds of dependencies.

Classes

- Represent entities with common featuresi.e, attributes and operations.
- Represented using solid outline rectangles with compartments.
- Witten in bold face
- Mixed case convention and begins with an uppercase.

Attributes

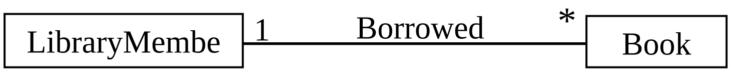
- Named property of a class.
- Represent kind of data that an object might contain.
- Attributes names followed by square brackets containing a multiplicity expression.
- An attribute without square brackets must hold exactly one value.
- The type of an attribute is written by following the attribute name with a colon and a type name.

Operation

- Names typically left justified, in plain type and begin with lower case letter.
- Abstract operations are written in italics.

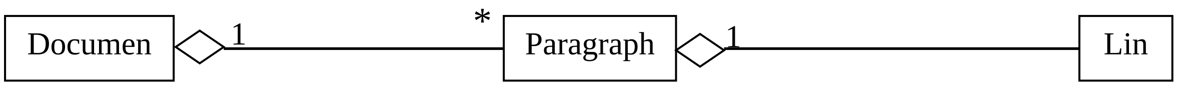
Association

- Represent by drawing straight line between the concerned classes.
- The name of the association is written alongside the association line.
- An arrowhead may be placed on the association line to indicate the reading direction of the association.
- The multiplicity indicates how many instances of one class are associated with other.
- An asterisk is used as a wild card and means many(zero or more).



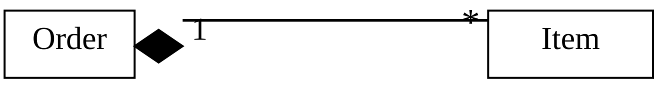
Aggregation

- Special type of association relation where the involved classes are not only associated to each other , but a whole part relationship exists between them.
- Not only knows the address of its parts, but also invokes the responsibility of creating and destroying its parts.



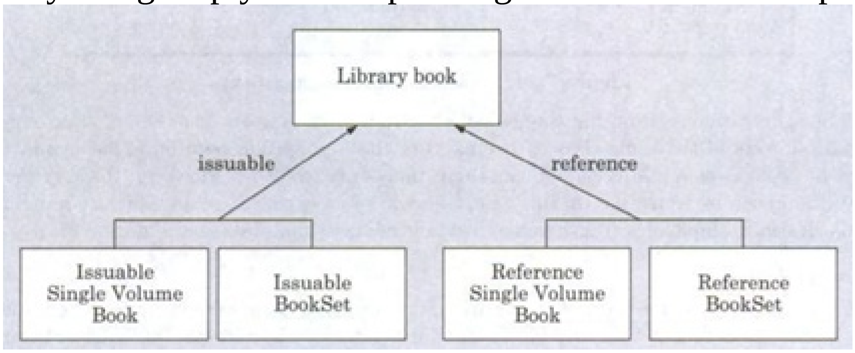
Composition

- Stricter form of aggregation.
- The life of parts cannot exist outside whole.



Inheritance

- Represented by using empty arrows pointing from sub class to super class



Dependency

- Represented by dotted arrows

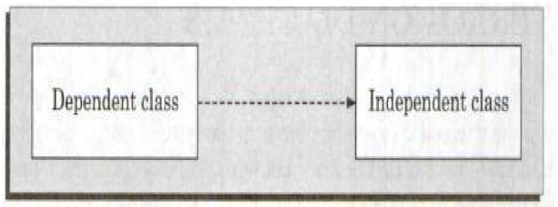


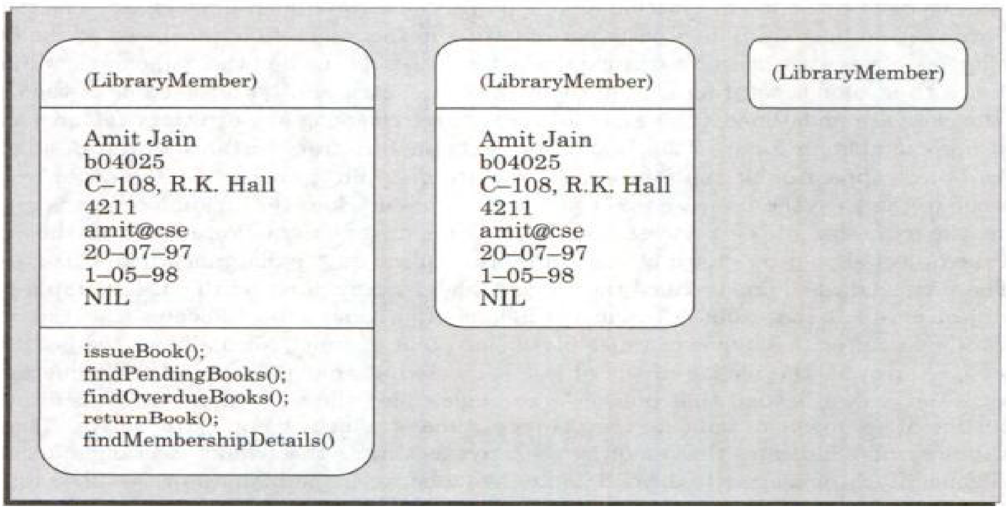
Figure 7.27: Representation of dependence between classes.

Constraints

- Describes a condition or an integrity rule
- to specify post- and pre- conditions for operations.

2. **Object Diagrams**

- Shows snap shot of objects at a point of time.
- Often called as instance diagram.
- Objects are drawn using rounded rectangles.



