Charact. of good S/w design

- **Correctness:**
- **Understandability:**
- **Efficiency:**
- **Maintainability:**

Understandability

Complex design leads the cost of SDLC.

Two approaches:

- Modularity
- Layered

1). use consistent and meaningful names

2). should follow the principle of decomposition and abstraction.

Modularity

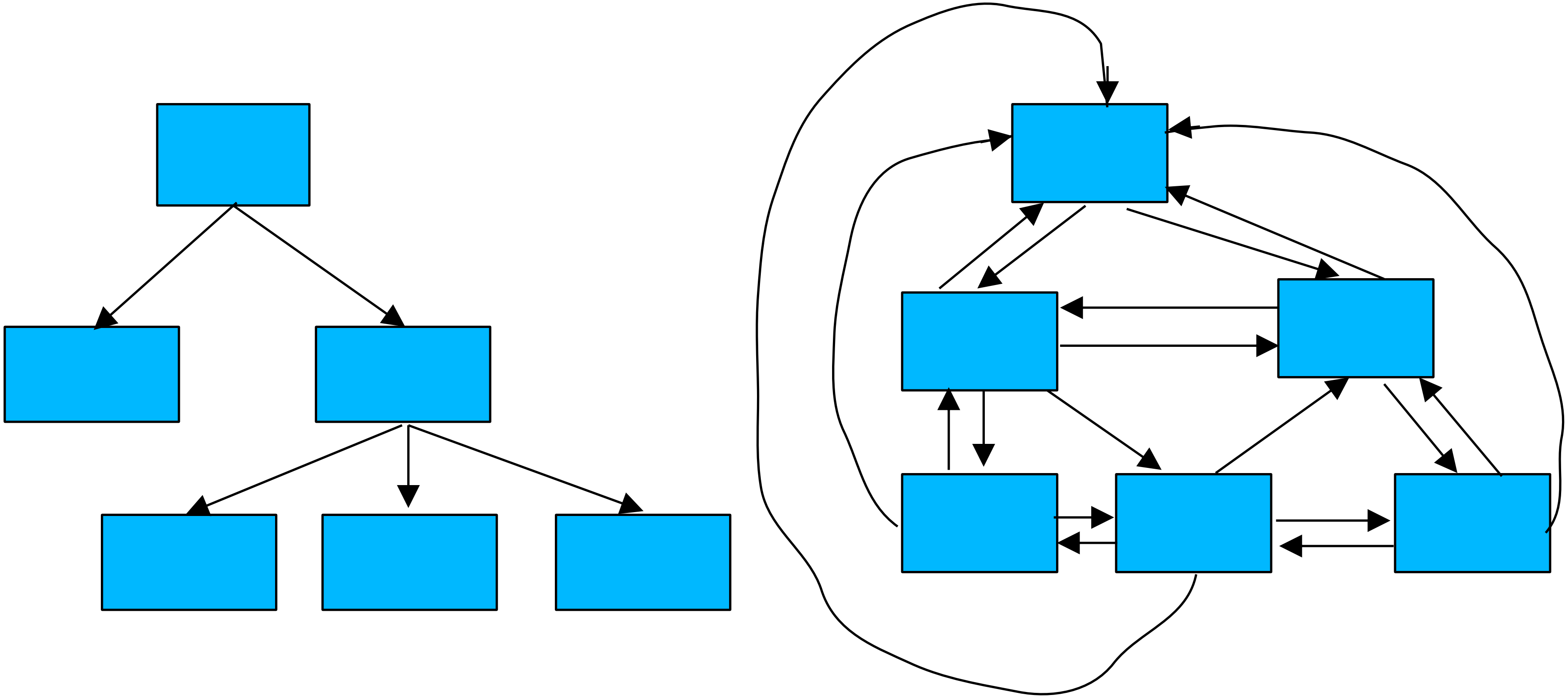
- Modularity is a fundamental attributes of any good design.
- Decomposition of a problem cleanly into modules:
  1. divide and conquer principle.
  2. Modules are almost independent of each other

Cont…

If modules are independent:

- modules can be understood separately.
- reduces the complexity greatly.

Cleanly and Poor Modularity



Cont…

In technical terms, modules should display:

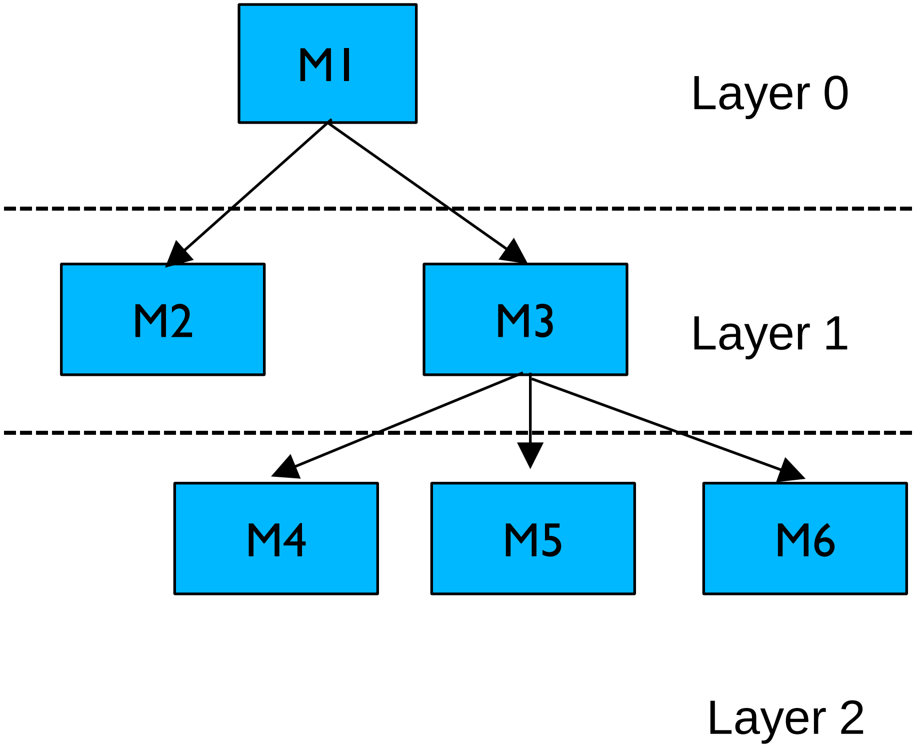
1. high cohesion
2. low coupling.

Layered Design

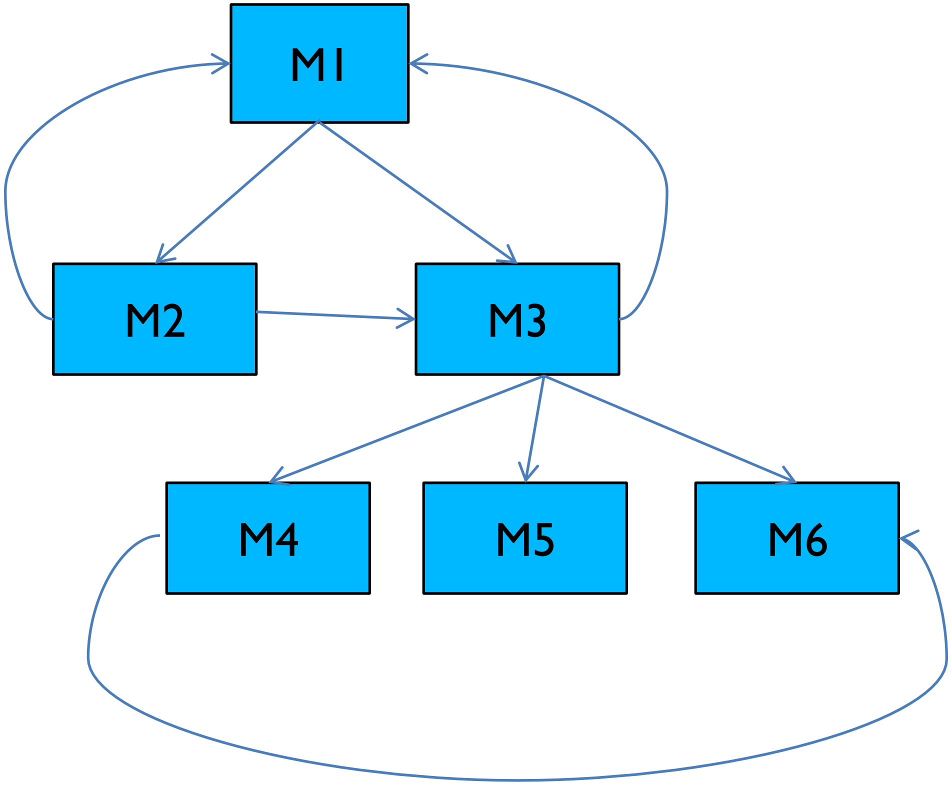
- Graphical representation of call relations and form hierarchy of layers.

- Lower layer is unaware of the higher layer module.
- Easy to understand and detect the source of error.
- Reduce the debugging time.
- It achieves control abstraction.

LAYERED ARRANGEMENT OF MODULES



Layered design with good Control abstraction



Layered design showing poor Control abstraction

Terminology of L.D.

1) Superordinate and Subordinate modules:
2) Visibility:
3) Control abstraction:
4) Depth and width:
5) Fan-out (low)and Fan-in(high):

Cohesion And Coupling

- **Good system design:** System decomposed into modules.
- **Good decomposition:** high cohesion and low coupling.

➢ Cohesion is a measure of the Functional strength of a module.

➢ Coupling is a measure of the degree of interaction between two modules.

Cohesion Cont…

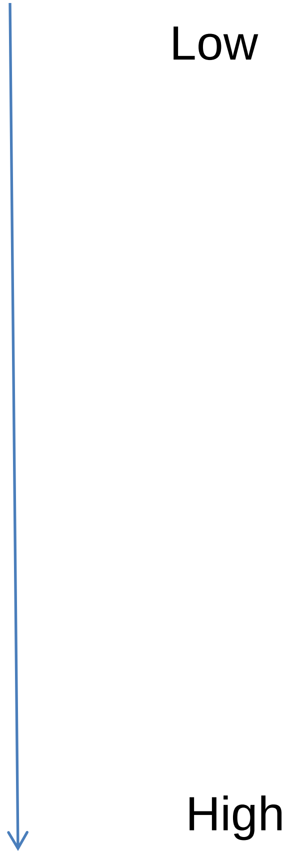
Less cohesion often correlates with high coupling and vice-versa. **How?**

- **high cohesive:** Modules cooperate with each other for performing single objective.
- **Functional independency**(highly cohesive and low coupling): error isolation, understandability and scope of reuse.

Classification of cohesiveness

Classes of cohesion:

Coincidental
Logical
Temporal
Procedural
Communicational
Sequential
Functional



Cohesion cont…

1. Coincidental Cohesion:

- A module performs set of tasks that relate to each other very loosely.
- Random collection of functions.

Cohesion cont…

2. Functional cohesion:

If different functions cooperate to complete a single task.

Example:

- Module Name: Manage-Book-Lending
- Function: issue-book, return-book, query book, Find-borrower.

Cohesion cont…

3. Logical Cohesion:

- All elements of the module perform similar operations such as error handling, data input, data output, etc.

- Example: various types of reports(time table).

Cohesion cont…

4. Temporal Cohesion:

- Functions of the module are executed in the same time span.
- Example: Booting of the system.

5. Procedural Cohesion:

- Set of functions of the module are executing one after the other.
- can differ in purpose and data.

Example: Place an order.

login -> place_order -> check_order

->Print_order-> Print_bill->logout.

6. communicational Cohesion:

- Functions refer to or update the same data structure.
- Example: module: student

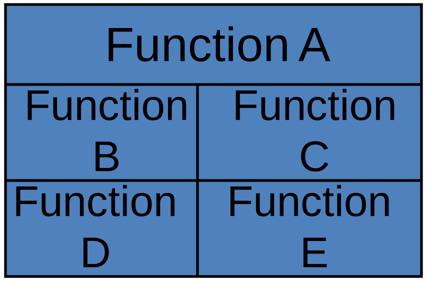
- Functions: admit_student, enter_marks, print grade report. etc all access same database.

7. Sequential Cohesion:

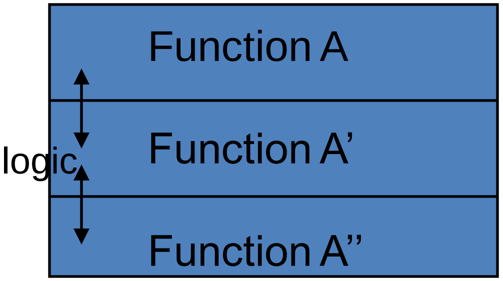
- Different functions of the module execute in a sequence, and output from one function is input to the next in sequence.

- Example: Attendance module
  - Enter list/ retrieve list
  - Mark attendance
  - View attendance

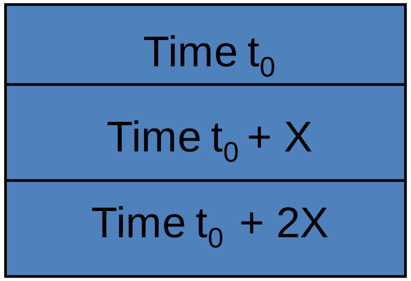
Examples of Cohesion-1



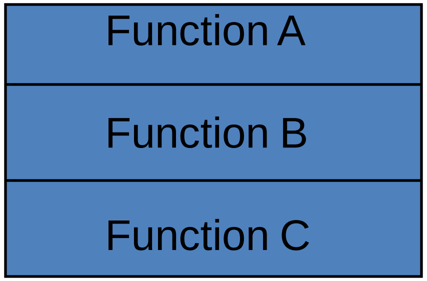
Coincidental
Parts unrelated



Logical
Similar functions

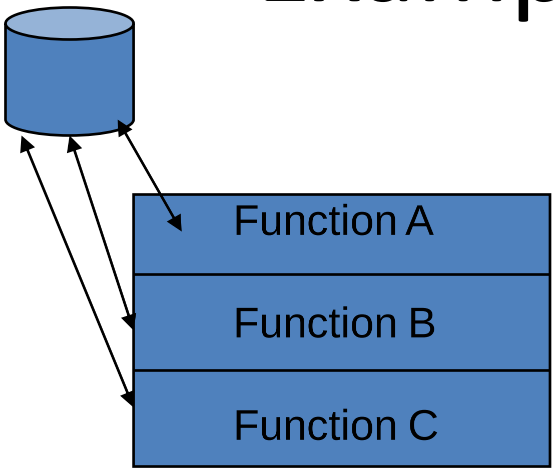


Temporal
Related by time

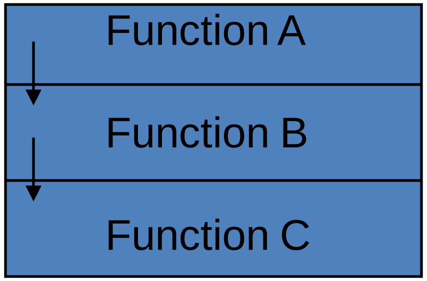


Procedural
Related by order of functions

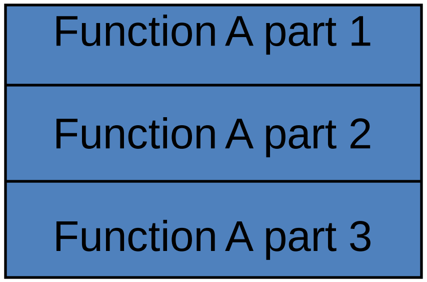
Examples of Cohesion-2



Communicational
Access same data



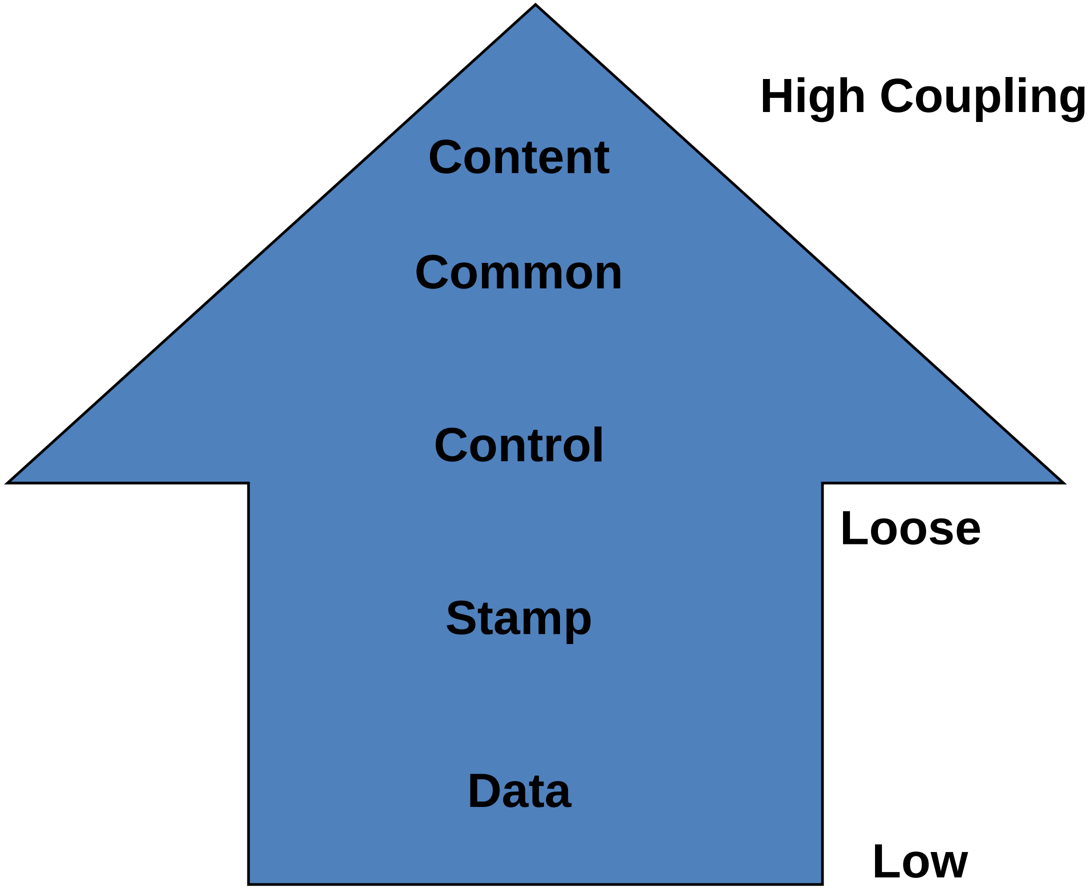
Sequential
Output of one is input to another



Functional
Sequential with complete, related functions

Classification of coupling

Range of Coupling



High Coupling

Content

Common

Control

Loose

Stamp

Data

Low

Data Coupling

- Definition: Two components are data coupled if there are homogeneous data items.
- Tramp data: data travelling across the modules before being used.
- **Example:** call by value and call by reference.

Stamp Coupling

- Definition: Component passes a data structure to another component that does not have access to the entire structure.

Example: in C str. and in PASCAL

- Composite data is passed between the modules.
- Requires second component to know how to manipulate the data structure.
- May be necessary due to efficiency factors

Example-1

Module: Customer billing

The print routine of the customer billing accepts a customer data structure as an argument, parses it, and prints the name, address, and billing information.

- **There is a trade-off between data coupling and stamp coupling. Increasing one often decreases the other.**

Control Coupling

- Definition: Component passes control parameters to coupled components.

Example: Module p calls module q by passing control flag that says to execute the set of instructions.

Common Coupling

Definition: Two components share data

- Global data structures
- Common blocks

Usually a poor design choice because

- Lack of clear responsibility for the data
- Reduces maintainability.
- Difficult to reuse components
- Reduces ability to control data accesses

Example-1

Process control component maintains current data about state of operation. Gets data from multiple sources. Supplies data to multiple sinks.

Data manager component is responsible for data in data store. Processes send data to and request data from data manager

Content coupling

- Definition: One component references contents of another.
- Example:
  - Mostly in low level languages.
  - Branch into another branch.
  - Component modifies another’s code, e.g., jumps into the middle of a routine

Example of Content Coupling-1

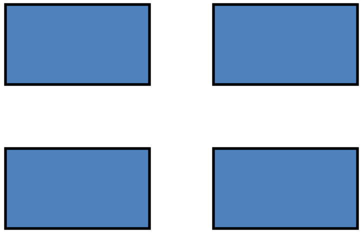
Part of program handles lookup for customer.

When customer not found, component adds customer by directly modifying the contents of the data structure containing customer data.

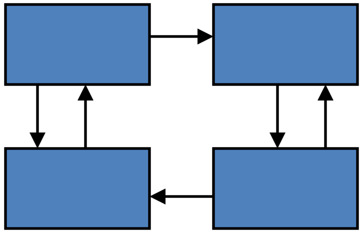
- When customer not found, component calls the AddCustomer() method that is responsible for maintaining customer data.

- High coupling leads to complexity and difficulty in the design ?