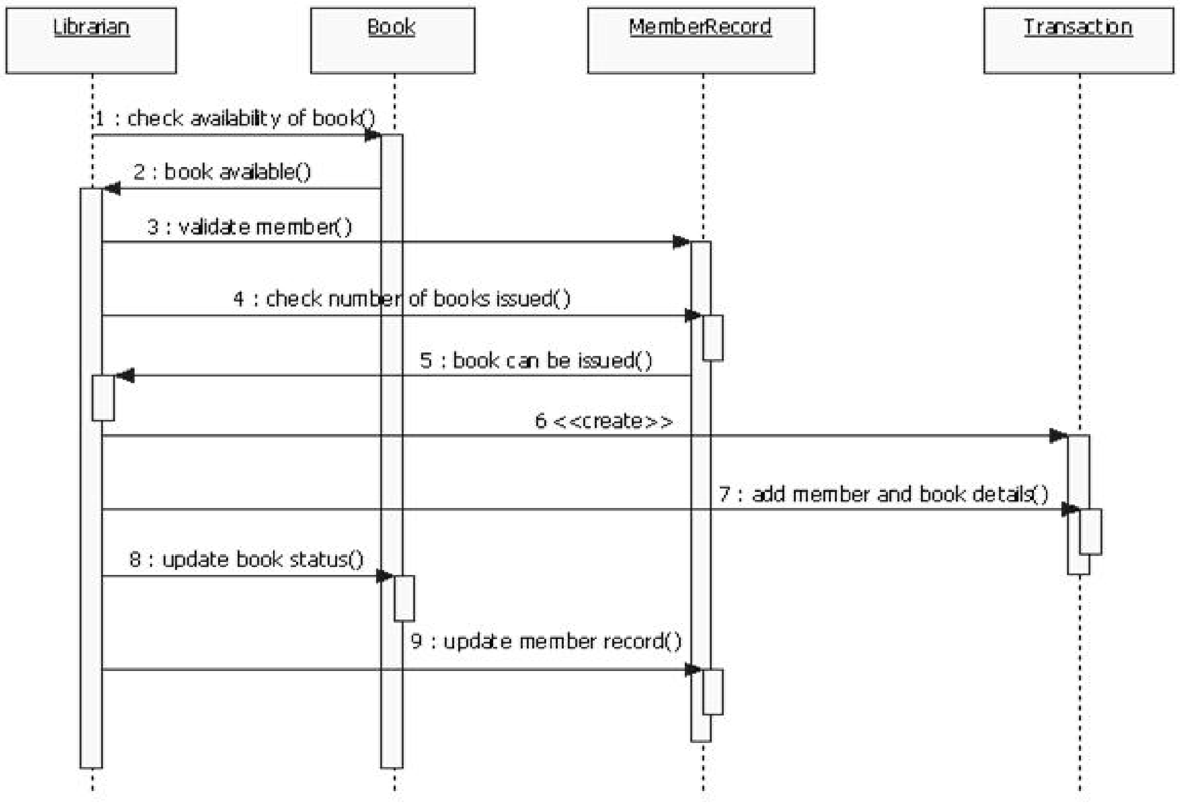
<u>Behavioural view</u>

1. **Interaction Diagrams**

- Realizes the behaviour of single use case
- shows the number of example objects and the messages that are passed between the object within the use case.
- Two kinds of interaction diagrams : sequence diagram and collaboration diagram
- Two diagrams are equivalent in sense that any one diagram can be derived automatically from the other.

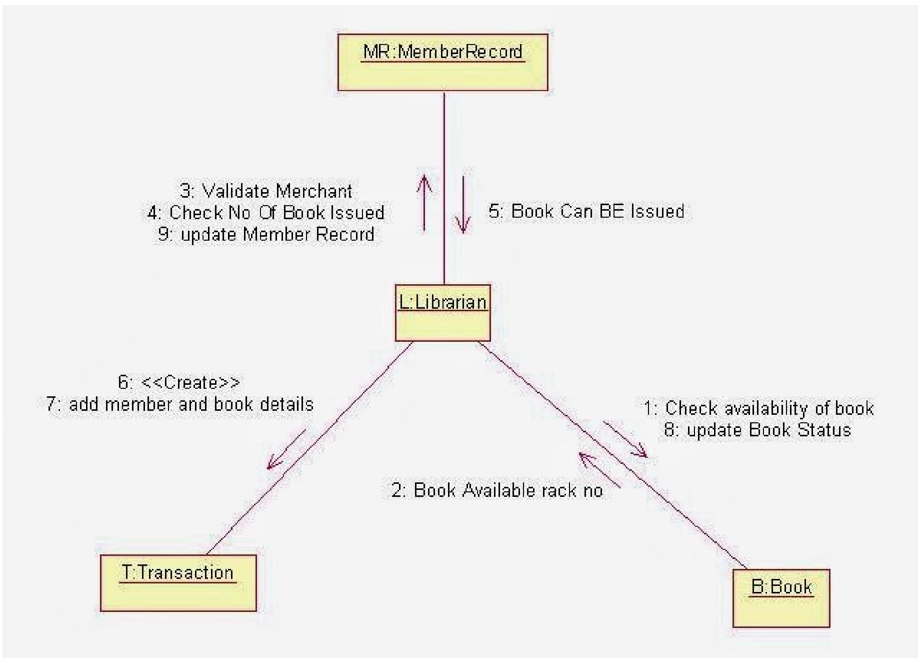
1. **Sequence Diagram**

- Shows interaction among objects as two – dimensional chart
- Shows behavioural aspect.
- Chart is read from top to bottom
- Objects participating in the interaction diagram are shown at the top of the chart as boxes attached to a vertical dashed line .
- Object is written with a colon



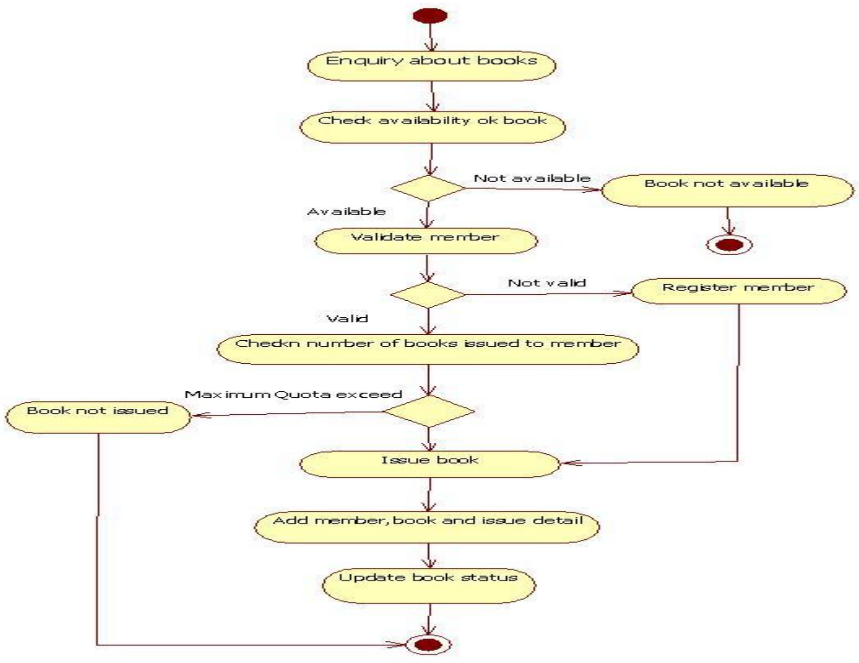
2. **Collaboration diagram**

- A **collaboration diagram** is a type of visual presentation that shows how various software objects interact with each other within an overall IT architecture and how users can benefit from this **collaboration**.
- A **collaboration diagram** often comes in the form of a visual **chart** that resembles a flow**chart**.



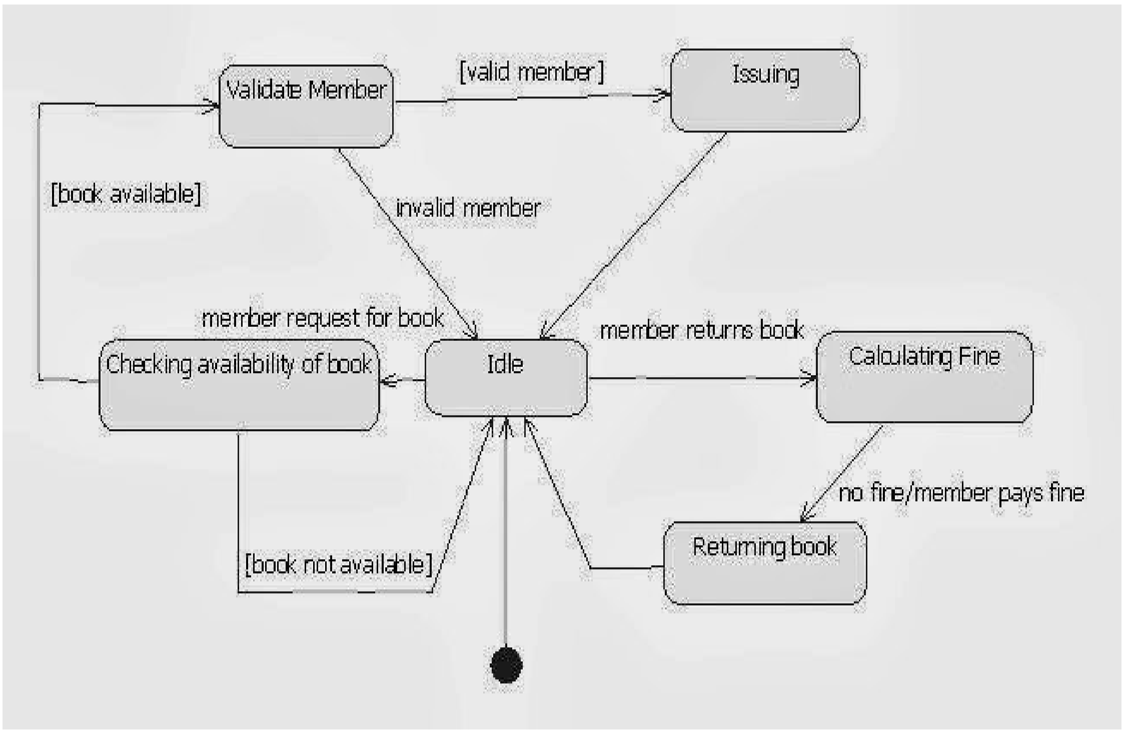
3. Activity Diagram

- Focus on representing various activities or chunks of processing and their sequence of activation.
- Similar to procedural flow chart.
- It support description of parallel activities and synchronization aspects involved.
- Employed in business process modeling.
- Carried out during initial stages of requirement analysis and specification.
-



4. State Chart Diagram

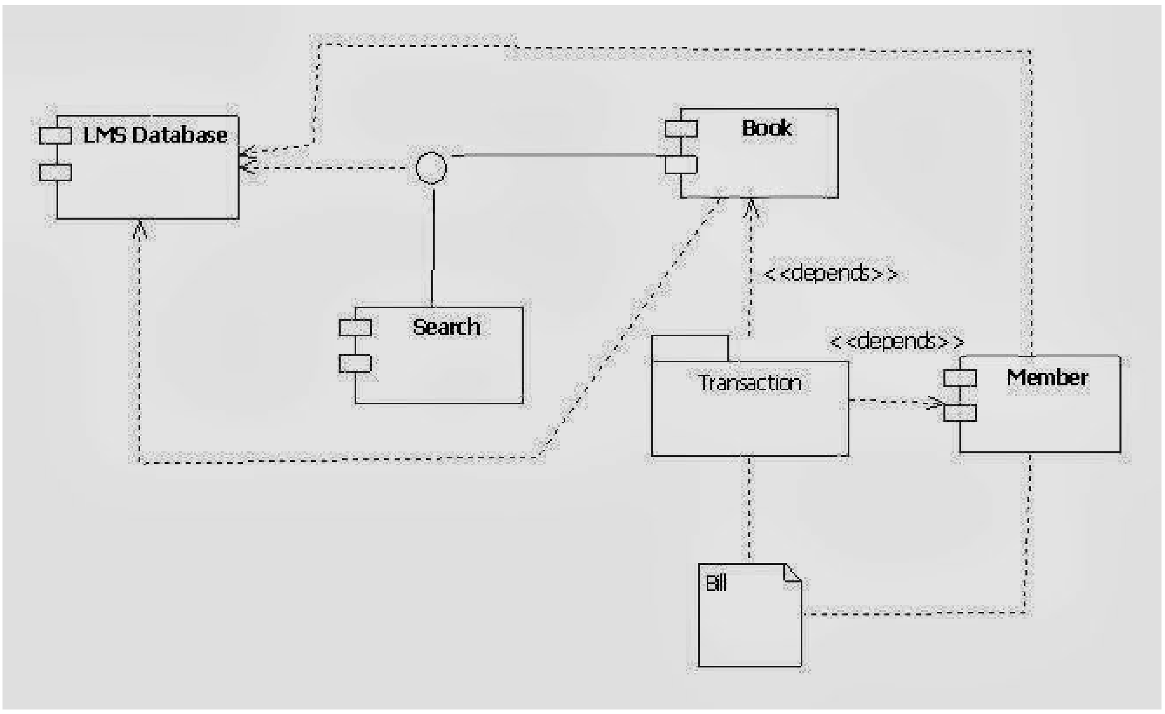
- Used to model how the state of an object changes in its life time
- Describes how the behaviour changes in several use case execution
- Based on Finite State machine (FSM) formalism
- Major disadvantage is that state explosion problem
- Basic elements in the state chart diagram is as follows :
  1. Initial state : represented as filled circle
  2. Final State : This is represented by a filled circle inside a larger circle
  3. State : These are represented by rectangles with rounded corners
  4. Transition : arrow between two states