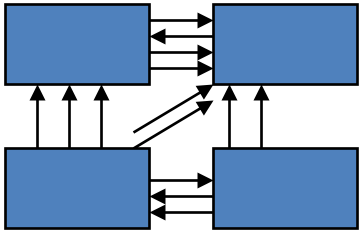
Coupling: Degree of dependence among components



No dependencies



Loosely coupled-some dependencies



Highly coupled-many dependencies

Function oriented design approach

- It is a mature technology

- Top-down decomposition
  - Black-box view
  - Provides certain services
  - Services are known as high-level functions

- Example: function: *create_new_library_member*
- Subfuctions:
  - *Assign_membership_number*
  - *Create_member_record*
  - *Print_bill*
  - Can be split into more detailed functions

Centralized system state

- System state can be defined as:
  - The values of certain data items that determine the response of the system to a user action or an external event.
  - Such data have global scope.
  - Example: set of books in library automation system

- Example:
  - *Create_new_member*
  - *Delete_member*
  - *Update_member_record*
  - All share data such as *member_records.*

Design approaches for function oriented design

- Structured design by Constantine and Yourdan [1979]
- Jackson’s structured design by Jackson [1975]
- Warnier_orr methodology [1977, 1981]
- Step-wise refinement by Wirth [1971]
- Hatley and Pirbhai’s methodology [1987]

Chapter 6

FUNCTION – ORIENTED SOFTWARE DESIGN

Data flow diagram
and
structure chart

Design Tech. : SA/SD

Overview of SA/SD Methodology

- Consists of two distinct activities called transformers.
- **During structured analysis:**
  - **functional decomposition takes place.**
- **During structured design:**
  - **mapped to a module structure.**

Structured Analysis

- **The results of structured analysis can be easily understood and reviewed even by ordinary customers:**
- **Based on principles of:**
  - **Top-down decomposition approach.**
  - **Divide and conquer principle:**

Data Flow Diagram(DFD)

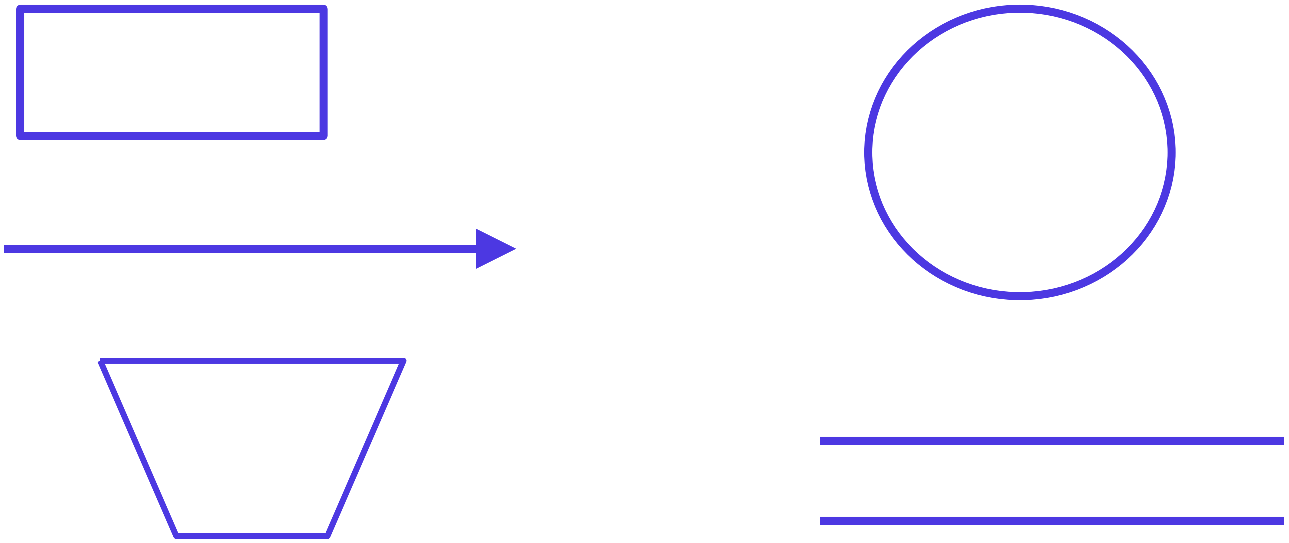
- **DFD is a hierarchical graphical model:**
  - shows the different functions (processes) of the system and
  - data interchange among the processes.
  - **Applicable to other areas also:**
    - e.g. for showing the flow of documents or items in an organization,

Data Flow Diagrams (DFDs)

- A DFD model:
  - uses limited types of symbols.
  - simple set of rules
  - easy to understand:
    - it is a hierarchical model.

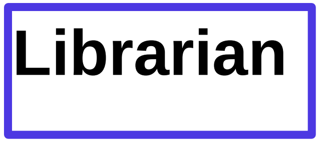
Data Flow Diagrams (DFDs)

- **Primitive Symbols Used for Constructing DFDs:**



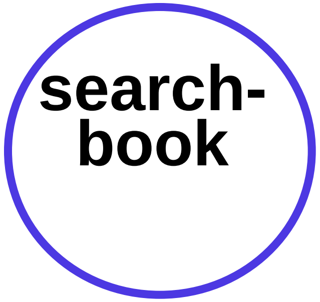
External Entity Symbol

- **Represented by a rectangle**
- **External entities are real physical entities:**
  - **input data to the system or**
  - **consume data produced by the system.**
  - **Sometimes external entities are called terminator, source, or sink, external h/w & s/w.**


Librarian

Function Symbol

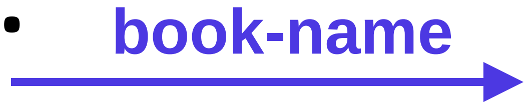
- **A function such as “search-book” is represented using a circle:**
  - **This symbol is called a process or bubble or transform.**
  - **Bubbles are annotated with corresponding function names.**



search-
book

Data Flow Symbol

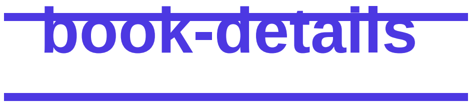
- **A directed arc or line.**
  - **represents data flow in the direction of the arrow.**
  - **Data flow symbols are annotated with names of data they carry.**



book-name

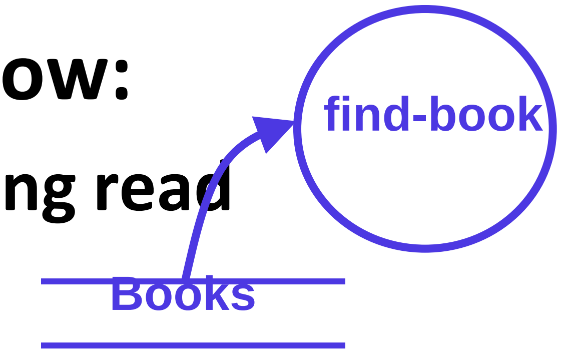
Data Store Symbol

- **Represents a logical file:**
  - **A logical file can be:**
    - **a data structure**
    - **a physical file on disk.**
  - **Each data store is connected to a process:**
    - **by means of a data flow symbol.**



Data Store Symbol

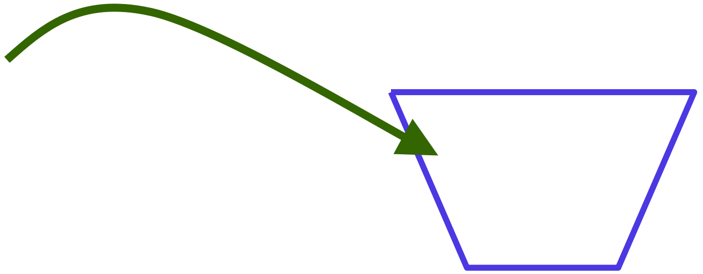
- **Direction of data flow arrow:**
  - **shows whether data is being read from or written into it.**



- **An arrow into or out of a data store:**
  - **implicitly represents the entire data of the data store**
  - **arrows connecting to a data store need not be annotated with any data name.**

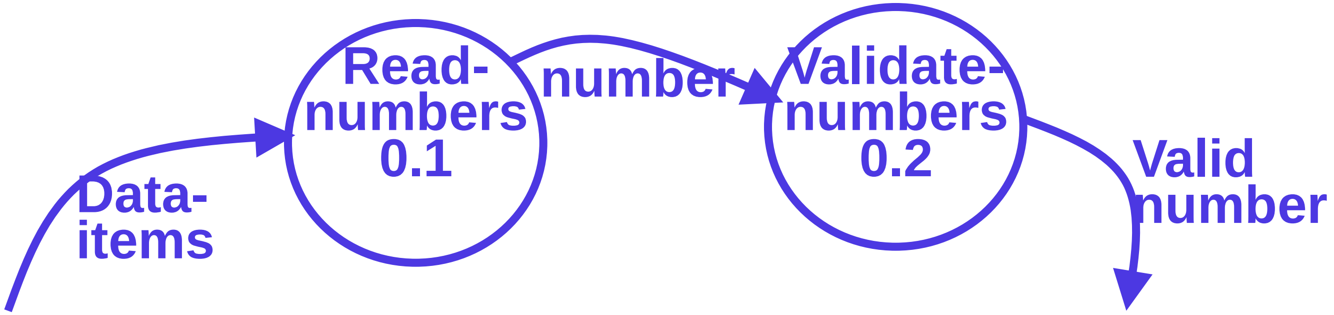
Output Symbol

- **Output produced by the system**
- **Eg: to generate any hard copy.**



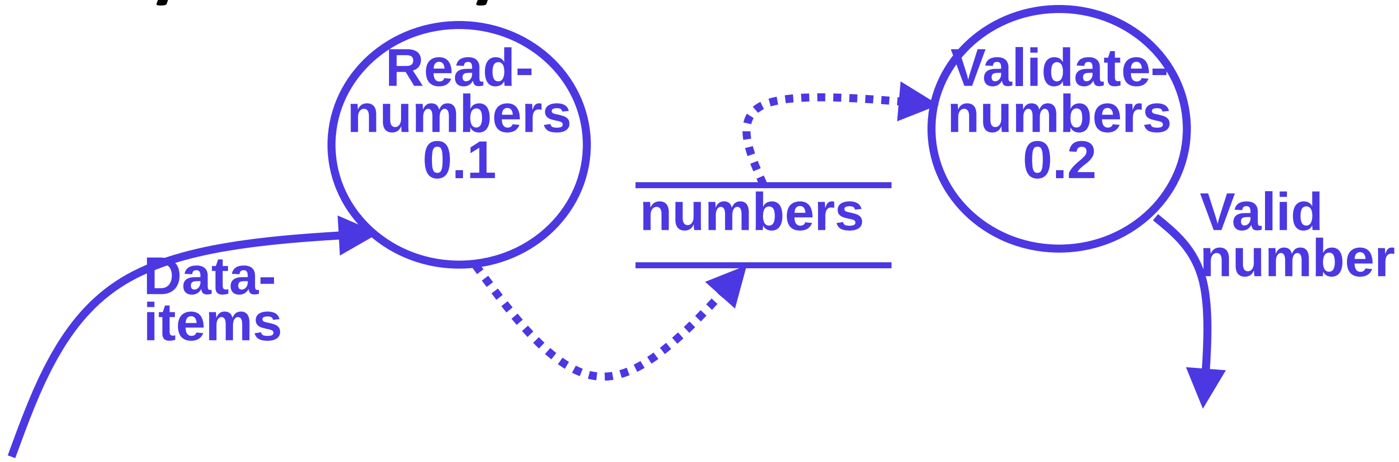
1. Synchronous operation

- **If two bubbles are directly connected by a data flow arrow:**
  - **they are synchronous**



2. Asynchronous operation

- **If two bubbles are connected via a data store:**
  - **they are not synchronous.**



Data dictionary

Every DFD model accompanied by Data Dictionary(for each DFD).

- It enlist the purpose of all data items and the definition of all composite data items in terms of their component data items appear in DFD’s.

Eg: total-pay = basic pay+ overtime pay

basic pay= integer

overtime pay= integer

List of operators

1. **+** : represents data a and b. eg: a+b
2. **[,,]**: anyone of the data item listed inside can occur. Eg: [a,b]
3. **()**: content inside bracket may or may not appear eg: a+(b)
4. **{}**:represents iterative data items. eg: {name}*
5. **=** :represents equivalence, eg: a= b+c
6. **/*   */ :**anything inside considered as comment.

Data Dict…

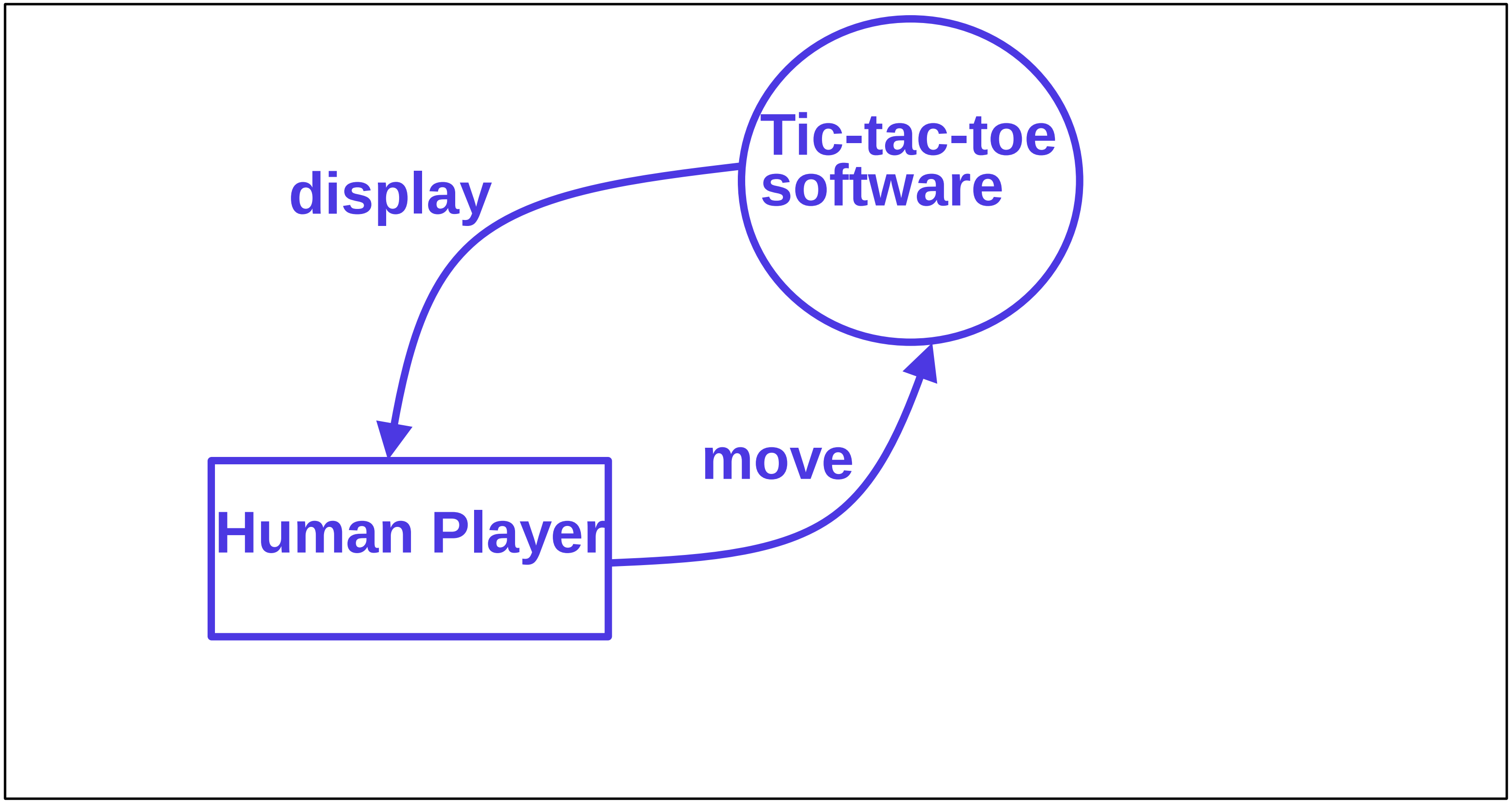
- **all data names along with the purpose of data items.**
- **For large systems,**
  - **the data dictionary grows rapidly in size and complexity.**
  - **Typical projects can have thousands of data dictionary entries.**
  - **It is extremely difficult to maintain such a dictionary manually.**

- **CASE tools** capture the data items appearing in a DFD automatically to generate the data dictionary.
- It Provides consistent vocabulary for data items
- Data-flow diagrams are a means of documenting end-to-end data flow. Structure charts represent the dynamic hierarchy of function calls

How is Structured Analysis Performed?

- Initially represent the software at the most abstract level:
  - called the context diagram.
  - the entire system is represented as a single bubble,
  - this bubble is labelled according to the main function of the system.

Tic-tac-toe: Context Diagram



Context Diagram

- **A context diagram shows:**
  - **data input to the system,**
  - **output data generated by the system,**
  - **external entities.**
  - **The context diagram is also called as the level 0 DFD.**

Level 1 DFD

- **Examine the SRS document:**
  - **Represent each high-level function as a bubble.**
  - **Represent data input to every high-level function.**
  - **Represent data output from every high-level function.**

Higher level DFDs

- **Each high-level function is separately decomposed into subfunctions:**
  - **identify the subfunctions of the function**
  - **identify the data input to each subfunction**
  - **identify the data output from each subfunction**
- **These are represented as DFDs.**

Decomposition

- **Decomposition of a bubble:**
  - **also called factoring or exploding.**
- **Each bubble is decomposed to**
  - **between 3 to 7 bubbles.**

Decompose how long?

- **Decomposition of a bubble should be carried on until:**
  - **a level at which the function of the bubble can be described using a simple algorithm.**

Numbering of bubbles:

- ➢ assign number to each bubble.
- ➢ helps to uniquely identify ancestors and successors.

If parent bubble is x then its children bubbles will be x.1,x.2,x.3,….etc.

Eg: for 0 level DFD, the number is 0.

for 1level DFD, the number is 0.1,0.2,0.3 etc.

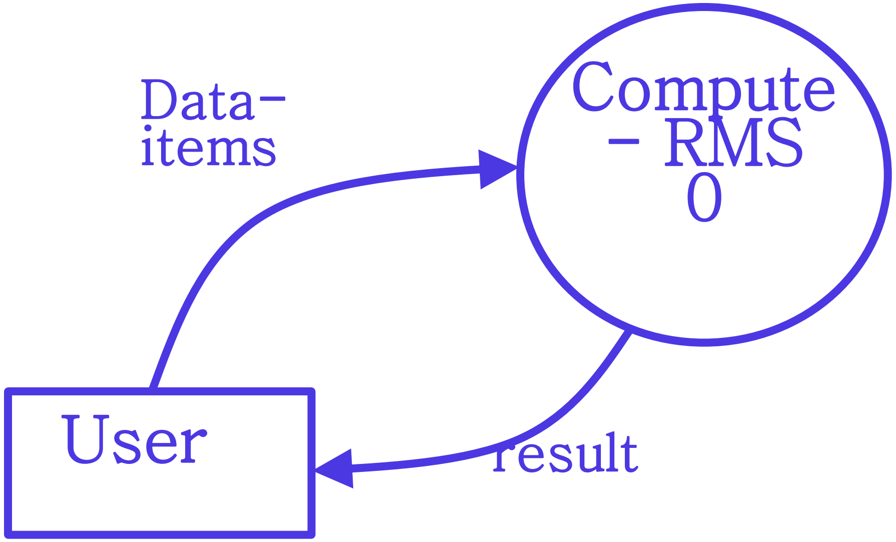
Example 1: RMS Calculating Software

- Consider a software called RMS calculating software:
  - reads three integral numbers in the range of -1000 and +1000
  - finds out the root mean square (rms) of the three input numbers
  - displays the result.

Example 1: RMS Calculating Software

- **The context diagram is simple to develop:**
  - **The system accepts 3 integers from the user**
  - **returns the result to him.**

Example 1: RMS Calculating Software



Context Diagram

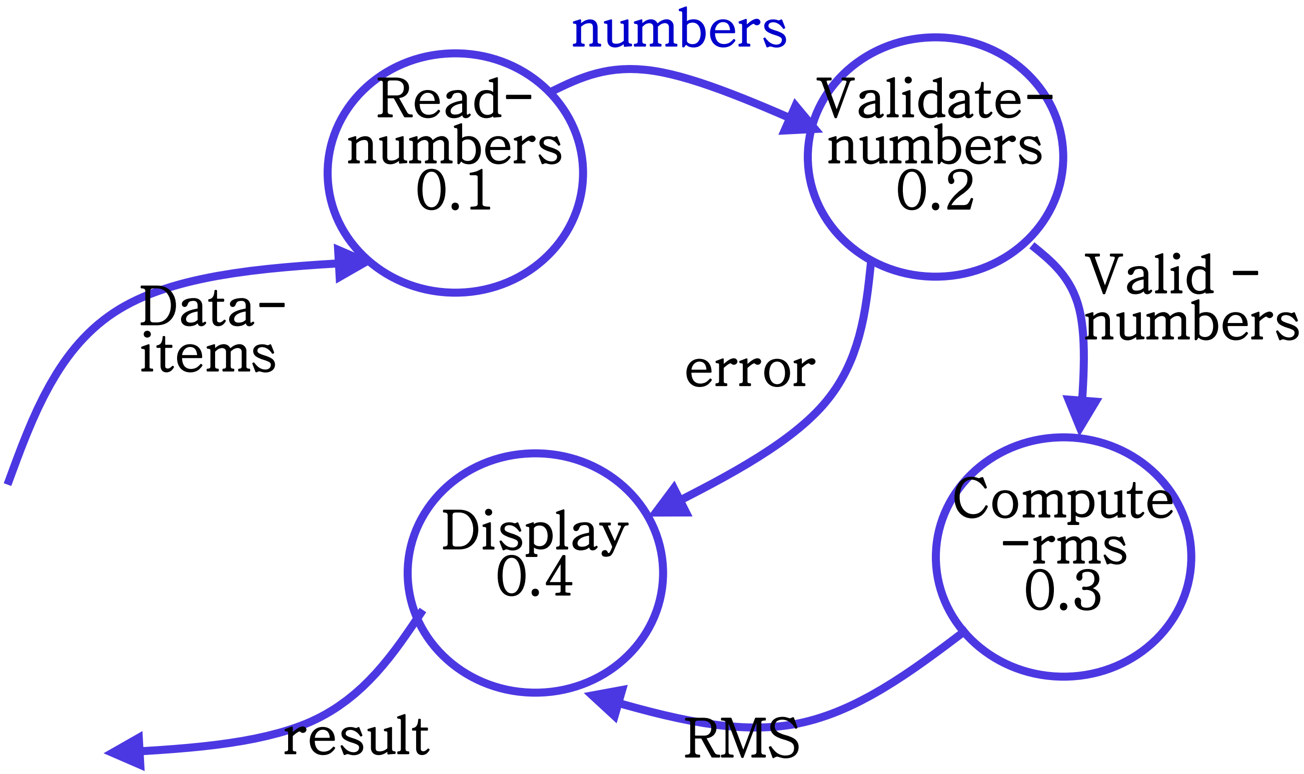
Example 1: RMS Calculating Software

- **From a cursory analysis of the problem description:**
  - **we can see that the system needs to perform several things.**

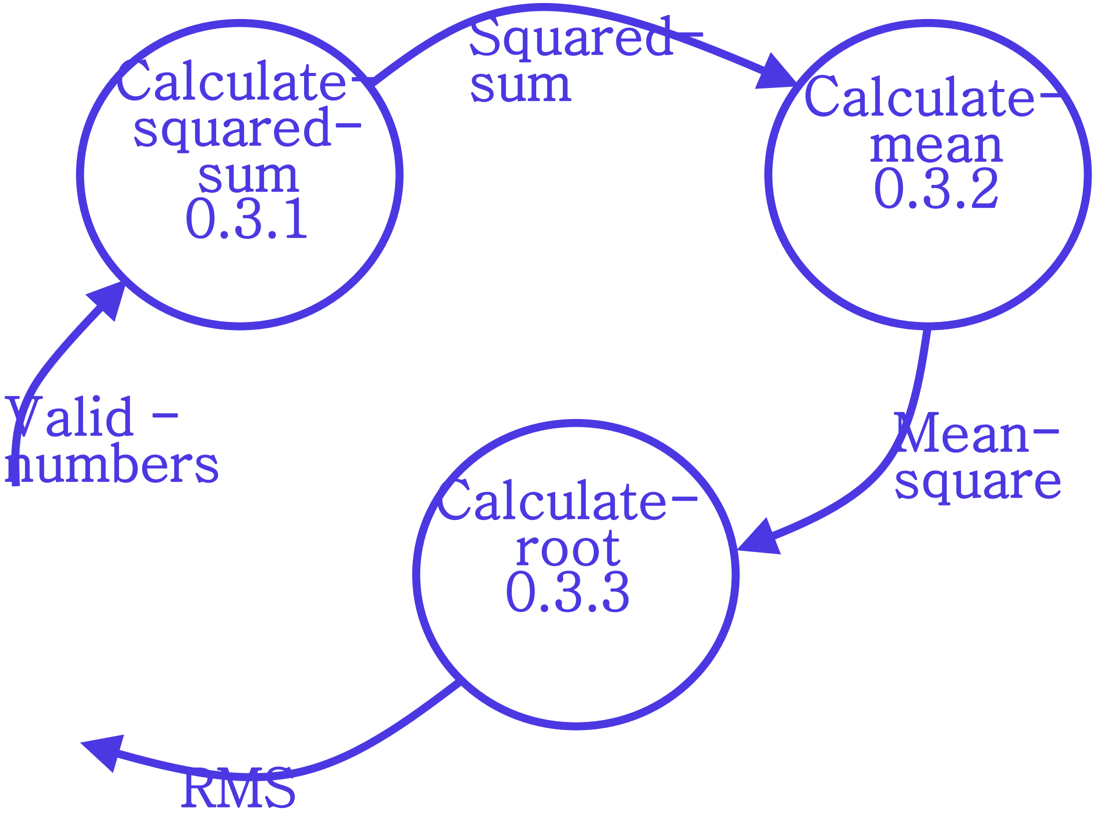
Example 1: RMS Calculating Software

- **Accept input numbers from the user:**
  - **validate the numbers,**
  - **calculate the root mean square of the input numbers**
  - **display the result.**

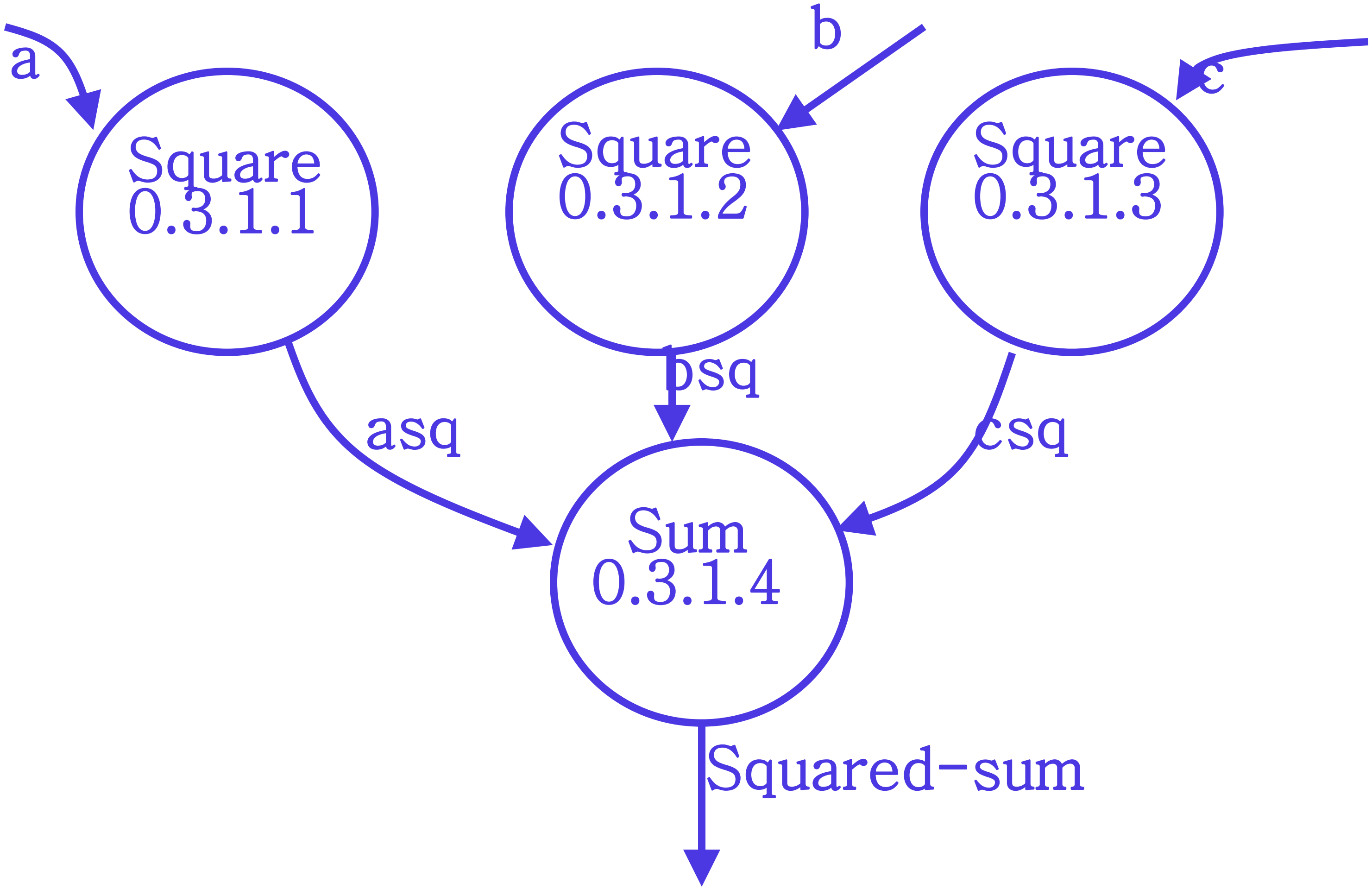
Example 1: RMS Calculating Software



Example 1: RMS Calculating Software



Example: RMS Calculating Software



Example: RMS Calculating Software

- Decomposition is never carried on up to basic instruction level:
  - a bubble is not decomposed any further:
    - if it can be represented by a simple set of instructions.

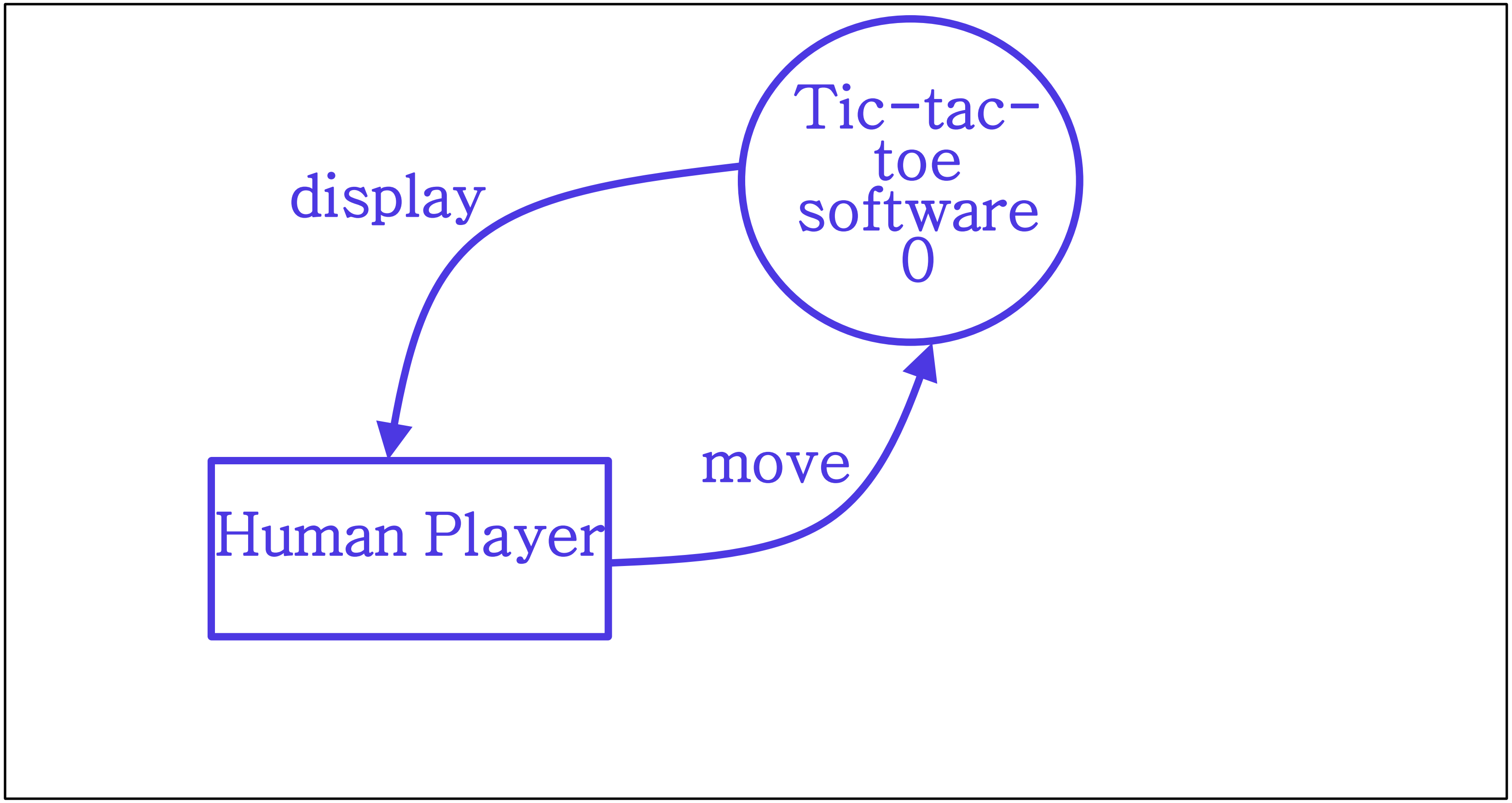
Data dictionary for RMS Software

- **Data items: {integer}3**
- **Result=[RMS,error]**
- **numbers=valid-numbers= a+b+c**
- **a:integer** * input number *
- **b:integer** * input number *
- **c:integer** * input number *
- **asq:integer bsq:integer csq:integer squared-sum: integer**
- **RMS: integer** * root mean square value*
- **error:string** * error message*

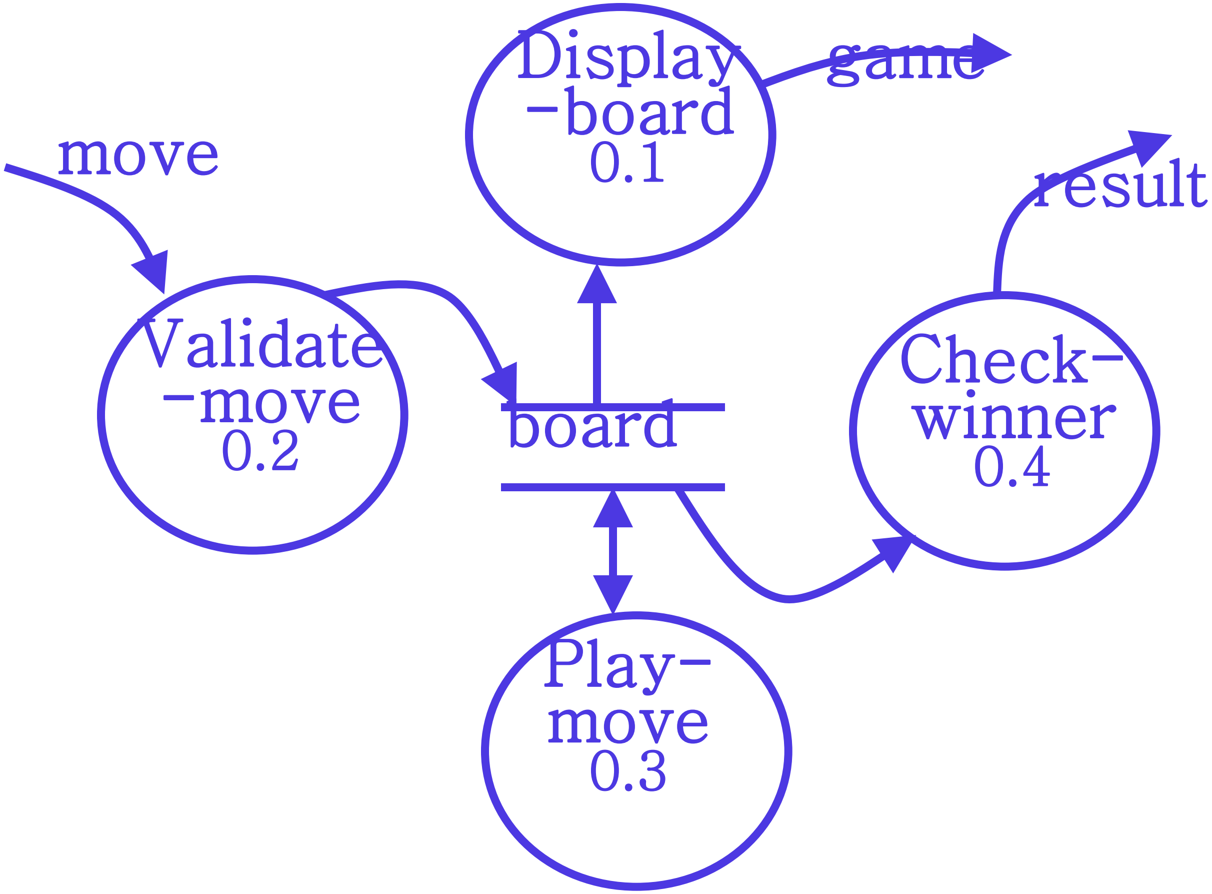
Example 2: tic –tac-toe Software

- Human player and computer **make alternate moves** on a 3*3 square.
- **Mark** the previously unmarked square.
- Place consecutive marks along a straight line will decide the winner.
- **Display** the winner.
- The **game is drawn,** if neither human player nor computer make consecutive marks.

Context Diagram for Example



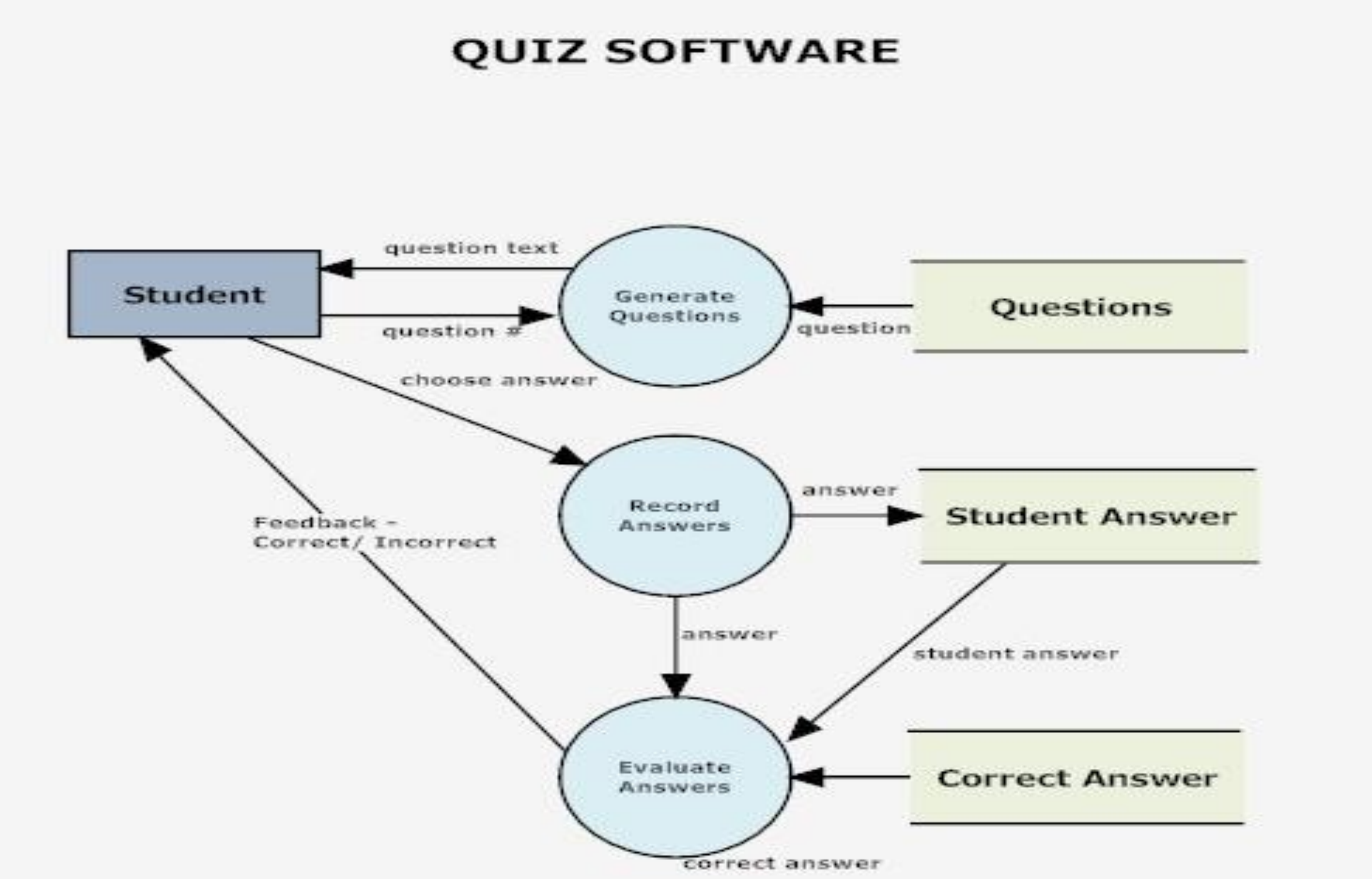
Level 1 DFD


Display
-board
0.1
game
move
result
Validate
-move
0.2
board
Check-
winner
0.4
Play-
move
0.3

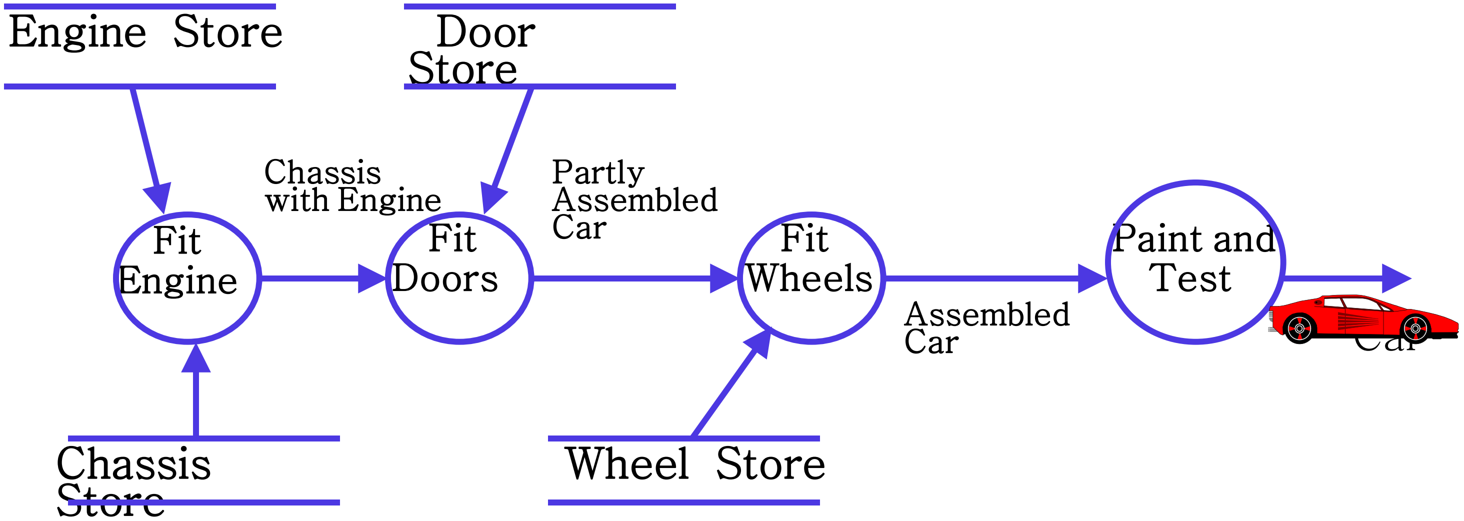
Data dictionary

- Display=game + result
- move = integer
- board = {integer}9
- game = board= {integer}9
- result=string [“computer won”, “Human won”, “drawn”]

DFD


QUIZ SOFTWARE
Student
question text
question #
Generate Questions
question
Questions
choose answer
Record Answers
answer
Student Answer
Feedback - Correct/ Incorrect
answer
student answer
Evaluate Answers
Correct Answer
correct answer

DFD of a Car Assembly Unit



Engine Store

Door Store

Chassis with Engine

Partly Assembled Car

Fit Engine

Fit Doors

Fit Wheels

Paint and Test

Assembled Car

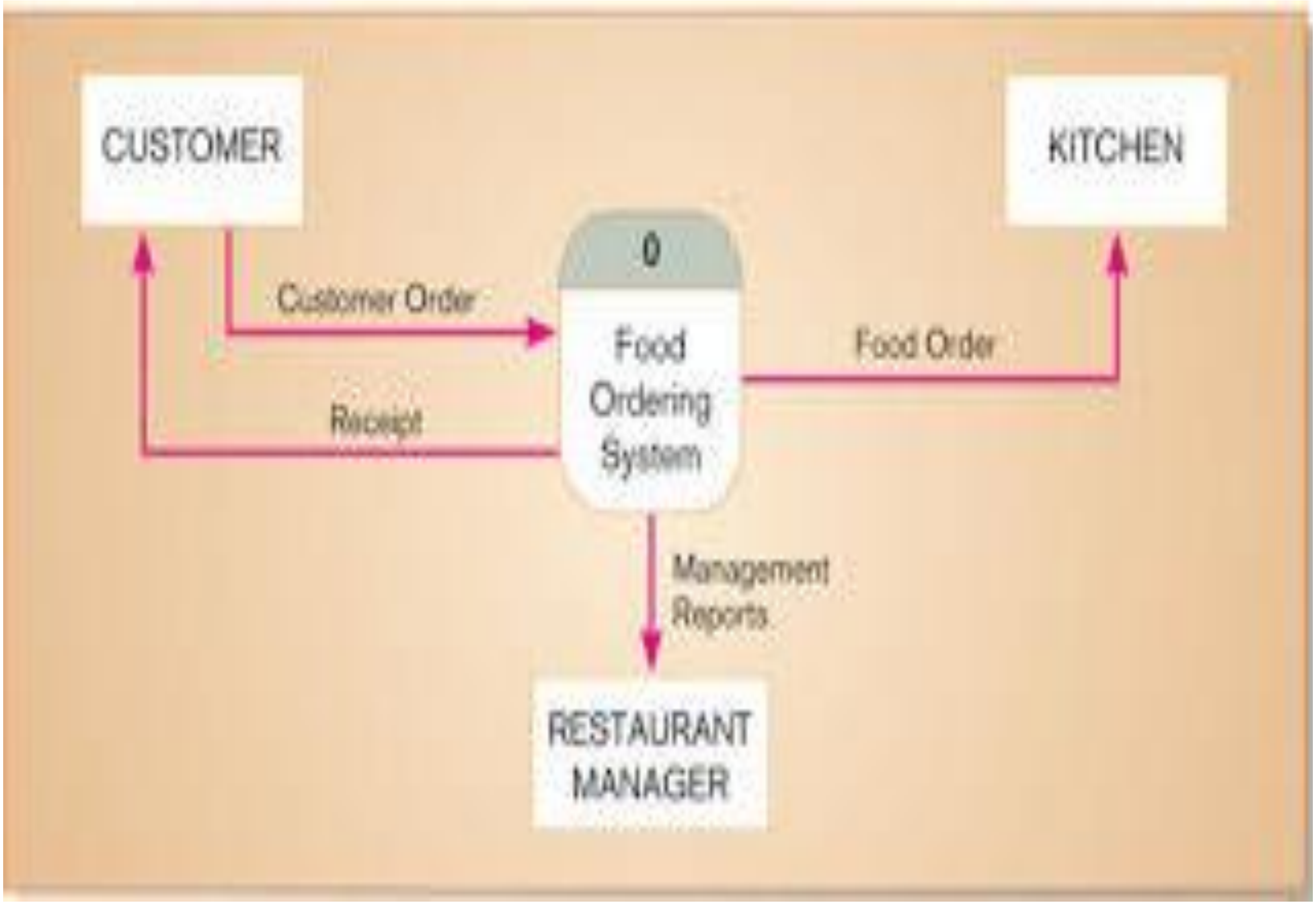
Car

Chassis Store

Wheel Store

DFD for Supermarket Prize Scheme & TAS

- TAS: trading house automation system


CUSTOMER
KITCHEN
0
Food Ordering System
Customer Order
Receipt
Food Order
Management Reports
RESTAURANT MANAGER

Structure Chart

- It represents the no. of modules, module dependency, parameters passed among modules.
- Easy to implement using programming language.