Implementation view

1. **Component diagram**

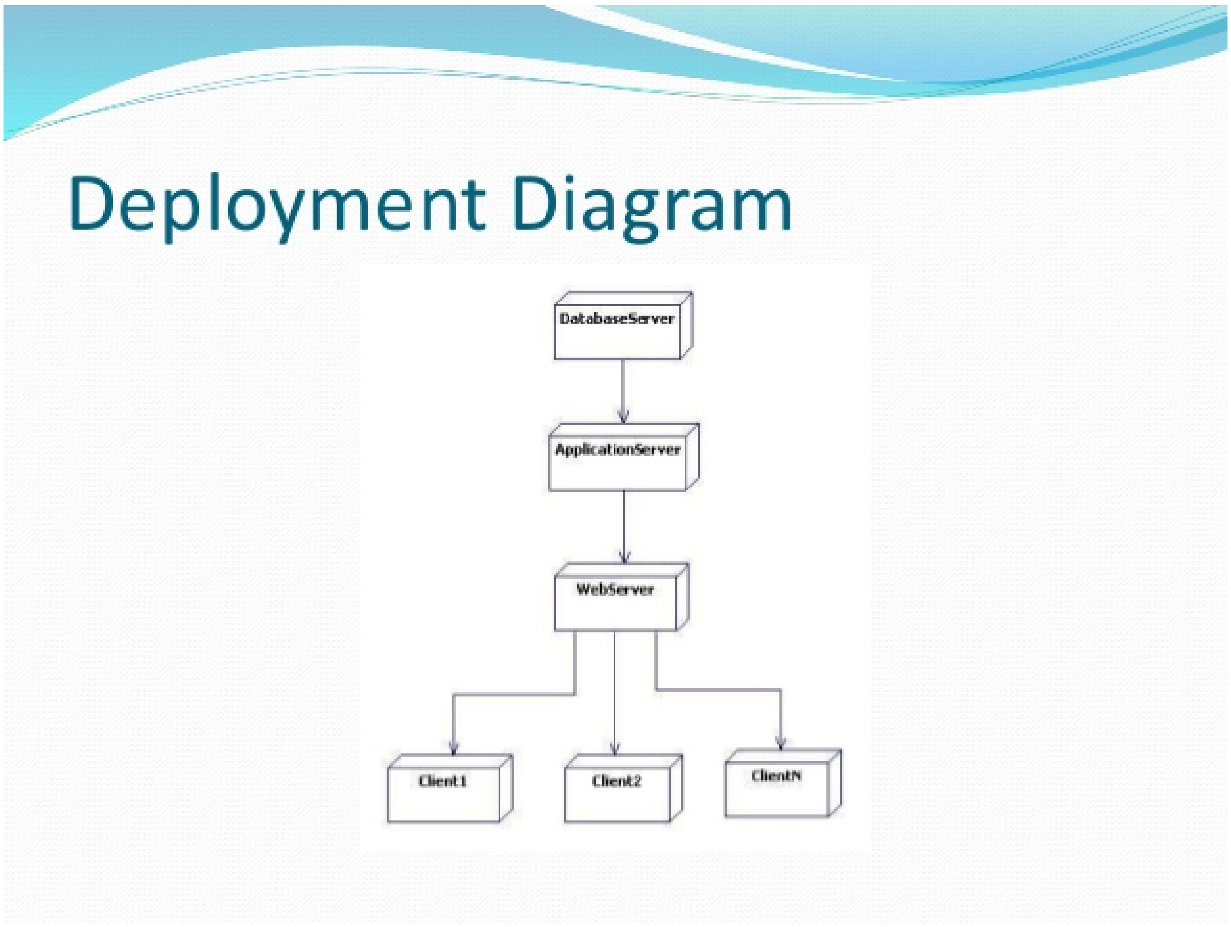
- **Component diagram** is a special kind of **diagram** in UML. The purpose is also different from all other **diagrams** discussed so far.
- It does not describe the functionality of the system but it describes the **components** used to make those functionalities.



Environment view

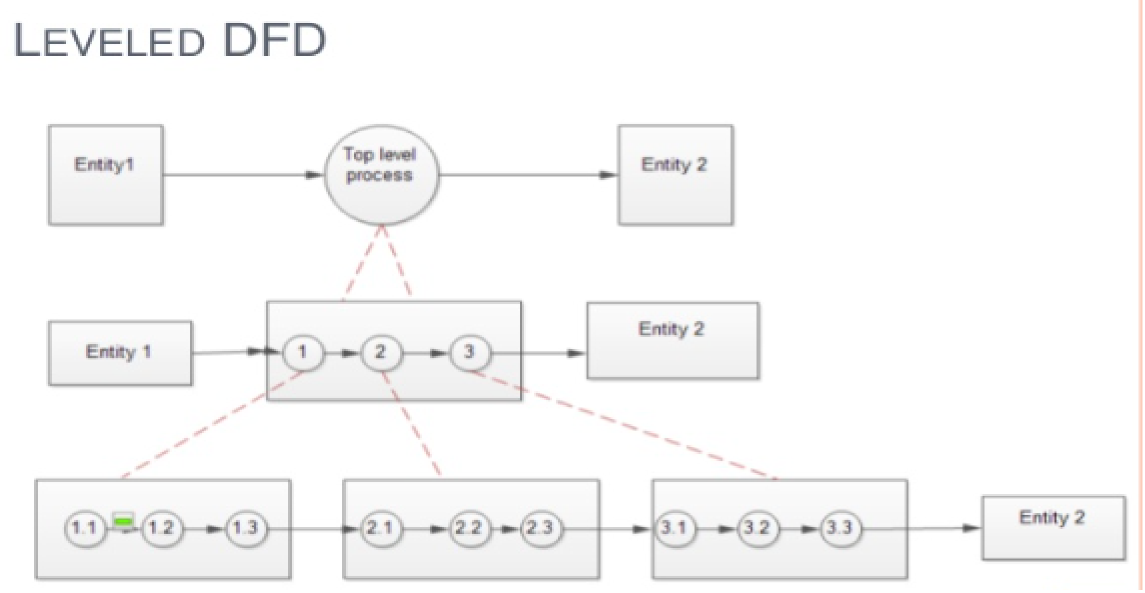
1. **Deployment diagram**

- A **deployment diagram** is a UML **diagram** type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them.
- **Deployment diagrams** are typically used to visualize the physical hardware and software of a system.