Building blocks of structured design are:

1. Rectangular boxes:
2. Module invocation arrows: can’t judge **no. and the order** of modules calls.

3. Data flow arrow: appears alongside the invocation arrows and annotated with data name.

4. Library module: designed as rectangle with double edges. Module having **high fan-in** named as **library** module.

- Should be one module at the top called **Root**.
- Should be at least one way control relationship between any two modules in the str. chart.

Two strategies:

a). Transform analysis and

b). Transaction analysis

Note: check DFD level's for applicability of these strategies.

Whether to apply Transform or transaction processing?

- If all the input data incident to a same bubble in the DFD then transform analysis, otherwise transaction.
- Transform is normally for very small problems.
- In Str. Chart, draw a root module and below it identified transaction modules.

Transform Analysis:

- **Divide DFD into 3 parts:** I/p, logical processing and O/p. Afferent and efferent branch, and central transform.

Transaction Analysis:

Is an alternative to transform analysis, used for designing transaction processing programs.

- In transform analysis, entered data item goes through the same processing steps. But in transaction goes through diff. processing steps.

Transform Analysis

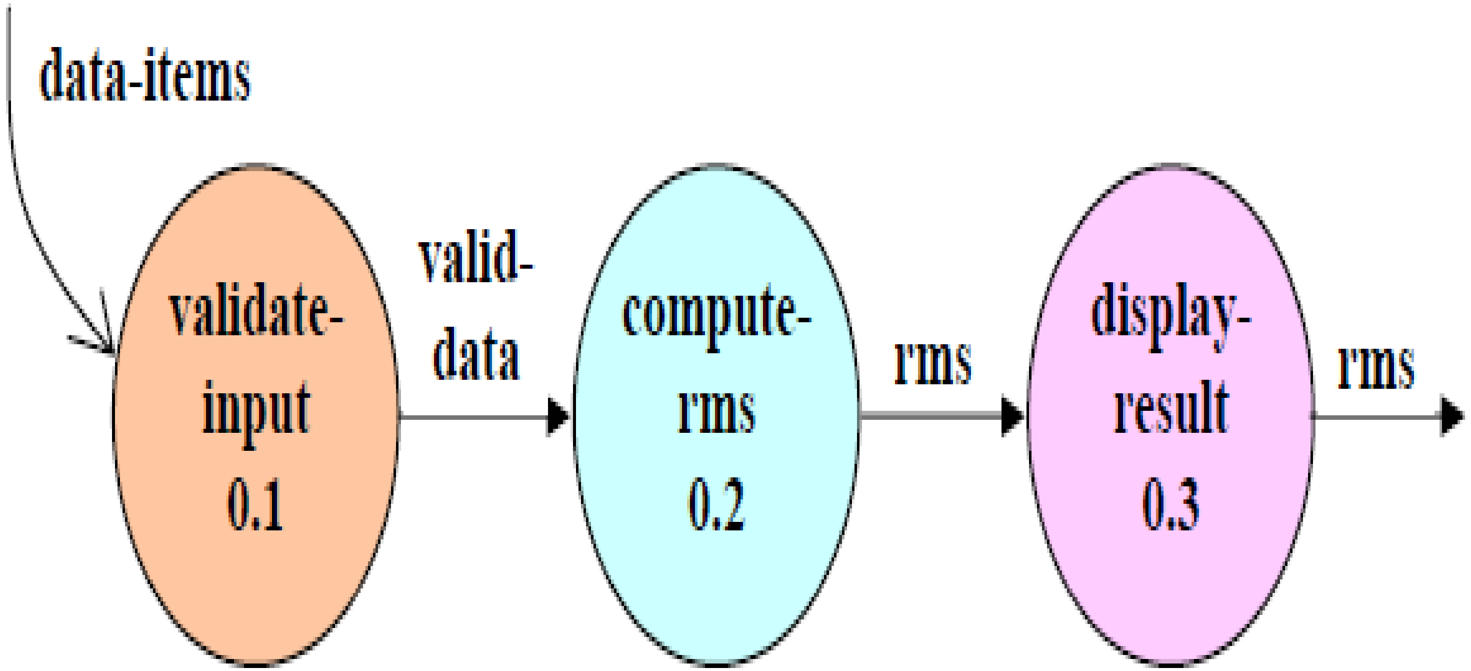


Fig. 5.11: Level 1 DFD

Str. chart

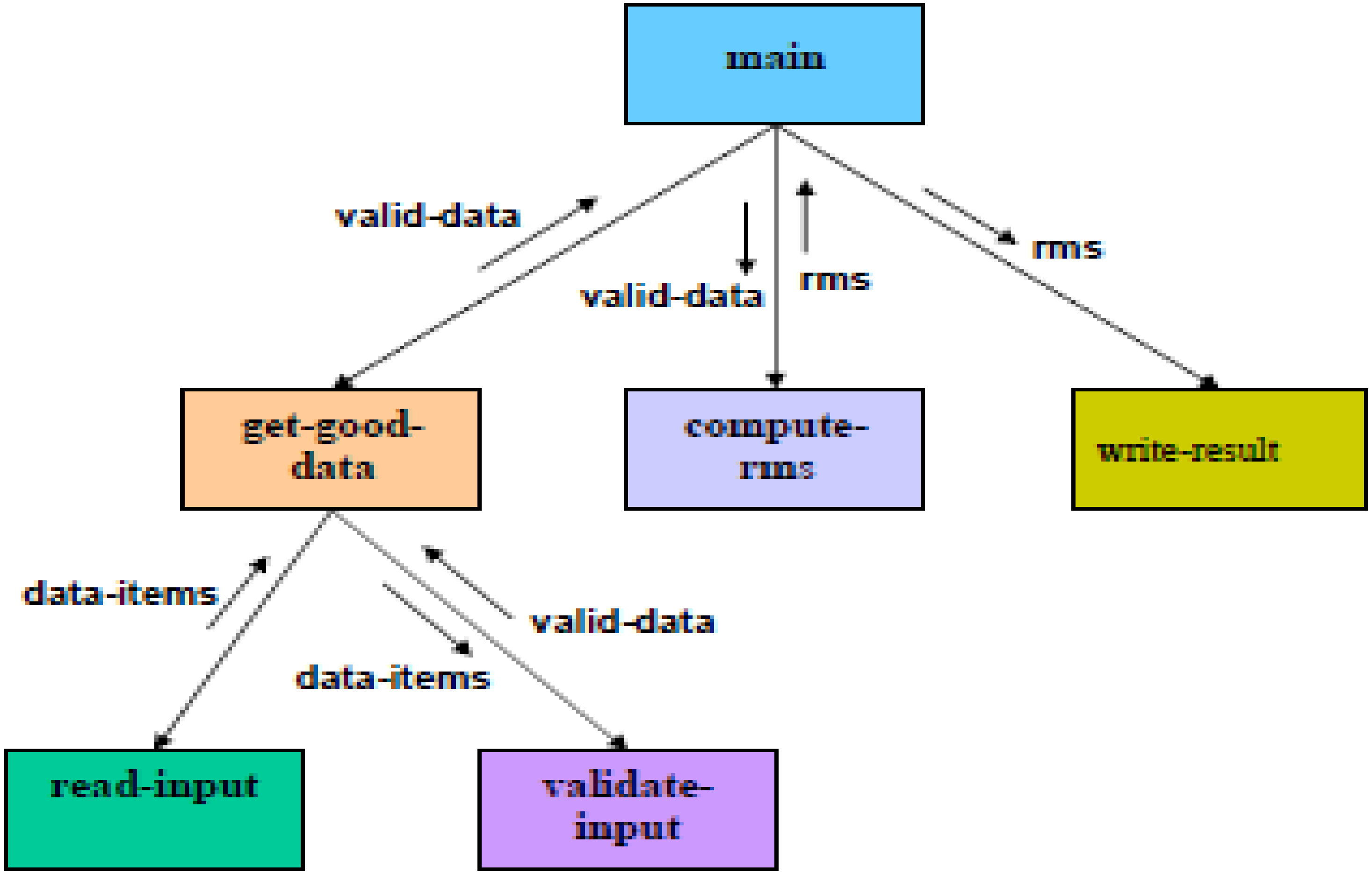


Fig. 5.12: Structure chart

Transaction Analysis

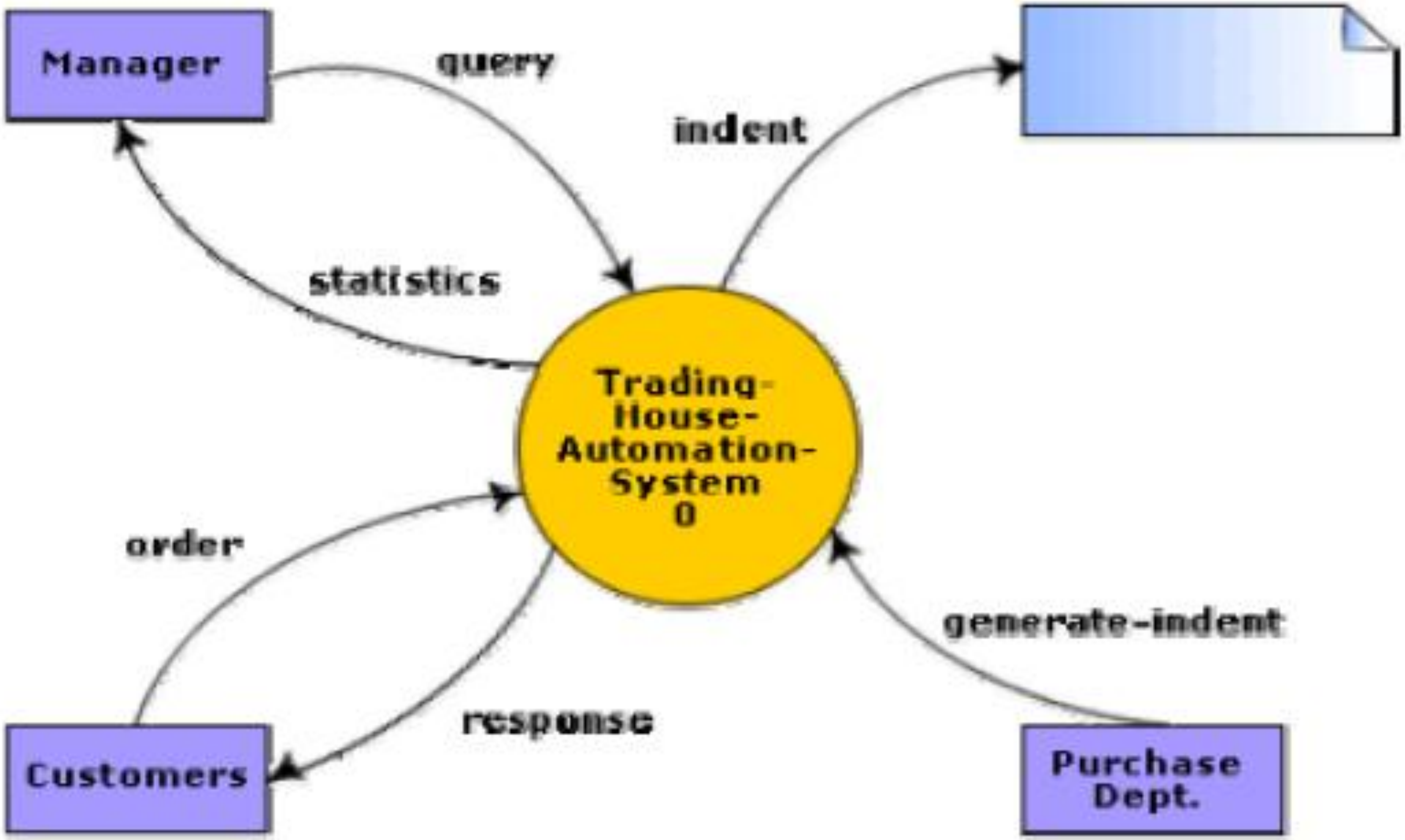
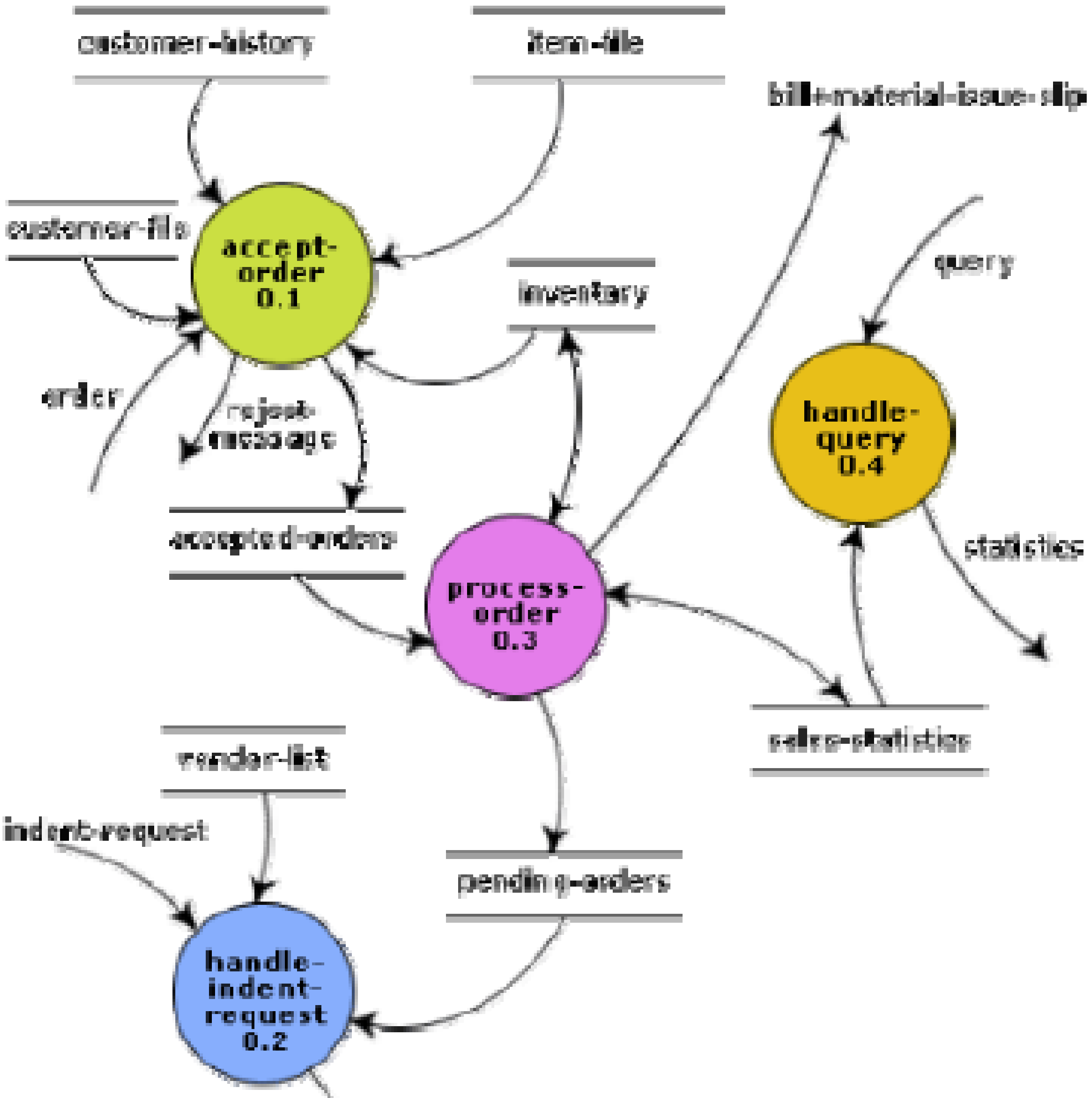


Fig. 5.8: Context diagram for TAS


customer-history
item-file
bill+material-issue-slip
customer-file
accept-order 0.1
inventory
query
order
reject-message
handle-query 0.4
accepted-orders
process-order 0.3
statistics
sales-statistics
vendor-list
indent-request
pending-orders
handle-indent-request 0.2

- Str. Charts are just the guidelines, rather than rules.

Detailed design: design pseudocode and data str. For processing of the modules and the outcome refers to MSPEC.

Design review:

Members: designer, tester, analyst, maintainer etc.

- They review the design doc. For following aspects:
- Traceability,correctness,maintainability,implementation.

Limitation of DFD

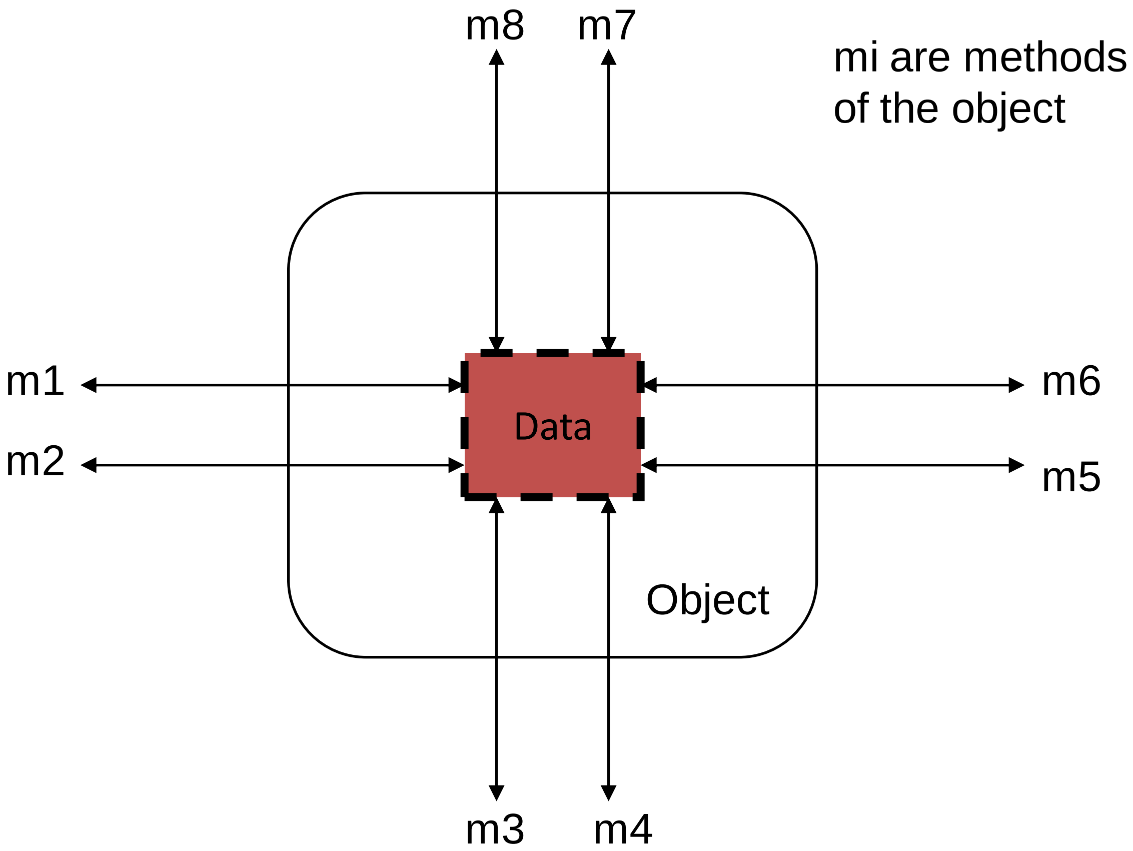
1. **DFDs leave ample scope to be imprecise:**
2. **Control aspects are not defined:**
3. **Data flow diagram:** subjective decomposition
4. **Decomposition:** can design several DFDs for the same problem.