Deployment Diagram

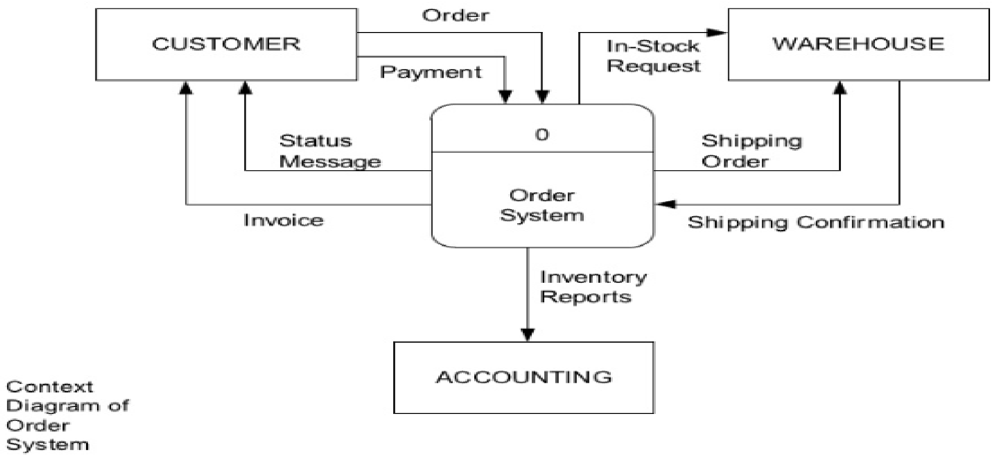
LEVELED DFD



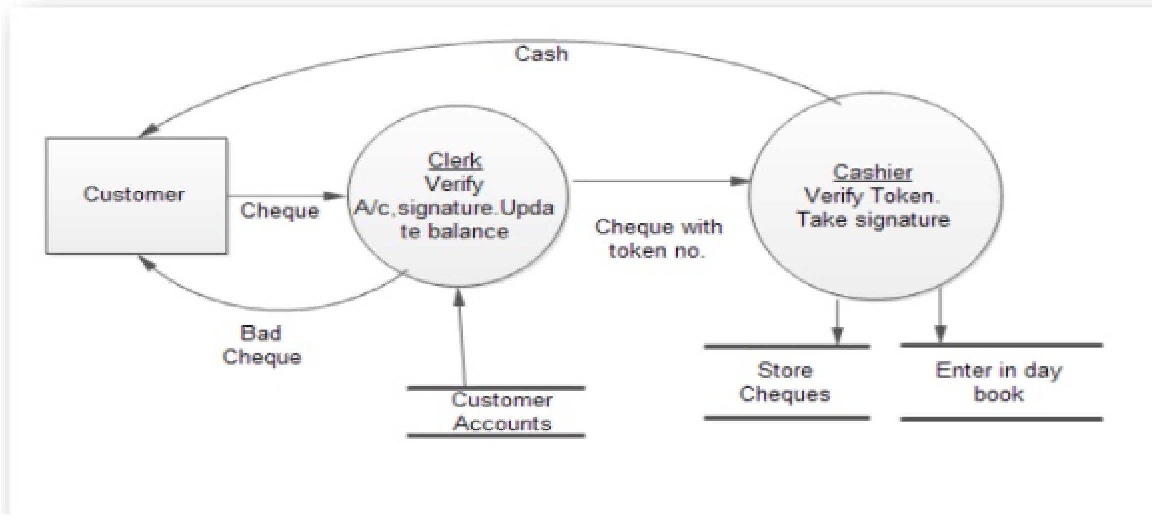
Leveled DFD

EXAMPLES:-

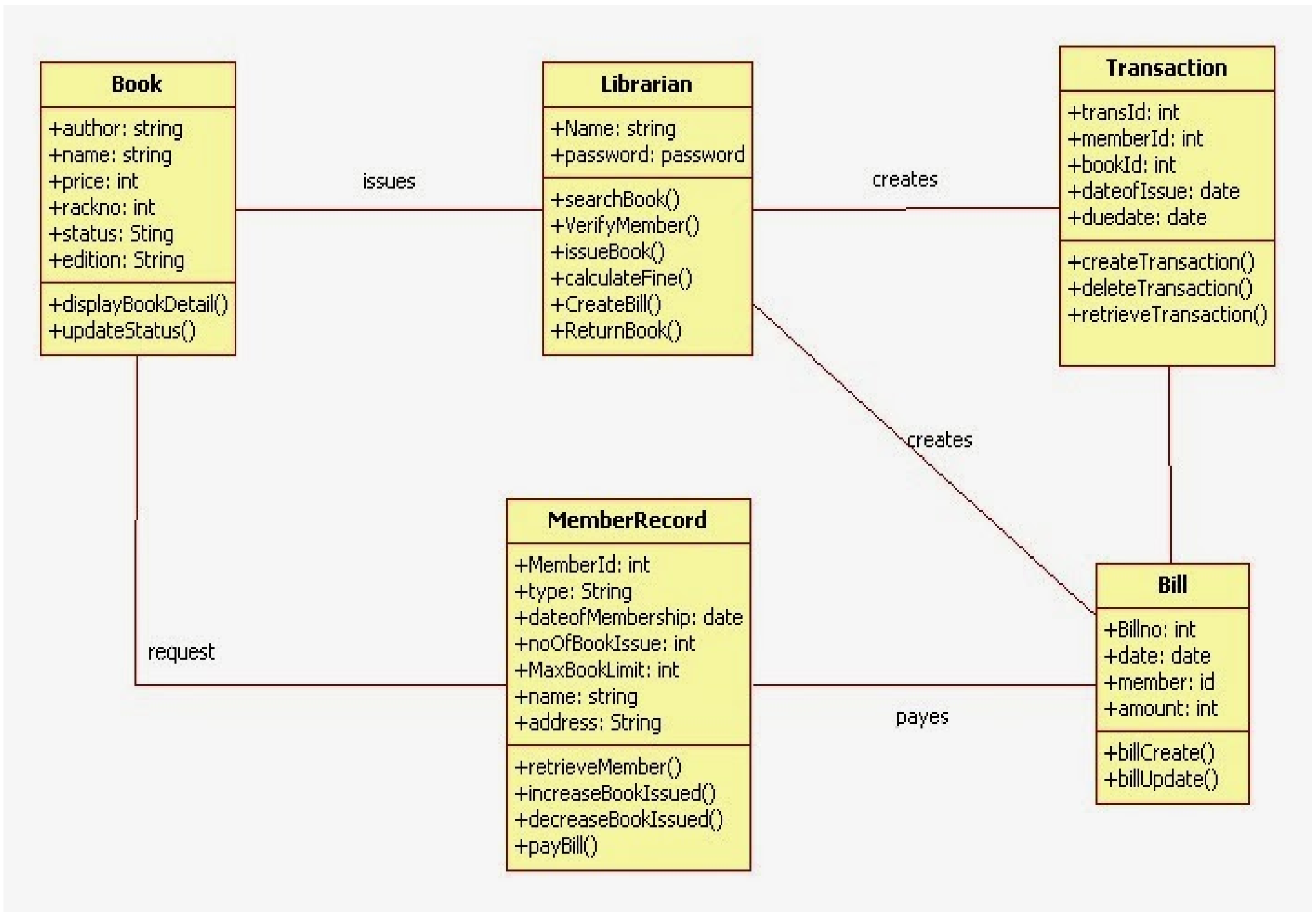
DFD:- ORDER MANAGEMENT-



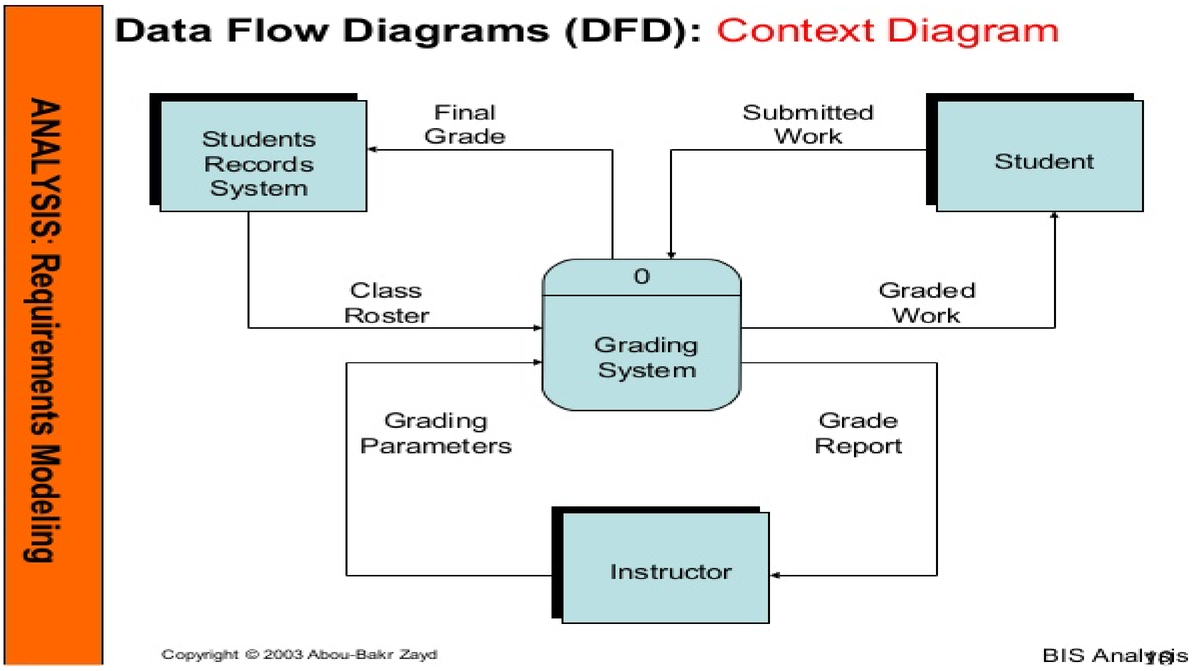
EXAMPLE 2”-



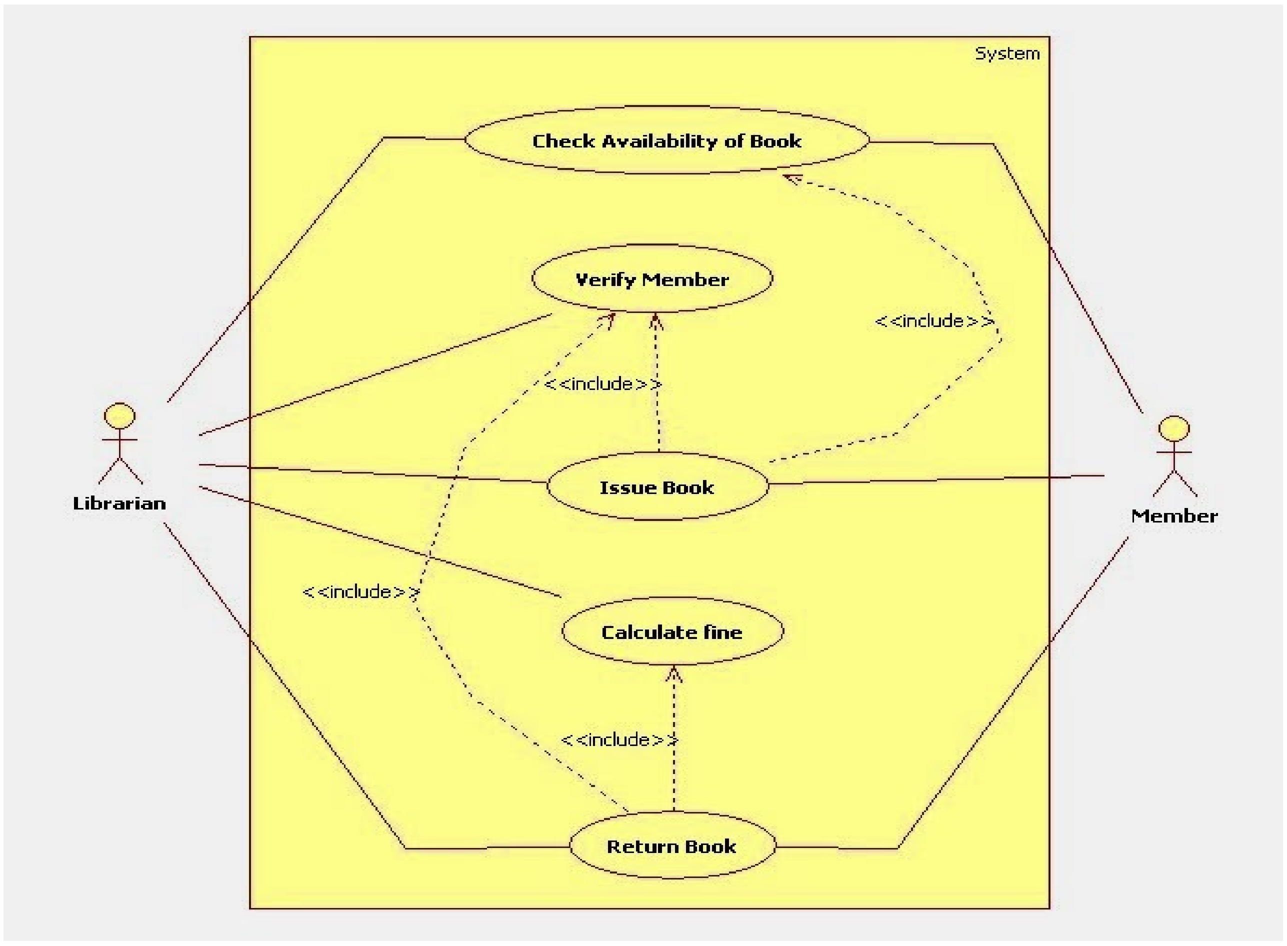
EXAMPLE :-CLASS DIAGRAM – LIBRARY MANAGEMENT SYSTEM



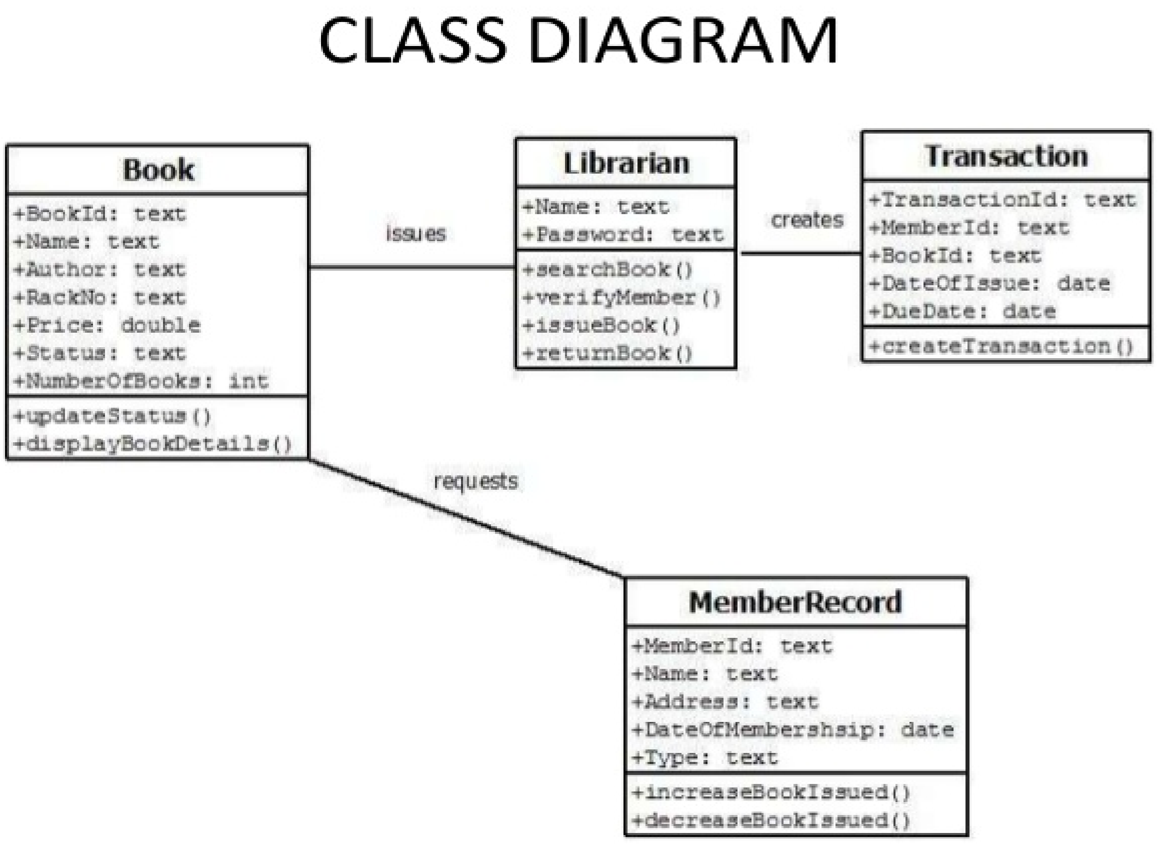
EXAMPLE – DFD- GRADING SYSTEM



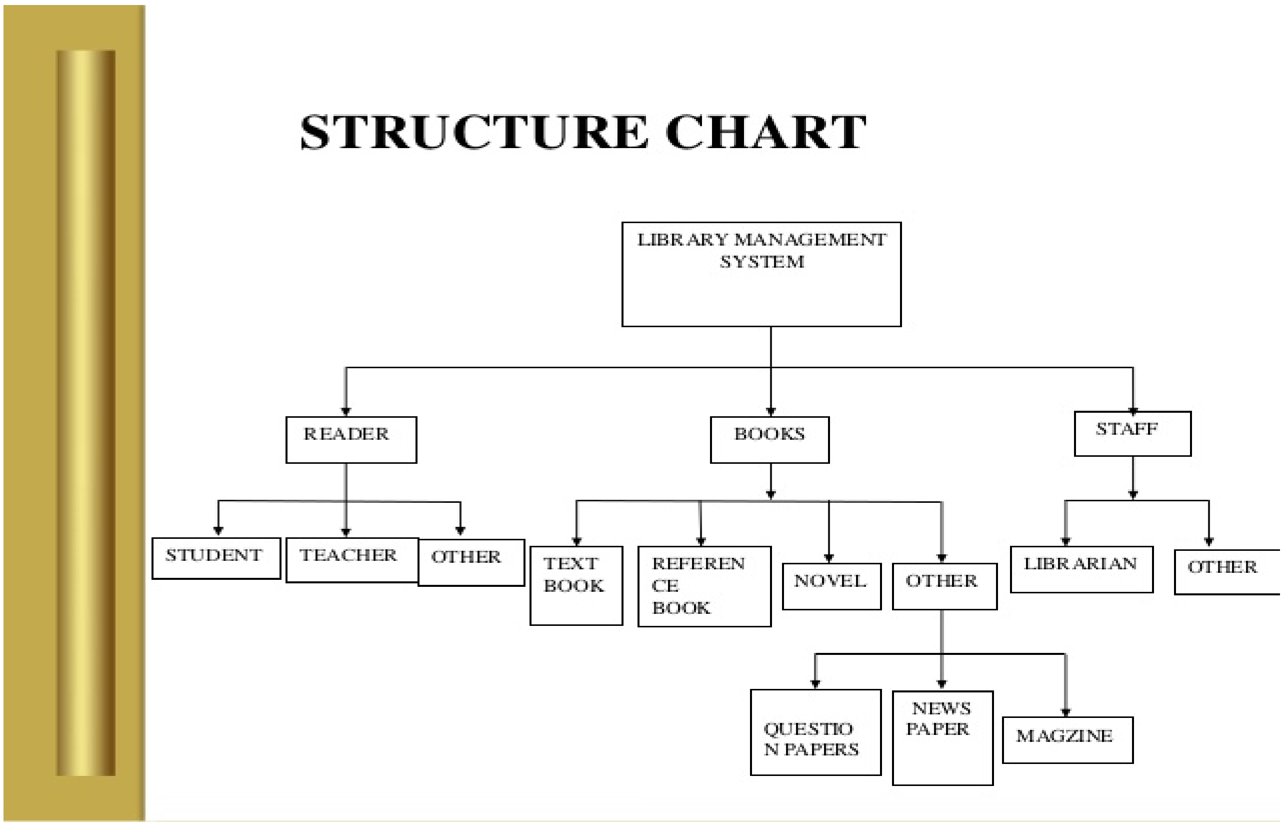