Object modelling using UML

Object-oriented Concepts

- **Basic Mechanisms:**
  - **Objects:**
    - A real-world entity.
    - A system is designed as a set of interacting objects.
    - Consists of data (attributes) and functions (methods) that operate on data
    - Hides organization of internal information (Data abstraction)
    - Examples: an employee, a book etc.

Object-oriented Concepts



mi are methods
of the object

Model of an object

Object-oriented Concepts

- Class:
  - Instances are objects
  - Template for object creation
  - Examples: set of all employees, different types of book

Object-oriented Concepts

- Methods and message:
  - Operations supported by an object
  - Means for manipulating the data of other objects
  - Invoked by sending message
  - Examples: calculate_salary, issue-book, member_details, etc.

Object-oriented Concepts

- Inheritance:
  - Allows to define a new class (derived class) by extending or modifying existing class (base class)
  - Represents Generalization-specialization relationship

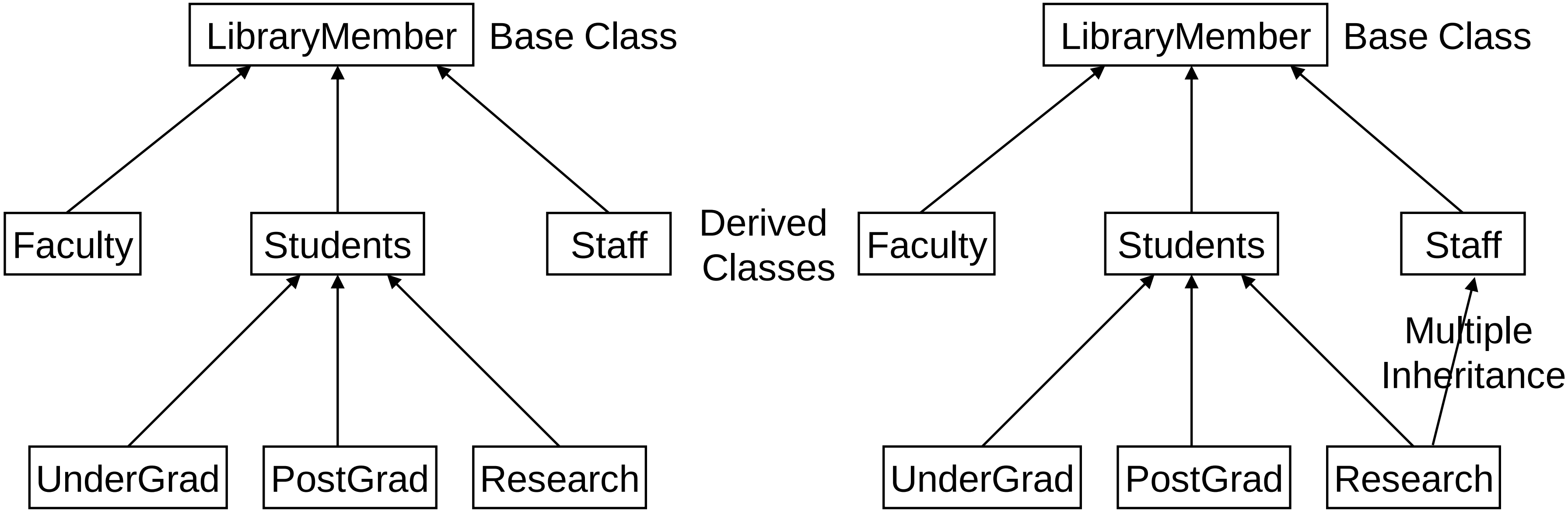
Object-oriented Concepts

– Multiple Inheritance:

- Subclass can inherit attributes and methods from more than one base class

- Multiple inheritance is represented by arrows drawn from the subclass to each of the base classes

Object-oriented Concepts



LibraryMember

Base Class

Faculty

Students

Staff

Derived Classes

UnderGrad

PostGrad

Research

LibraryMember

Base Class

Faculty

Students

Staff

Multiple Inheritance

UnderGrad

PostGrad

Research

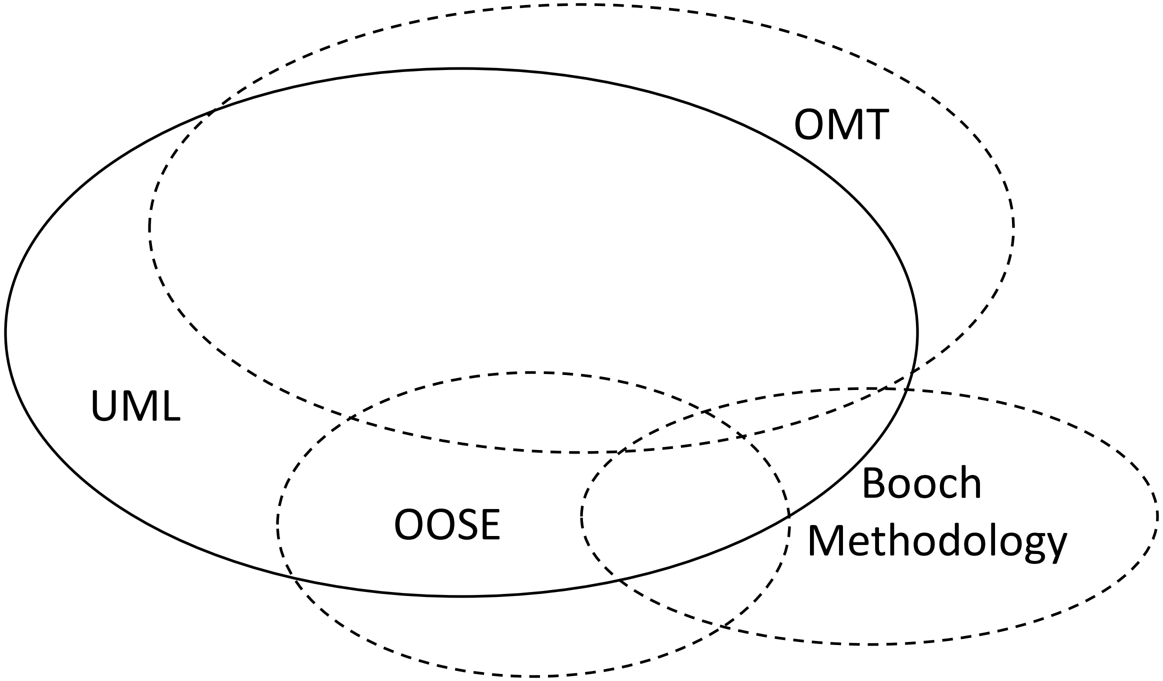
Object modelling using UML

- **UML** is a modelling language
- Not a system design or development methodology
- Used to document object-oriented analysis and design
- Independent of any specific design methodology

UML

- Based Principally on
  - **OMT** [Rumbaugh 1991]
  - **Booch's methodology**[Booch 1991]
  - **OOSE** [Jacobson 1992]
  - **Odell's methodology**[Odell 1992]
  - **Shlaer and Mellor** [Shlaer 1992]

UML



Different object modelling techniques in UML

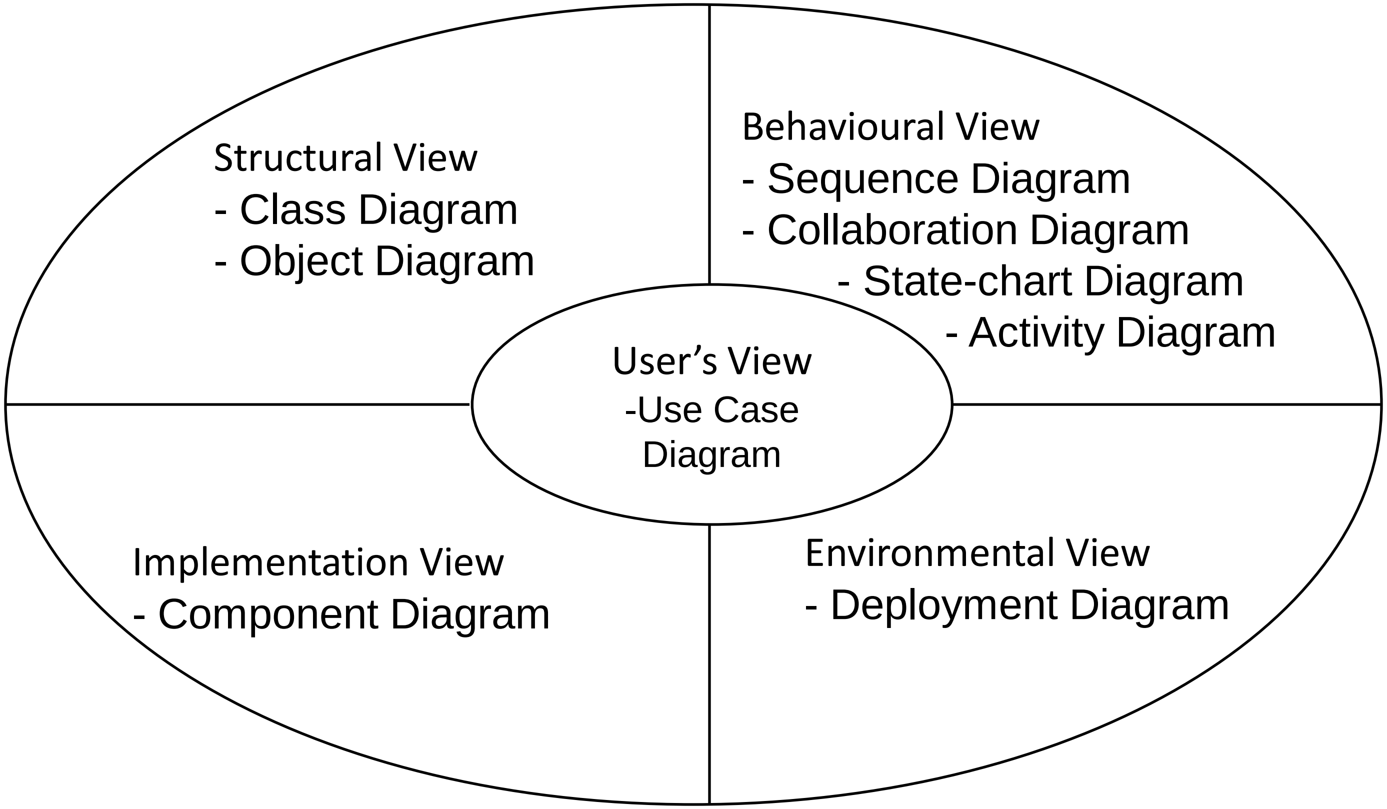
UML diagrams

- Nine diagrams to capture different views of a system
- Provide different perspectives of the software system
- Diagrams can be refined to get the actual implementation of the system

UML diagrams

- Views of a system
  - **User's** view
  - **Structural** view(static model)
  - **Behavioral** view(dynamic)
  - **Implementation** view
  - **Environmental** view

UML diagrams



Diagrams and views in UML

Use Case MODEL

- **Consists of set of "use cases"**

- **Different ways in which system can be used by the users**

- **Corresponds to the high-level requirements**

Use Cases

- Normally, use cases are independent of each other
- Implicit dependencies may exist
- **Example**: In Library Automation System, issue-book & reserve-book are independent use cases. But in actual implementation of issue-book, a check is made to see if any book has been reserved using reserve-book

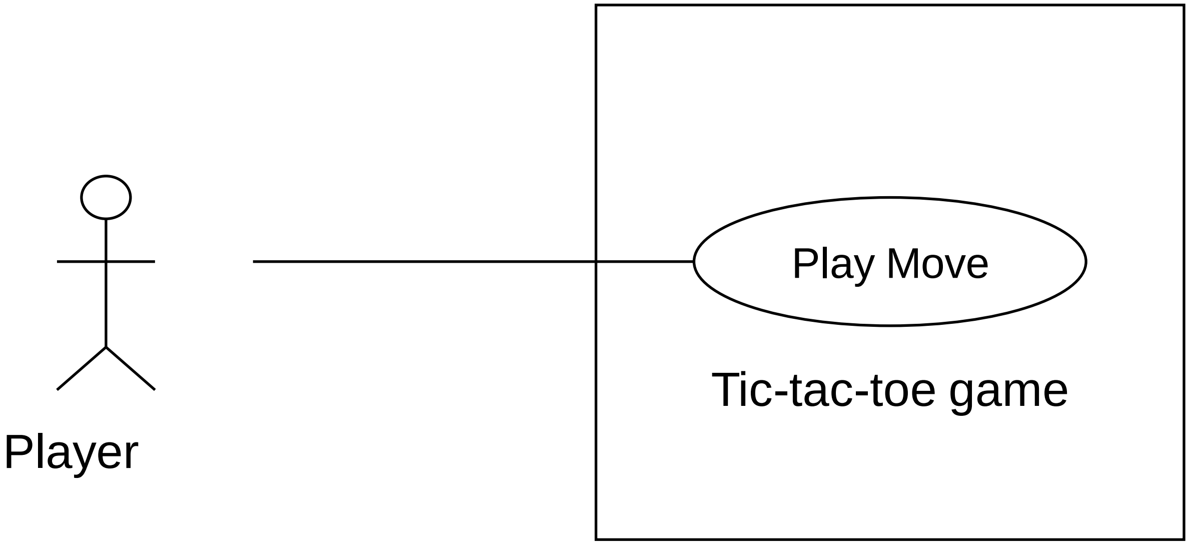
Example of Use Cases

- For library information system
  - issue-book
  - Query-book
  - Return-book
  - Create-member
  - Add-book, etc.

Representation of Use Cases

- Represented by use case diagram
- **Use case** is represented by **ellipse**
- **System boundary** is represented by **rectangle**
- **Users** are represented by **stick person** icon (**actor**)
- **Communication relationship** between actor and use case by **line**

Example of
Use Cases

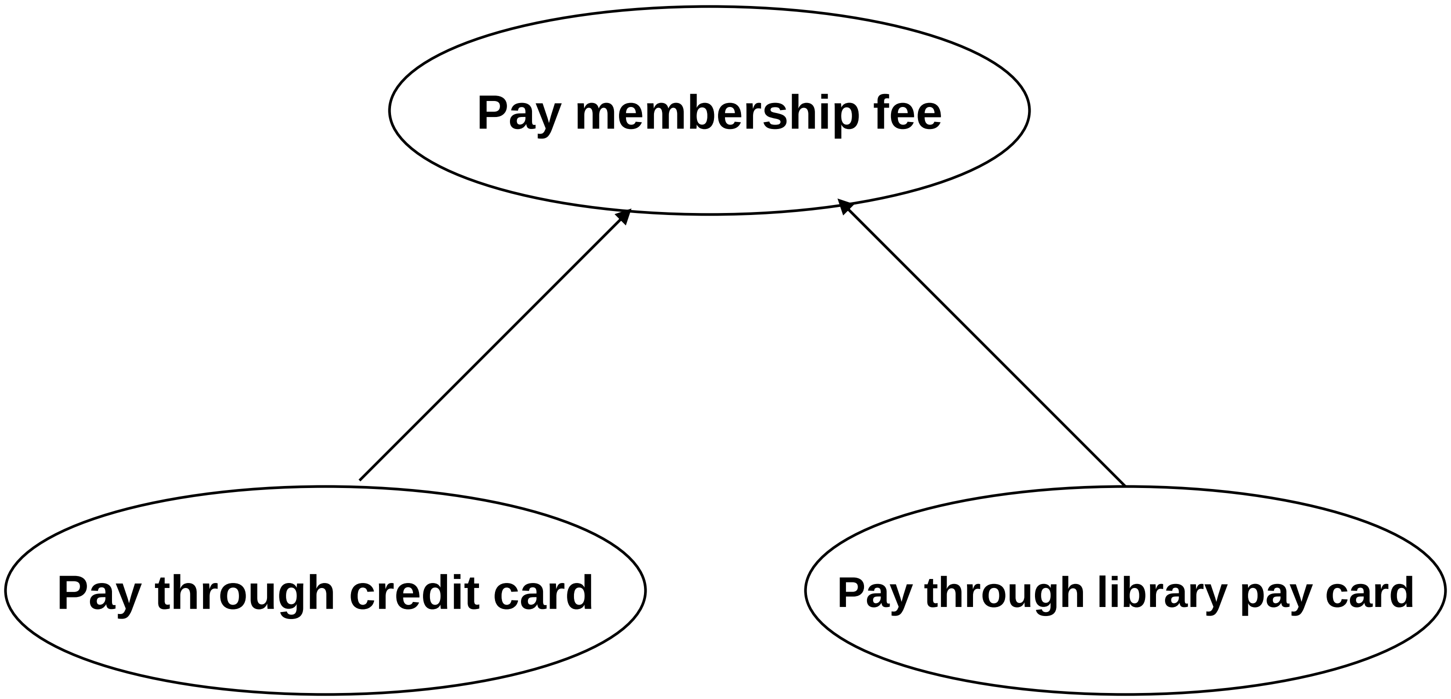


Use case model

Factoring Use Cases

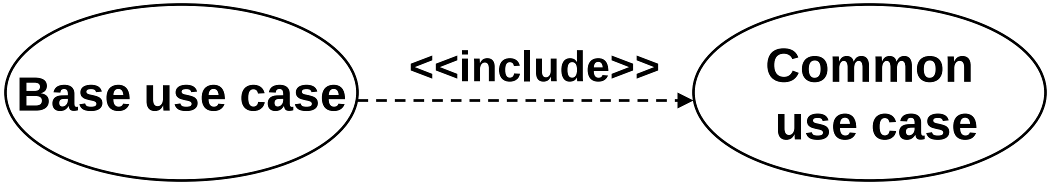
- Complex use cases need to be factored into simpler use cases
- Represent common behavior across different use cases
- Three ways of factoring
  - **Generalization**
  - **Includes**
  - **Extends**

Factoring Using Generalization

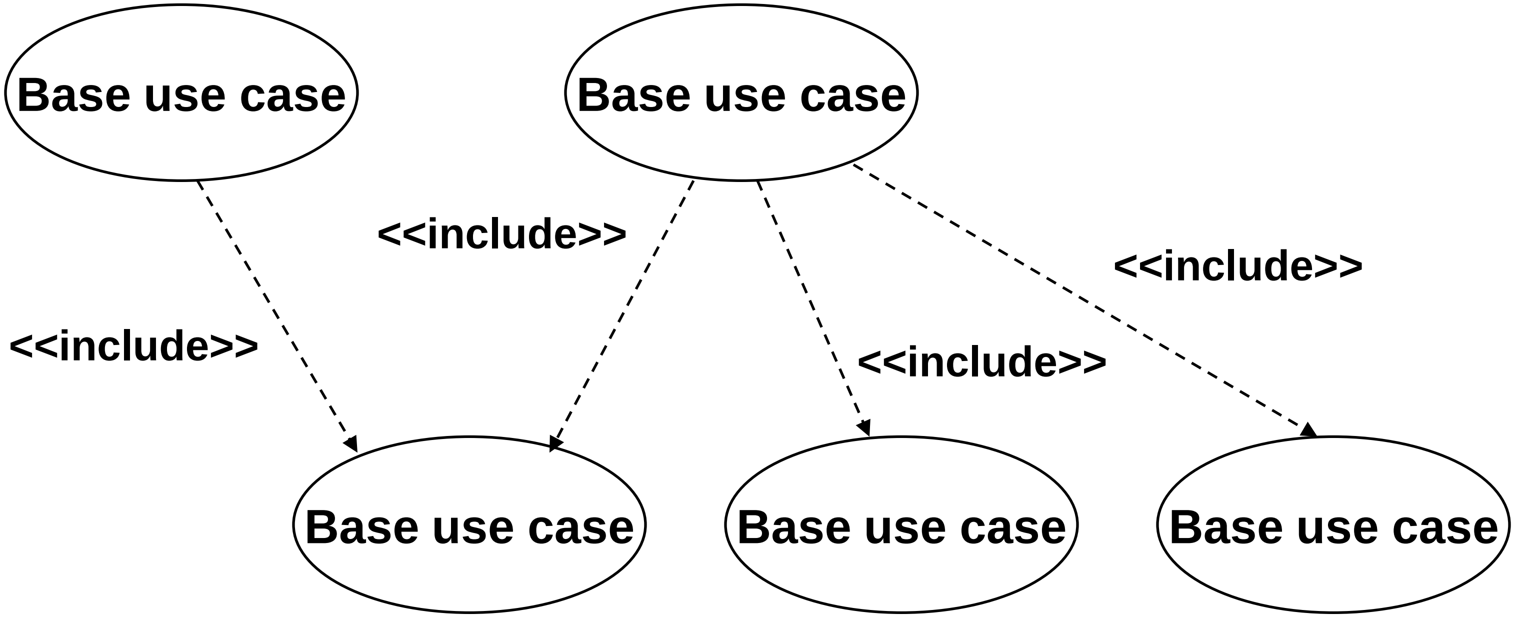


Use case generalization

Factoring Using Includes

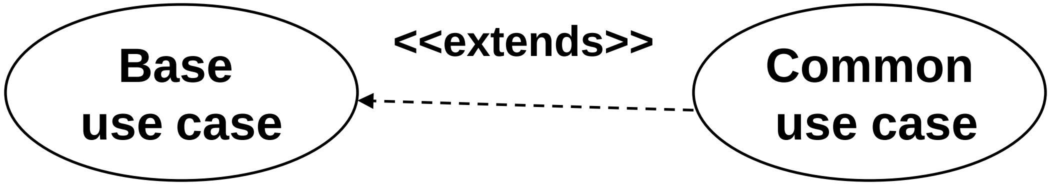


Use case inclusion



Paralleling model

Factoring Using Extends



Use case extension

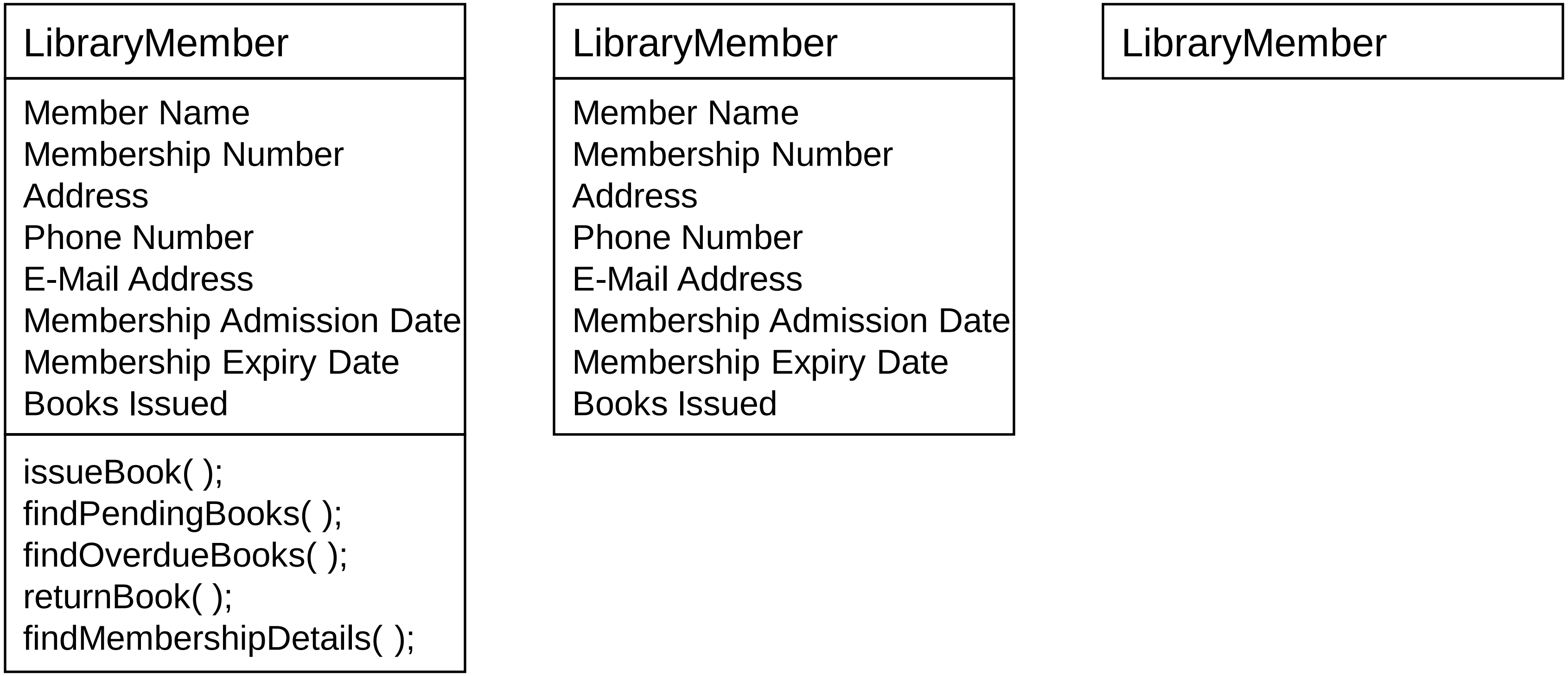
Class diagram

- Describes static structure of a system
- Main constituents are classes and their relationships:
  - **Generalization**
  - **Aggregation**
  - **Association**
  - Various kinds of **dependencies**

Class diagram

- Entities with common features, i.e. attributes and operations
- Classes are represented as solid outline rectangle with compartments
- Compartments for **name**, **attributes** & **operations**
- Attribute and operation compartment are optional for reuse purpose

Example of Class diagram



LibraryMember

Member Name
Membership Number
Address
Phone Number
E-Mail Address
Membership Admission Date
Membership Expiry Date
Books Issued

issueBook( );
findPendingBooks( );
findOverdueBooks( );
returnBook( );
findMembershipDetails( );

LibraryMember

Member Name
Membership Number
Address
Phone Number
E-Mail Address
Membership Admission Date
Membership Expiry Date
Books Issued

LibraryMember

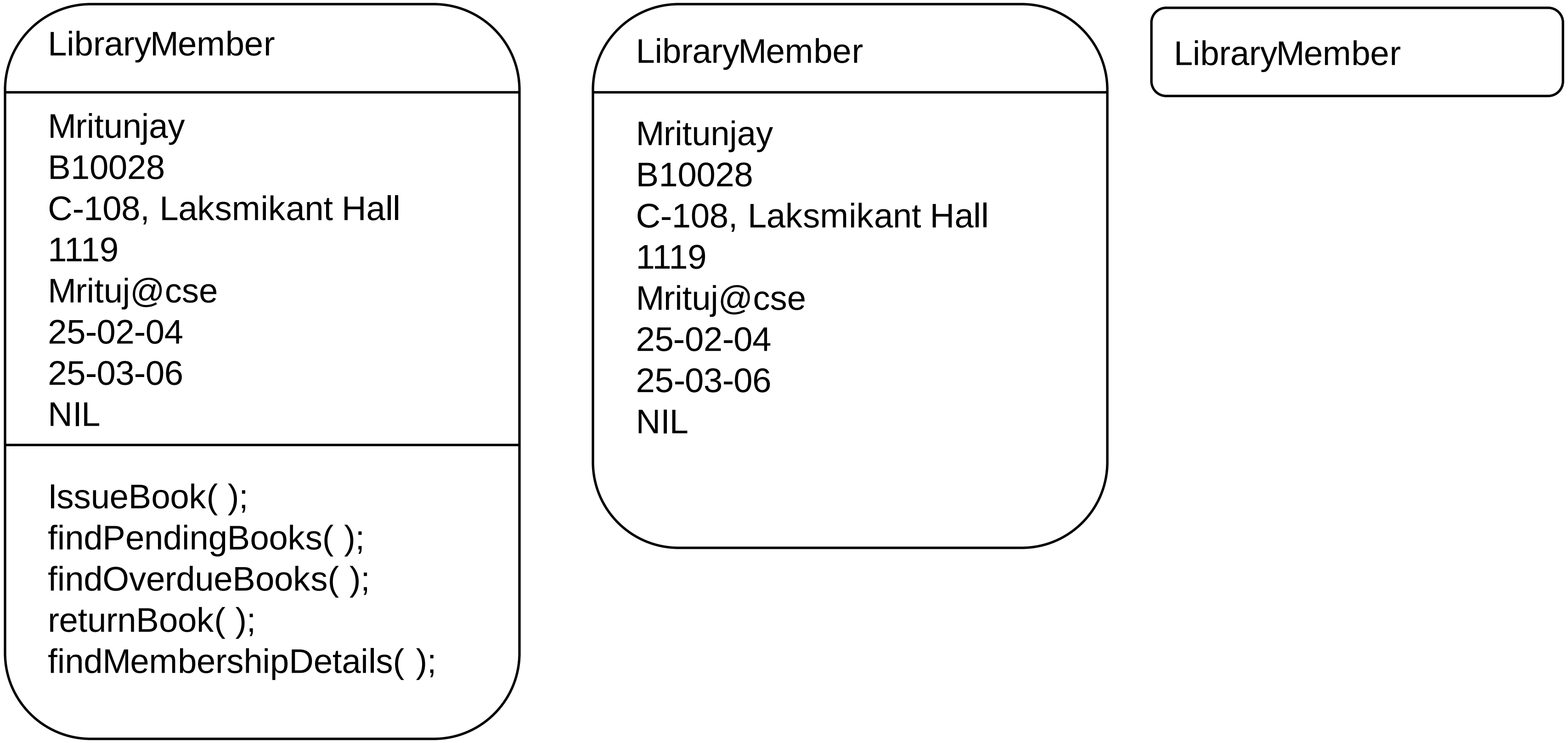
Different representations of the LibraryMember class

- **Class name: begin with Uppercase**

Object name: begin with lower case(student Member)

Attribute: property of a class, and define the data object might contain.

Object diagram



LibraryMember

Mritunjay
B10028
C-108, Laksmikant Hall
1119
Mrituj@cse
25-02-04
25-03-06
NIL

IssueBook( );
findPendingBooks( );
findOverdueBooks( );
returnBook( );
findMembershipDetails( );

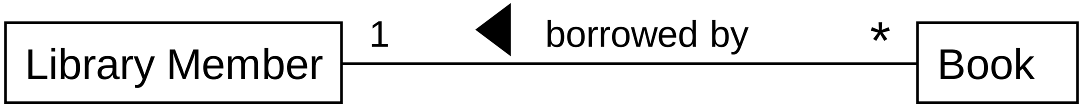
LibraryMember

Mritunjay
B10028
C-108, Laksmikant Hall
1119
Mrituj@cse
25-02-04
25-03-06
NIL

LibraryMember

Different representations of the LibraryMember object

Association Relationship

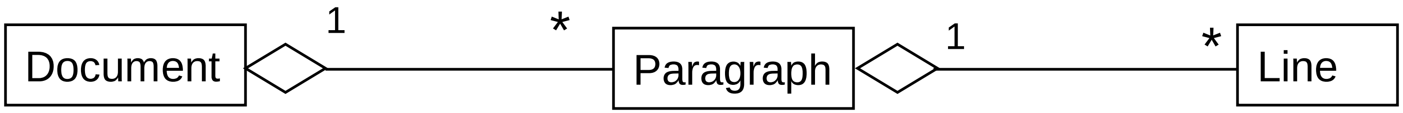


Association between two classes

Aggregation Relationship

- – Represent a whole-part relationship
- – Represented by **diamond** symbol at the composite end
- – It can be transitive

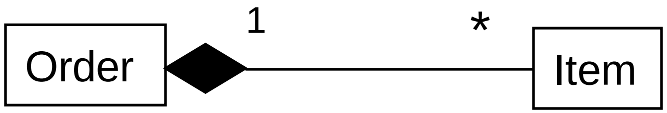
Aggregation Relationship



Representation of aggregation

Composition Relationship

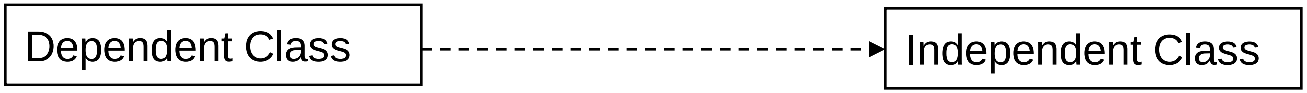
– **Life of item is same as the order**



Representation of composition

Class Dependency

Example: client and server



Representation of dependence between class

Interaction diagram

- Models how groups of objects collaborate to realize some behaviour
- Typically each interaction diagram realizes behaviour of a use case
- Two kinds: **Sequence** & **Collaboration**
- These diagram play a very important role in the design process

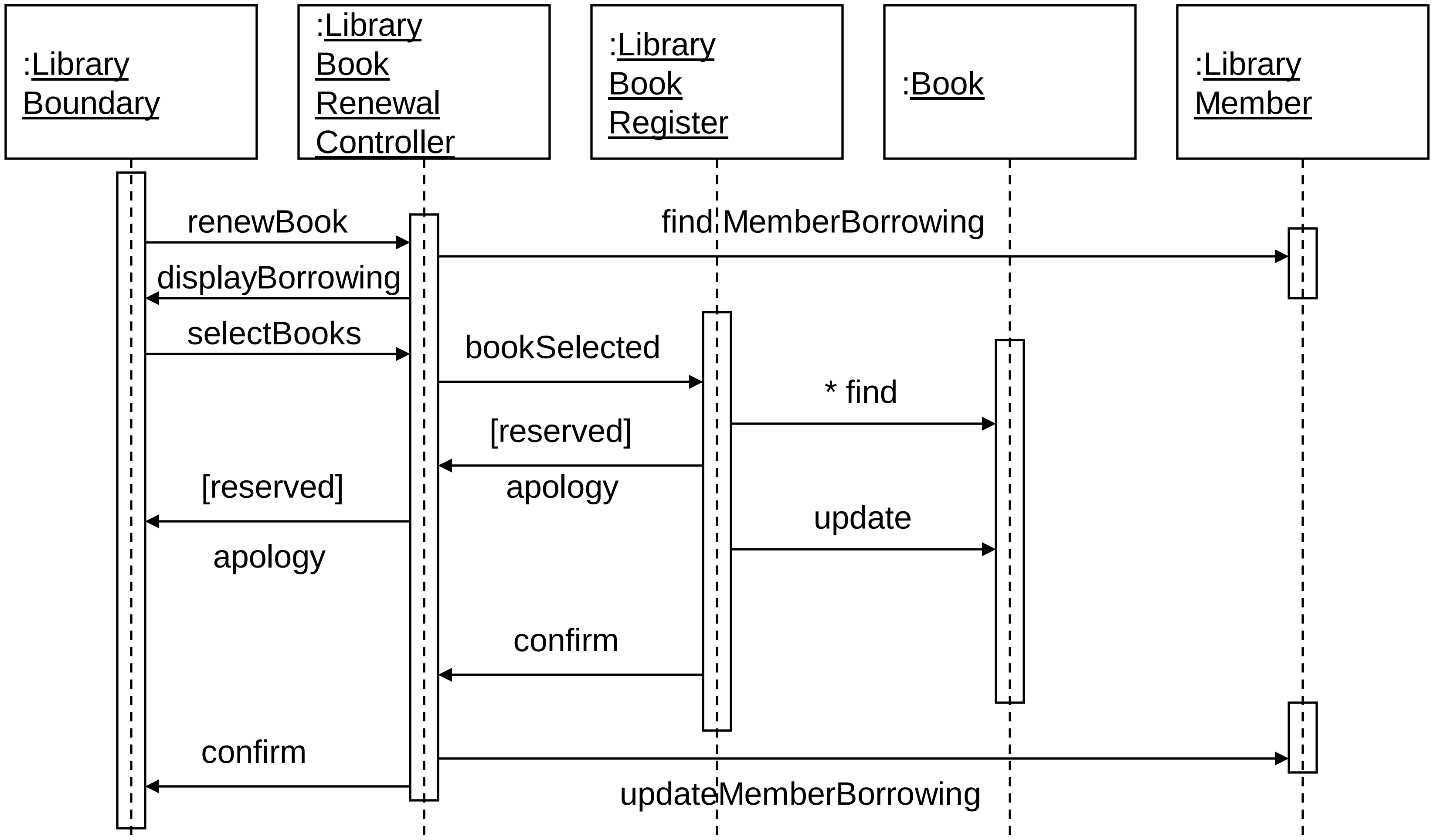
Sequence diagram

- Shows interaction among objects as two-dimensional chart
- **Objects** are shown as **boxes** at top
- If object created during execution then shown at appropriate place
- **Objects existence** are shown as **dashed lines** (lifeline)
- **Objects activeness**, shown as **rectangle** on lifeline

Sequence diagram

- **Messages** are shown as **arrows**
- Message labelled with message name
- Message can be labelled with **control information**
- Two types of control information: **condition** ([]) & an **iteration** (*)

Example of Sequence diagram

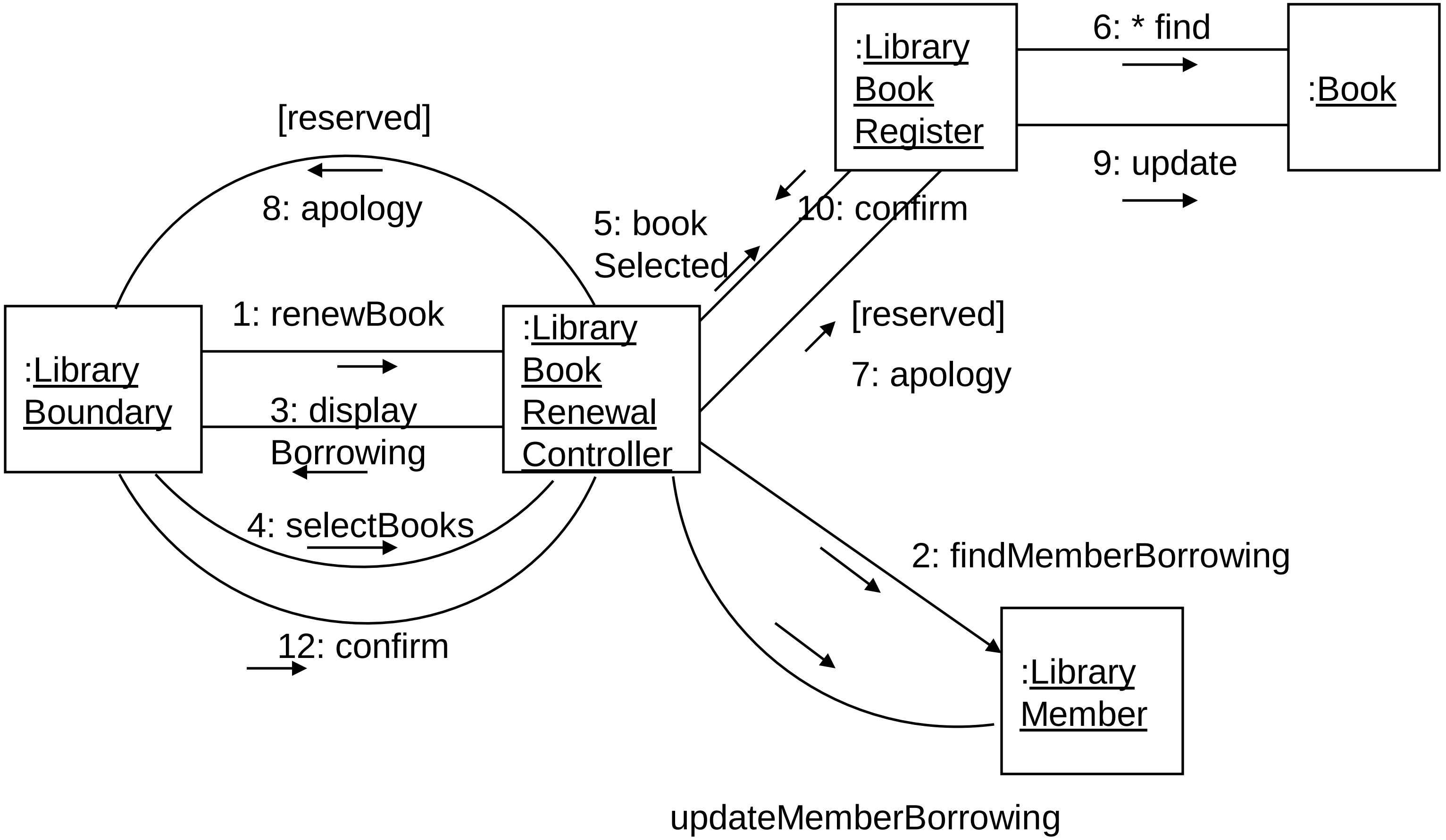


Sequence Diagram for the renew book use case

Collaboration diagram

- Shows both **structural** and **behavioural** aspects
- Objects are **collaborator**, shown as boxes
- Messages between objects shown as a **solid line**
- Message is shown as a **labelled arrow** placed near the link
- Messages are prefixed with **sequence numbers** to show relative sequencing

Example of Collaboration diagram



Collaboration Diagram for the renew book use case

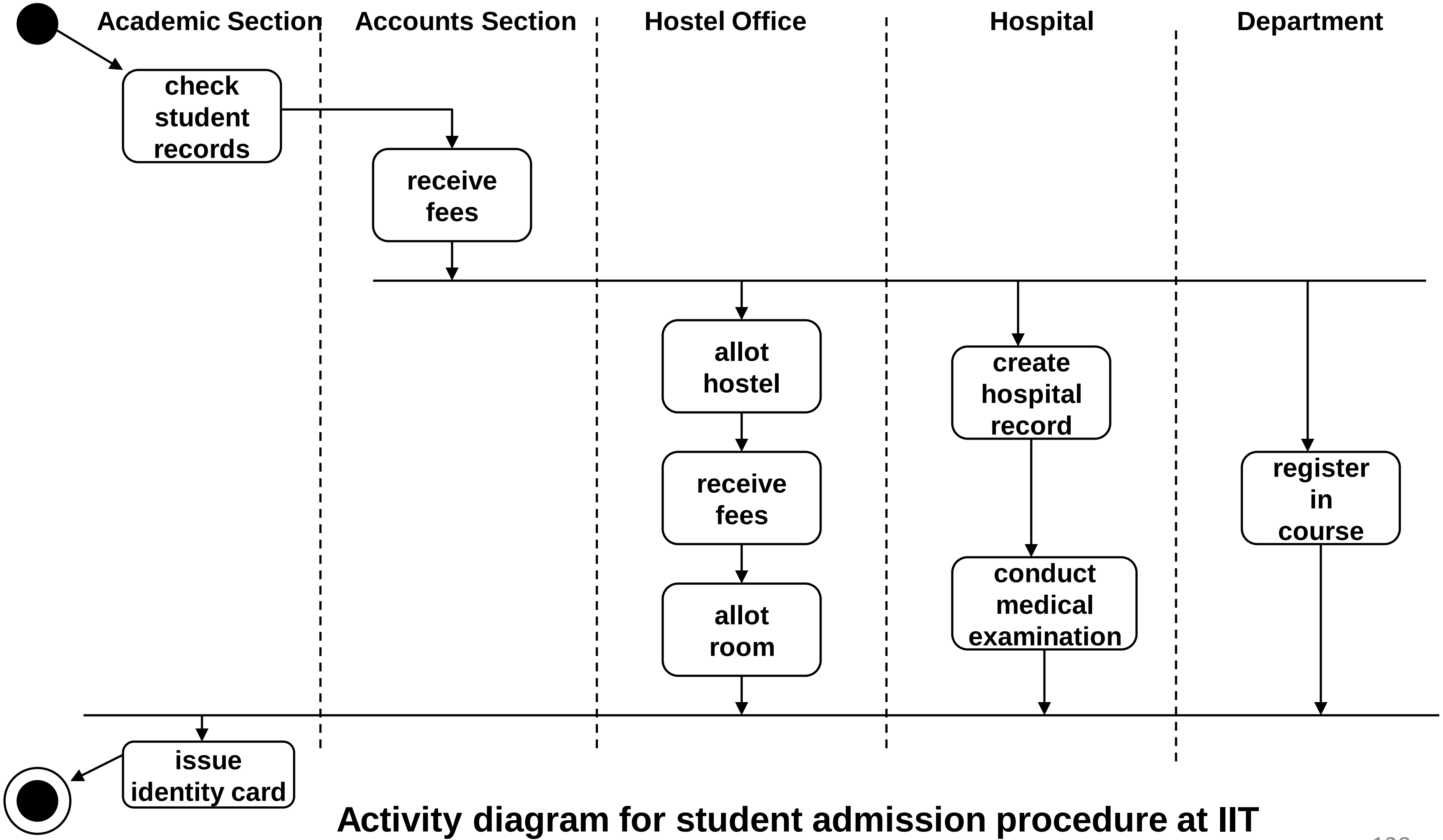
Activity diagram

- Represent processing activity, may not correspond to methods
- Can represent parallel activity and synchronization aspects
- **Swim lanes** enable to group activities based on who is performing them. Example: academic department vs. hostel