EXAMPLE- USECASE DIAGRAM-LIBRARY MANAGEMENT



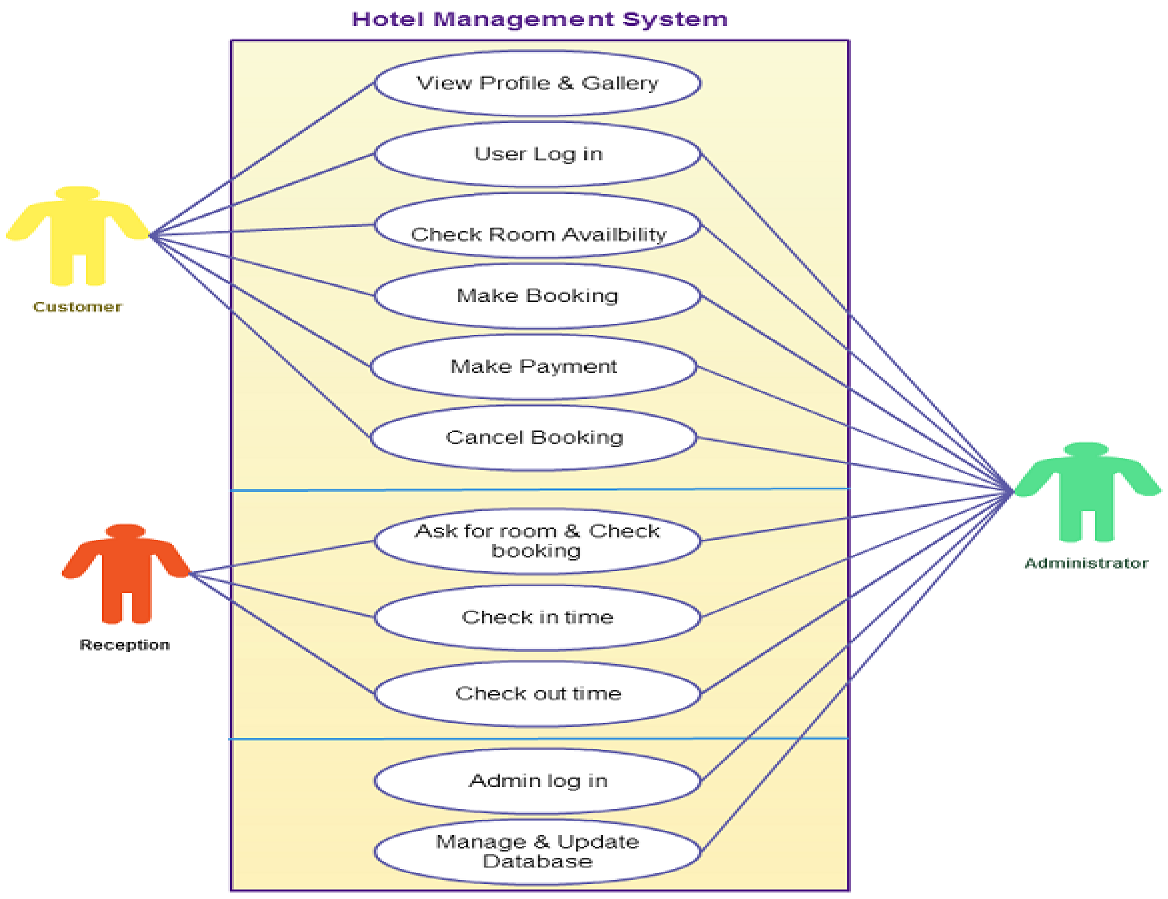
EXAMPLE –CLASS DIAGRAM-LIBRARY MANAGEMENT



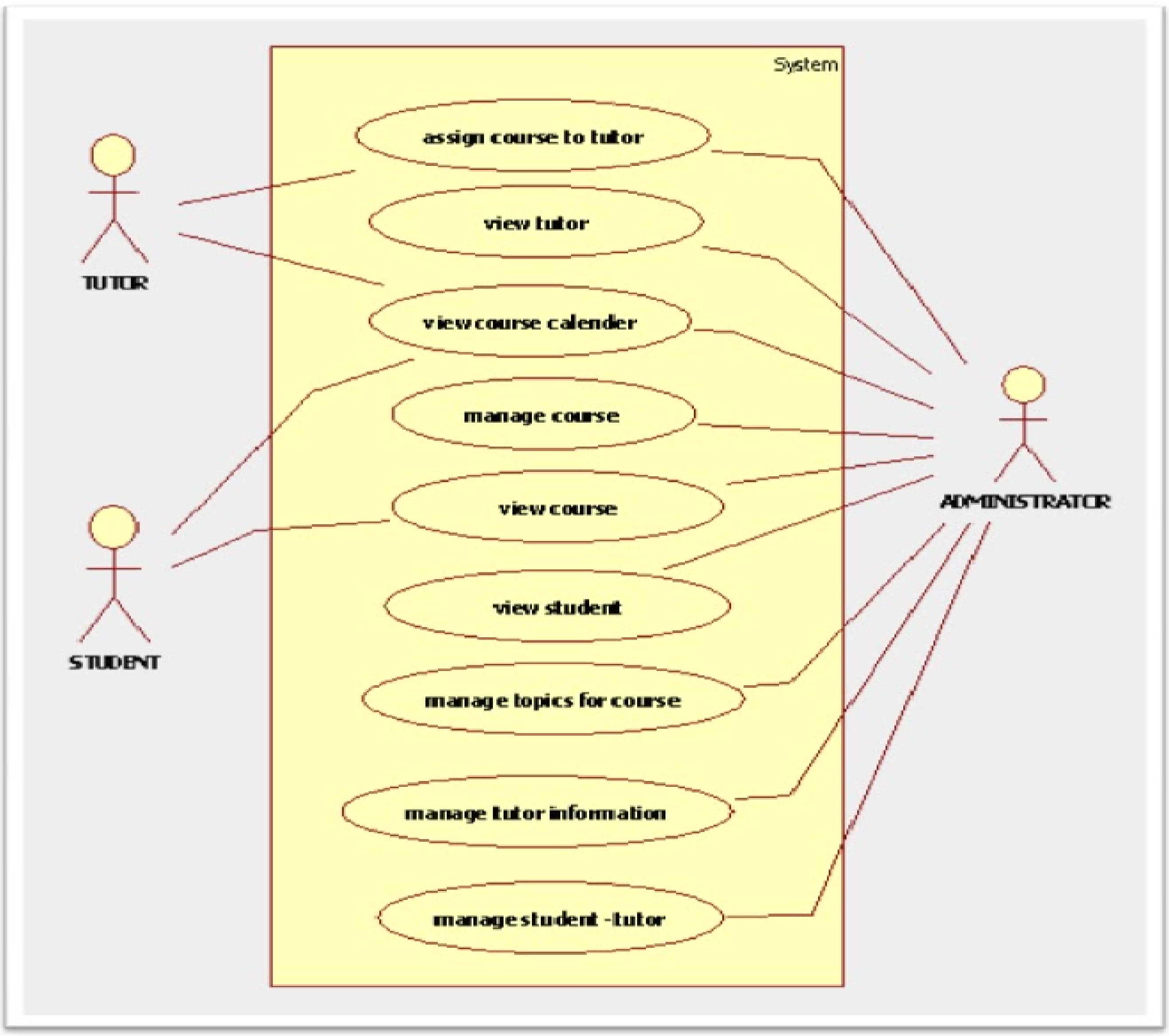
EXAMPLE- STRUCTURE CHART-LIBRARY MANAGEMENT



EXAMPLE –USECASE DIAGRAM-HOTEL MANAGEMENT



EXAMPLE-USECASE DIAGRAM-TUTOR STUDENT SYSTEM


System
assign course to tutor
view tutor
view course calender
manage course
view course
view student
manage topics for course
manage tutor information
manage student -tutor
TUTOR
STUDENT
ADMINISTRATOR