Activity diagram

- Normally employed in business process modelling
- Carried out during requirement analysis and specification
- Can be used to develop interaction diagrams

Example of Activity diagram



Activity diagram for student admission procedure at IIT

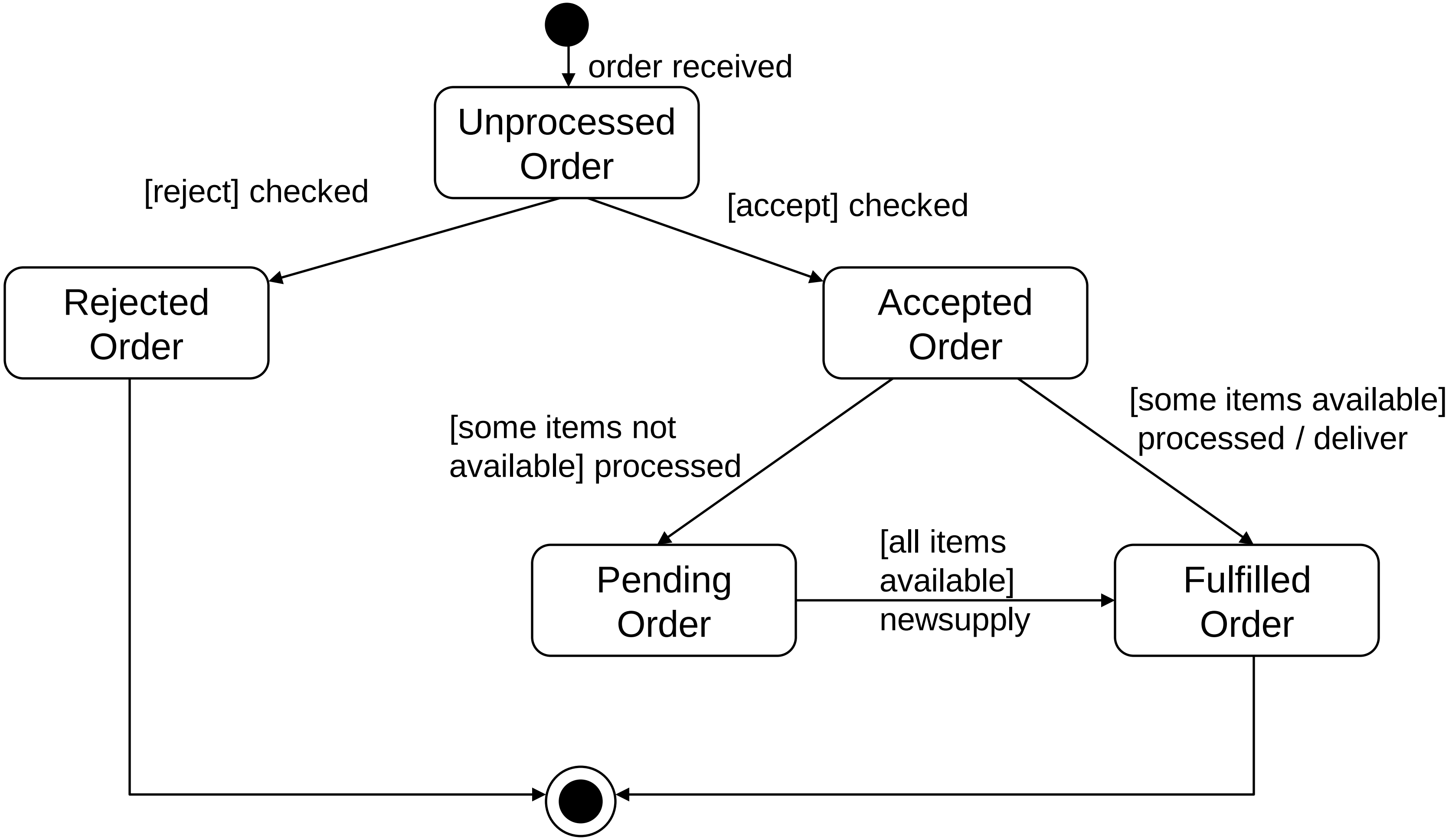
State Chart diagram

– Model how the state of an object changes in its lifetime

Elements of state chart diagram

- **Initial State:** Filled circle
- **Final State:** Filled circle inside larger circle
- **State:** Rectangle with rounded corners
- **Transitions:** Arrow between states, also boolean logic condition (**guard**)

Example of State Chart diagram



Example: State chart diagram for an order object

- **Fundamental diff.** b/w func. Oriented and object- oriented design approach is:

➢ Read the specification of the software you want to bulid. Underline the verbs if you are after procedural code, the nouns if you aim for an object –oriented program.

Example 1: Supermarket Prize Scheme

- Supermarket needs to develop software to encourage regular customers.
- Customer needs to supply his residence address, telephone number, and the driving licence number.
- Each customer who registers is assigned a unique customer number (CN) by the computer.

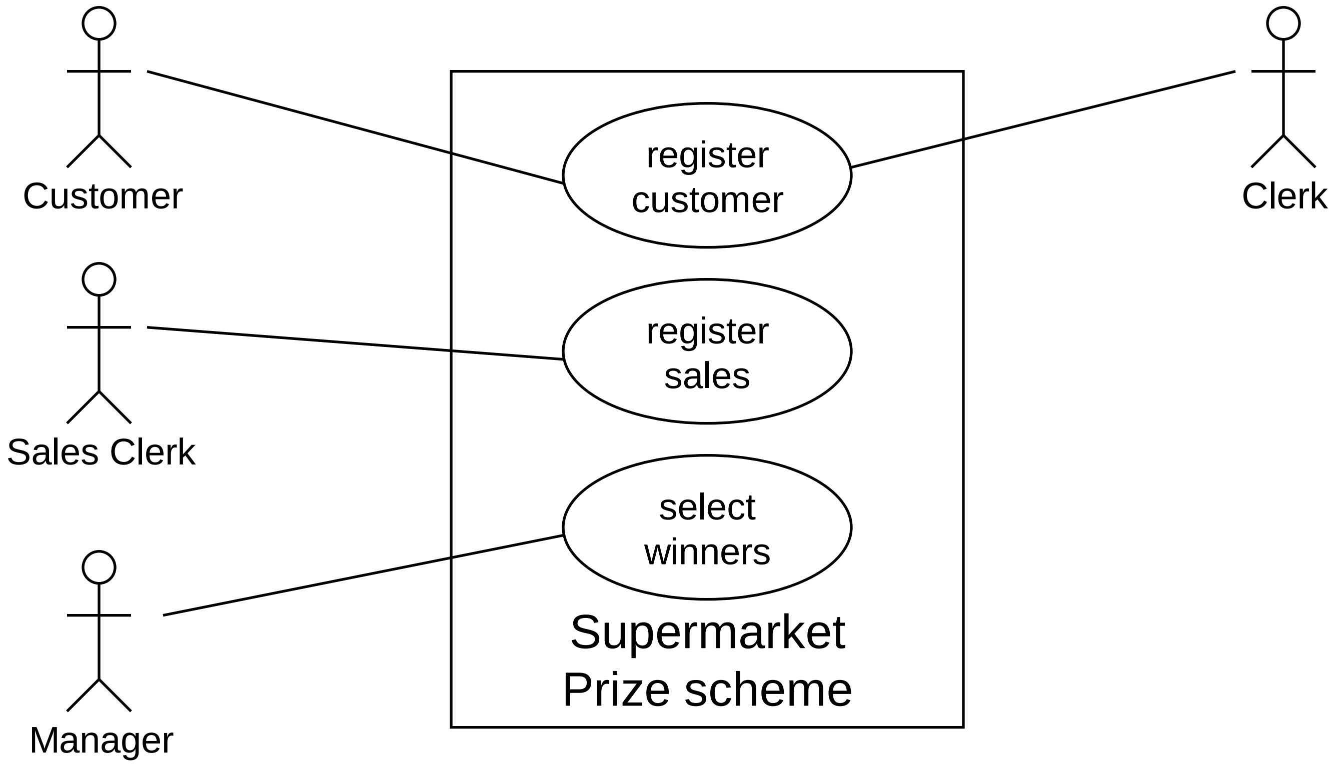
Example 1: Supermarket Prize Scheme

- A customer can present his CN to the staff when he makes any purchase.
- The value of his purchase is credited against his CN.
- At the end of each year, the supermarket awards surprise gifts to ten customers who make highest purchase.

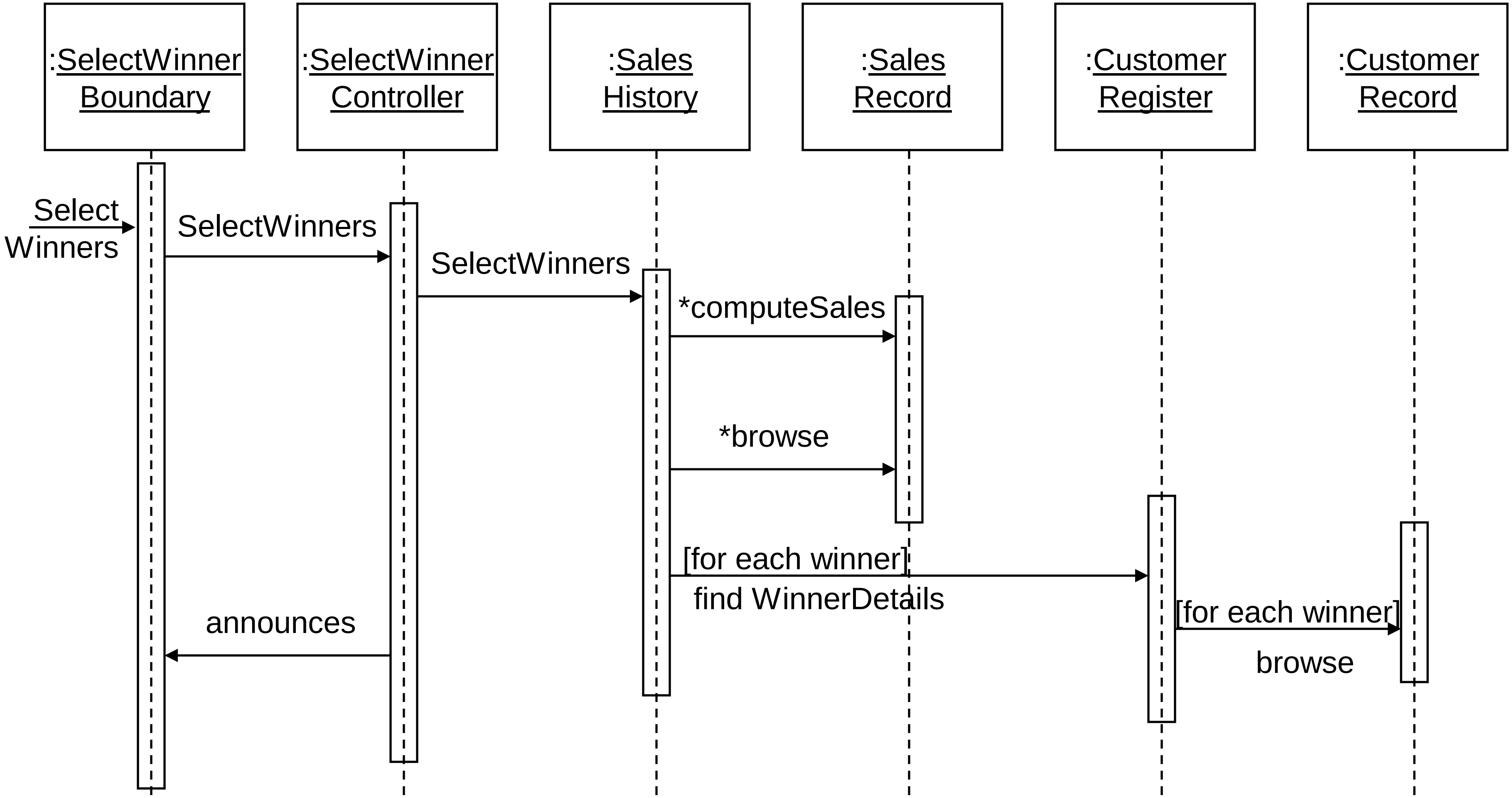
Example 1: Supermarket Prize Scheme

- **Also, it awards a 22 carat gold coin to every customer whose purchases exceed Rs. 10,000.**
- **The entries against the CN are reset on the last day of every year after the prize winner's lists are generated.**

Example 1: Use Case Model

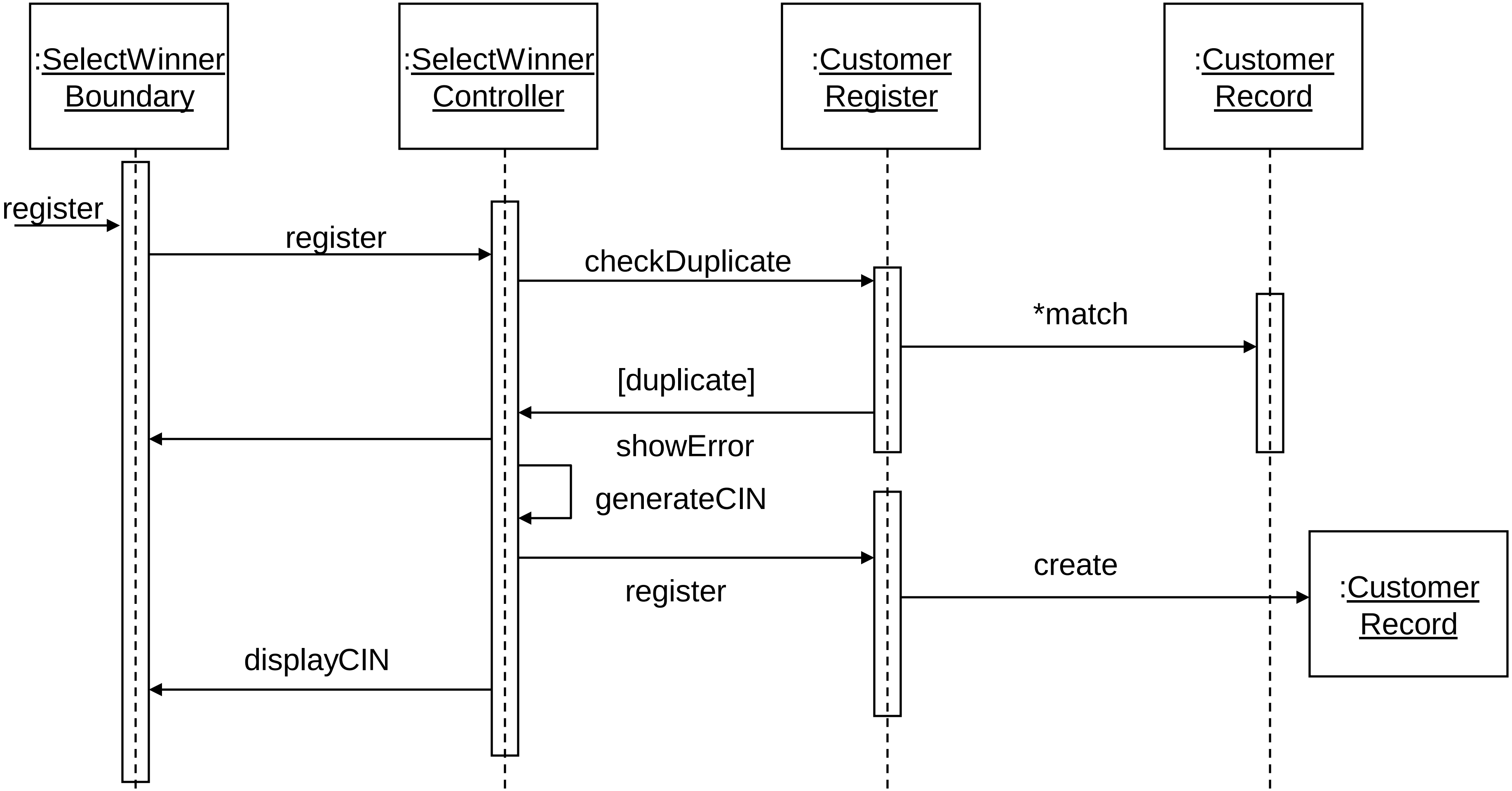


Example 1: Sequence Diagram for the Select Winners Use Case



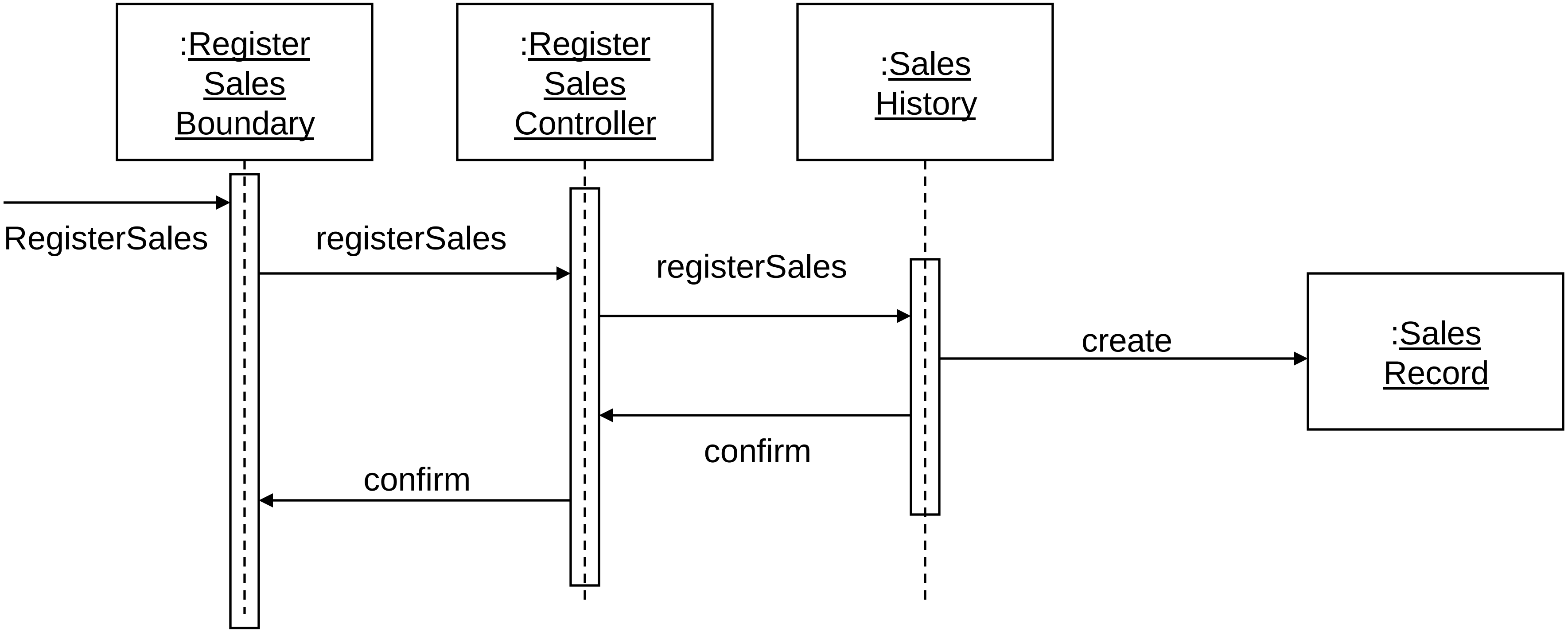
Sequence Diagram for the select winners use case

Example 1: Sequence Diagram for the Register Customer Use Case



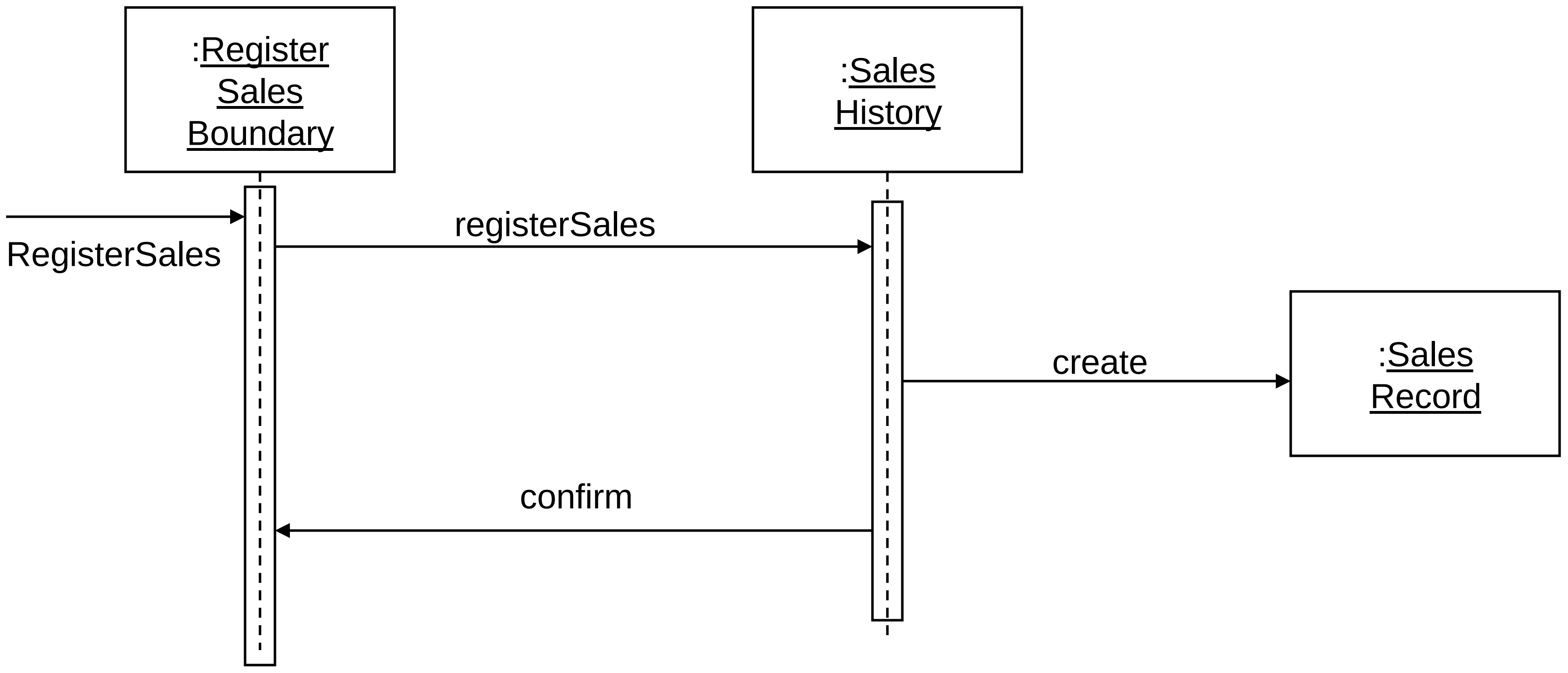
Sequence Diagram for the register customer use case

Example 1: Sequence Diagram for the Register Sales Use Case



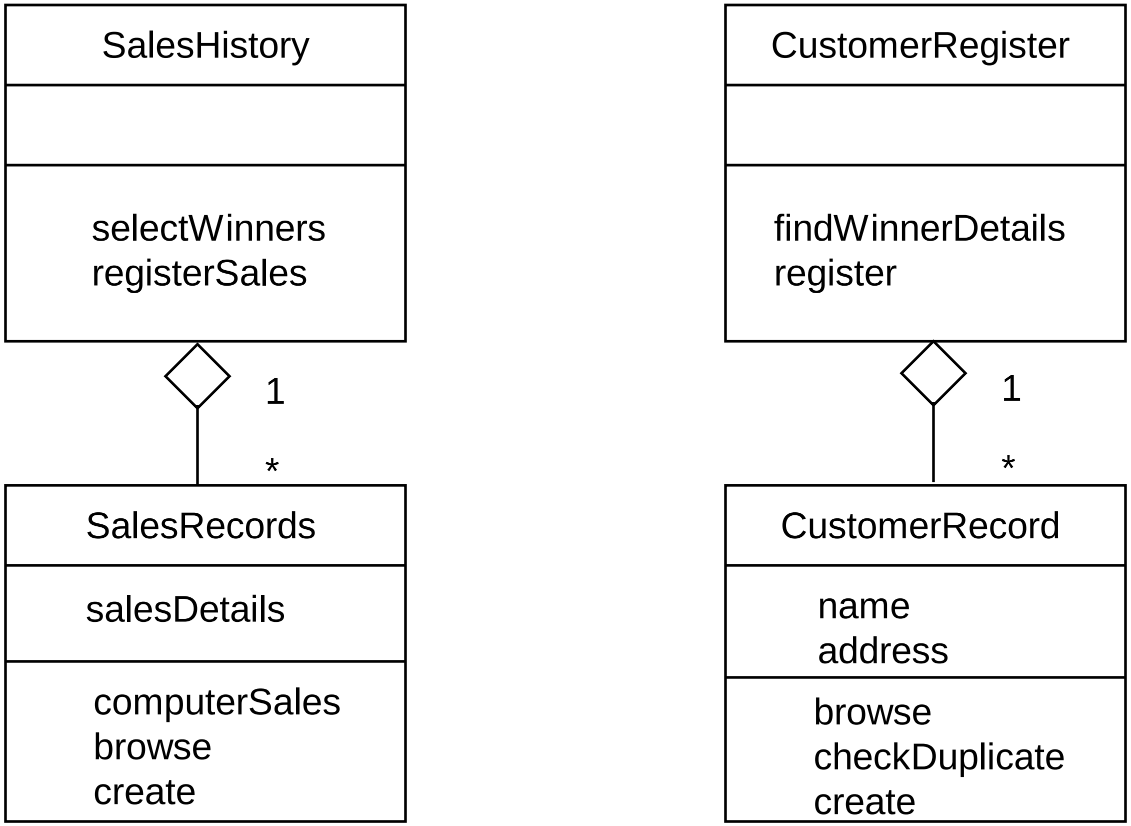
Sequence Diagram for the register sales use case

Example 1: Sequence Diagram for the Register Sales Use Case



Refined Sequence Diagram for the register sales use case

Example 1: Class Diagram



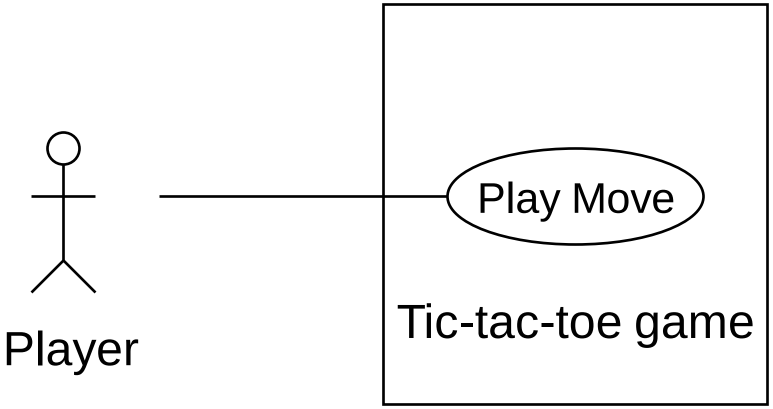
Example 2: Tic-Tac-Toe Computer Game

- A human player and the computer make alternate moves on a 3 3 square.
- A move consists of marking a previously unmarked square.
- The user inputs a number between 1 and 9 to mark a square
- Whoever is first to place three consecutive marks along a straight line (i.e., along a row, column, or diagonal) on the square wins.

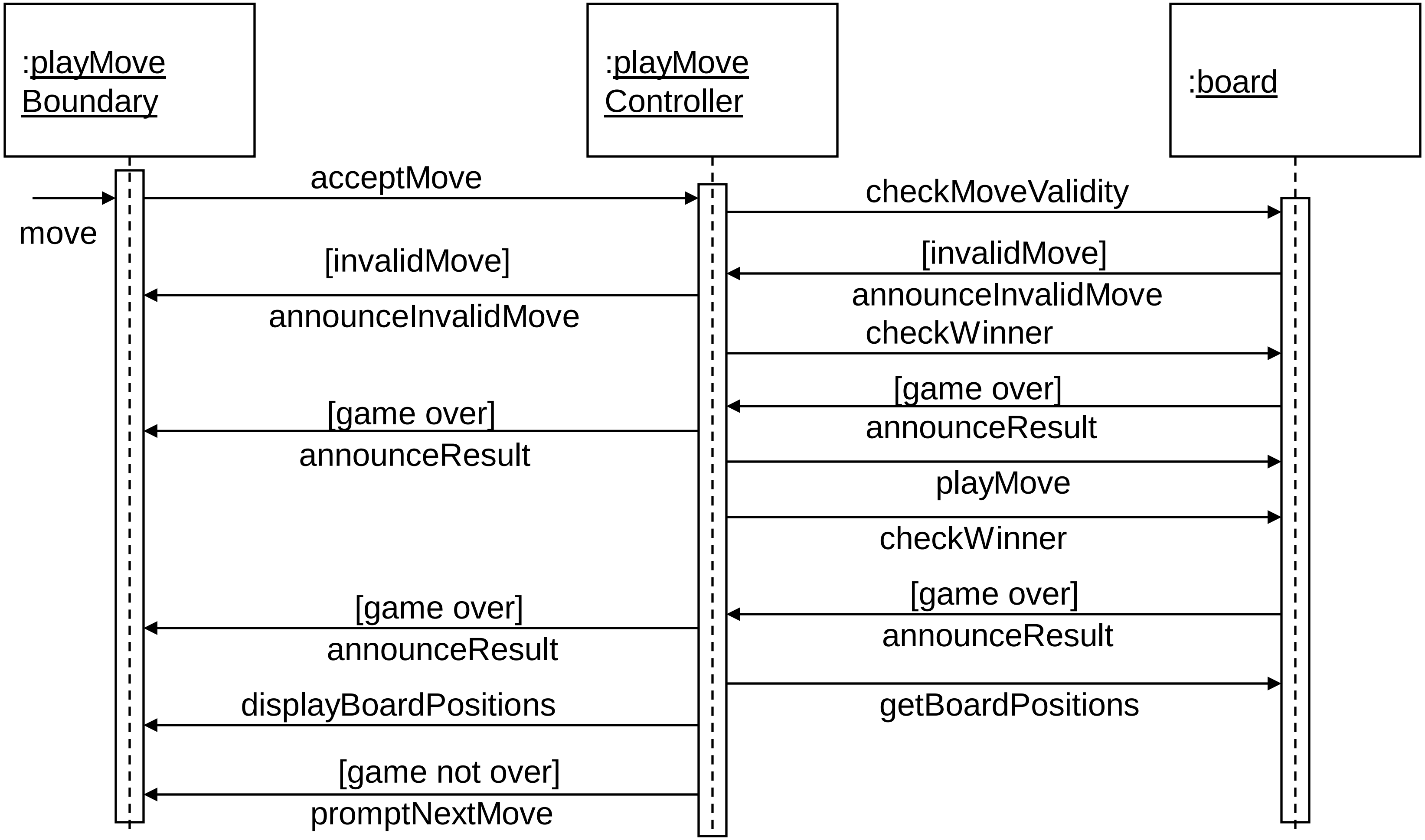
Example 2: Tic-Tac-Toe Computer Game

- As soon as either of the human player or the computer wins,
  - a message announcing the winner should be displayed.
- If neither player manages to get three consecutive marks along a straight line,
  - and all the squares on the board are filled up,
  - then the game is drawn.
- The computer always tries to win a game.

Example 2: Use Case Model

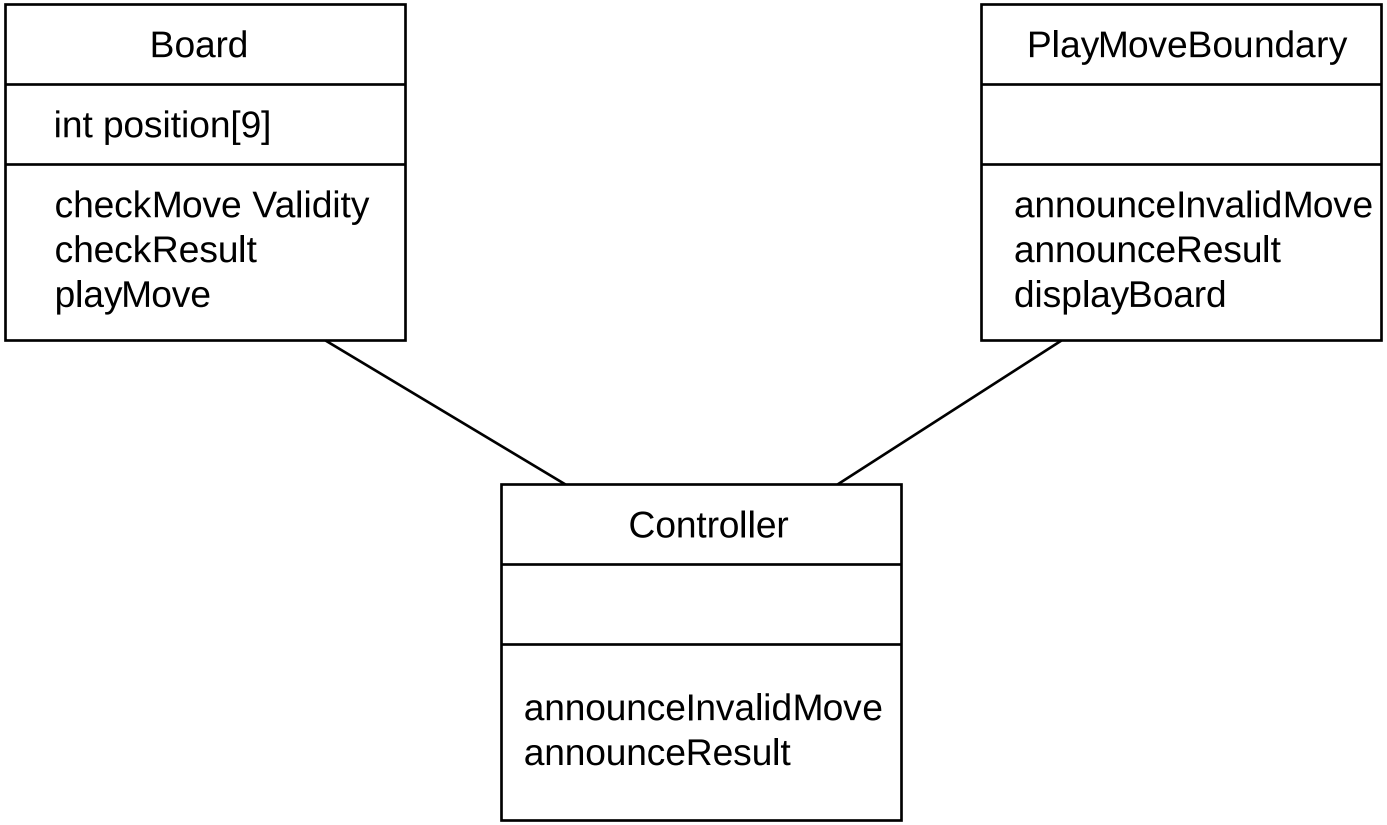


Example 2: Sequence Diagram



Sequence Diagram for the play move use case

Example 2: Class Diagram



Board

int position[9]

checkMove Validity
checkResult
playMove

PlayMoveBoundary

announceInvalidMove
announceResult
displayBoard

Controller

announceInvalidMove
announceResult

Chapter 8

Design Patterns

Design Patterns

- Standard solutions to commonly recurring problems
- Provides a good solution to model
- Pattern has four important parts
  - The **problem**
  - The **context (problem)**
  - The **solution**
  - The **context (solution)**

Types of patterns

1. **Architectural Patterns:** high level strategies that concern with overall solutions to large scale problems.
2. **Design Patterns:** Medium level strategies, describe str. and behaviour of entities.
3. **Idioms:** low level strategies and language specific programming solutions.

Pros and Cons of design patterns

- Provide standard vocabulary that provide understanding and communication in new design ideas.
- Help designer to produce flexible, efficient and well maintained design.
- Reduce number of design iterations, improve design productivity and reduce overall efforts.

Chapter 9

User Interface Design

Characteristics of a user interface

1. **Speed of learning.** A good user interface should be easy to learn. Speed of learning complex syntax and semantics of the command issue procedures. A good user interface should not require its users to memorize commands.

E.g.: use of metaphors, consistent and component based interface.

2. **Speed of use.** Speed of use of a user interface is determined by the time and user effort necessary to initiate and execute different commands.
3. **Speed of recall.** Once users learn how to use an interface, the speed with which they can recall the command issue procedure should be maximized.

4. **Error prevention.** A good user interface should minimize the scope of committing errors while initiating different commands.
5. **Attractiveness.** A good user interface should be attractive to use. In this respect, graphics-based user interfaces have a definite advantage over text-based interfaces.
6. **Consistency.** The commands supported by a user interface should be consistent. Consistency facilitates speed of learning, speed of recall, and also helps in reduction of error rate.
7. **Feedback.** A good user interface must provide feedback to various user actions.

8. **Error recovery (undo facility).** While issuing commands, even the expert users can commit errors. Therefore, a good user interface should allow a user to undo a mistake committed by him while using the interface.
9. **User guidance and on-line help.** Users seek guidance and on-line help when they either forget a command or are unaware of some features of the software. Whenever users need guidance or seek help from the system, they should be provided with the appropriate guidance and help.

BASIC CONCEPTS:

Mode-based interface vs. modeless interface

- A mode is a state or collection of states in which only a subset of all user interaction tasks can be performed.
- In a modeless interface, the same set of commands can be invoked at any time during the running of the software. Thus, a modeless interface has only a single mode and all the commands are available all the time during the operation of the software.
- On the other hand, in a mode-based interface, different set of commands can be invoked depending on the mode in which the system is.
- Caps lock and insert keys

Graphical User Interface vs. Text-based User Interface

- In a GUI multiple windows with different information can simultaneously be displayed on the user screen. This is perhaps one of the biggest advantages of GUI over text-based interfaces.
- Iconic information representation and symbolic information manipulation is possible in a GUI.

- A GUI usually supports command selection using an attractive and user-friendly menu selection system.
- In a GUI, a pointing device such as a mouse or a light pen can be used for issuing commands. The use of a pointing device increases the efficacy issue procedure.

Some advantages about TBUI

- On the other hand, a text-based user interface can be implemented even on a cheap alphanumeric display terminal.
- Graphics terminals are usually much more expensive than alphanumeric terminals.

Types of user interfaces

- User interfaces can be classified into the following three categories:
  1. Command language based interfaces
  2. Menu-based interfaces
  3. Direct manipulation interfaces

Command Language-based Interface

- is based on designing a command language which the user can use to issue the commands.
- The user is expected to frame the appropriate commands in the language and type them in appropriately whenever required.
- A command language-based interface can be made concise requiring minimal typing by the user.
- Command language-based interfaces allow fast interaction with the computer and simplify the input of complex commands.

Menu-based Interface

- An important advantage of a menu-based interface over a command language-based interface is that a menu-based interface does not require the users to remember the exact syntax of the commands.
- A menu-based interface is based on recognition of the command names, rather than recollection.

- Further, in a menu-based interface **the typing effort is minimal** as most interactions are carried out through menu selections using a pointing device.

Techniques are:

1). Scrolling menu

2). Walking menu

Direct Manipulation Interfaces

- Direct manipulation interfaces present the interface to the user **in the form of visual models** (i.e. icons or objects).
- For this reason, direct manipulation interfaces are sometimes called as **iconic interface.**
- In this type of interface, the user issues commands by performing actions on the visual representations of the objects, **e.g.** pull an icon representing a file into an icon representing a trash box, for deleting the file.

Important advantages of iconic interfaces include the fact that the icons can be recognized by the users very easily, and that icons are **language-independent.**

Coding

- At the end of the design phase we have:
  - module structure of the system
  - module specifications:
    - data structures and algorithms for each module.
- Objective of coding phase:
  - transform design into code
  - unit test the code.

Coding Standards

- Good software development organizations require their programmers to:
  - adhere some standard style of coding
  - called coding standards.

Coding Standards

- Many software development organizations:
  - formulate their own coding standards that suits them most,
  - require their engineers to follow these standards.

Coding Standards

- Advantage of adhering to a standard style of coding:
  - it gives a uniform appearance to the codes written by different engineers,
  - it enhances code understanding,
  - encourages good programming practices.

Coding Standards

- A coding standard
  - sets out standard ways of doing several things:
    - the way variables are named,
    - code is laid out,
    - maximum number of source lines allowed per function, etc.

Coding guidelines

- Provide general suggestions regarding coding style to be followed.

Code inspection and code walk throughs

- After a module has been coded,
  - code inspection and code walk through are carried out
  - ensures that coding standards are followed
  - helps detect as many errors as possible before testing.

Code inspection and code walk throughs

- Detect as many errors as possible during inspection and walkthrough:
  - detected errors require less effort for correction
    - much higher effort needed if errors were to be detected during integration or system testing.

Coding Standards and Guidelines

- Good organizations usually develop their own coding standards and guidelines:
  - depending on what best suits their organization.

Representative Coding Standards

- Rules for limiting the use of globals:
  - what types of data can be declared global and what can not.
- Naming conventions for
  - global variables,
  - local variables, and
  - constant identifiers.

Representative Coding Standards

- Contents of headers for different modules:
  - The headers of different modules should be standard for an organization.
  - The exact format for header information is usually specified.

Representative Coding Standards

- Header data:
  - Name of the module,
  - date on which the module was created,
  - author's name,
  - modification history,
  - synopsis of the module,
  - different functions supported, along with their input/output parameters,
  - global variables accessed/modified by the module.

- **Naming conventions for global variables, local variables, and constant identifiers:** A possible naming convention can be that global variable names always start with a capital letter, local variable names are made of small letters, and constant names are always capital letters.

Representative Coding Standards

- Error return conventions and exception handling mechanisms.
  - the way error and exception conditions are handled should be standard within an organization.
  - For example, when different functions encounter error conditions
    - should either return a 0 or 1 consistently.

Representative Coding Guidelines

- Do not use too clever and difficult to understand coding style.
  - Code should be easy to understand.
- Many inexperienced engineers actually take pride:
  - in writing cryptic and incomprehensible code.

Representative Coding Guidelines

- Clever coding can obscure meaning of the code:
  - hampers understanding.
  - makes later maintenance difficult.
- Avoid obscure side effects.

Representative Coding Guidelines

- Code should be well-documented.
- Rules of thumb:
  - on the average there must be at least one comment line
    - for every three source lines.
  - The length of any function should not exceed 10 source lines.

Representative Coding Guidelines

- Lengthy functions:
  - usually very difficult to understand
  - probably do too many different things.

Representative Coding Guidelines

- Do not use goto statements.
- Use of goto statements:
    - make a program unstructured
    - make it very difficult to understand.

Code review

- Code review for a model is carried out after the module is successfully compiled and the all the syntax errors have been eliminated.
- Normally, two types of reviews are carried out on the code of a module.
- These two types code review techniques are code inspection and code walk through.

Code Walk Through

- An informal code analysis technique.
  - undertaken after the coding of a module is complete.
- A few members of the development team select some test cases:
  - simulate execution of the code by hand using these test cases.
- Discussion should focus on discovery of errors:
  - and not on how to fix the discovered errors.

- The main objectives of the walk through are to discover the algorithmic and logical errors in the code.
- The members note down their findings to discuss these in a walk through meeting where the coder of the module is present.
- The team performing code walk through should not be either too big or too small.
  - Ideally, it should consist of between three to seven members.

Code Inspection

- In contrast to code walk through, the aim of code inspection is to discover some common types of errors caused due to oversight and improper programming.
- In addition to the commonly made errors, adherence to coding standards is also checked during code inspection.
- Good software development companies collect statistics regarding different types of errors commonly committed by their engineers and identify the type of errors most frequently committed.
- Such a list of commonly committed errors can be used during code inspection to look out for possible errors.

Commonly made errors

- Use of uninitialized variables.
- Nonterminating loops.
- Array indices out of bounds.
- Improper storage allocation and deallocation.
- Actual and formal parameter mismatch in procedure calls.
- Jumps into loops.

Code Inspection

- Use of incorrect logical operators
  - or incorrect precedence among operators.
- Improper modification of loop variables.
- Comparison of equality of floating point values, etc.
- Also during code inspection,
  - adherence to coding standards is checked.

Programming (Coding) Style & Conventions

- Check for errors early and often.
- Return from errors immediately.
- Have you checked for compiler warnings? Warnings often point to real bugs.
- If possible reduce object and file dependencies.
- Eliminate needless import or include statements.
- Check again for warnings or errors before committing source code.