2. Software Requirements and analysis

Prepared By
Prof.Jigar Dave
SOE
R K University
8469766496

2.1 Introduction to requirement engineering

- The process to gather the software requirements from client, analyze, and document them is known as requirement engineering.
- The process of collecting the software requirement from the client then understand, evaluate and document it is called as requirement engineering.
- Requirement engineering makes a bridge for design and construction

❖ Types Of Requirement Engineering[task of requirement Engineering]

- Requirement engineering consists of seven different tasks as follow:
- 1. Inception
- 2. Elicitation
- 3. Elaboration
- 4. Negotiation
- 5. Specification
- 6. Validation
- 7. Requirement management

1. Inception

- Inception is a task where the requirement engineering asks a set of questions to establish a software process.
- In this task, it understands the problem and evaluates with the proper solution.
- It collaborates with the relationship between the customer and the developer.
- The developer and customer decide the overall scope and the nature of the Project.

2. Elicitation

- Elicitation means to find the requirements from anybody.
- The requirements are difficult because the **following problems occur in elicitation**.

- **Problem of scope**: The customer give the unnecessary technical detail rather than clarity of the overall system objective.
- **Problem of understanding**: Poor understanding between the customer and the developer regarding various aspect of the project like capability, limitation of the computing environment.
- **Problem of fix req**: In this problem, the requirements change from time to time and it is difficult while developing the project.

3. Elaboration

- Elaboration means “to work out in detail”.
- In this task, the information taken from user during inception and elaboration and are expanded and refined in elaboration.
- Its main task is developing pure model of software using functions, feature and constraints of a software.
- Thus , elaboration is an “analysis modeling action”. This model focuses on how the end user will ‘interact with the system’

4. Negotiation

- In negotiation task, a software engineer decides the how will the project be achieved with limited business resources.
- To create rough guesses of development and access the impact of the requirement on the project cost and delivery time.

5. Specification

- In this task, the requirement engineer builds a final work product.
- The work product is in the form of software requirement specification(SRS).
- In this task, formalize the requirement of the proposed software such as informative, functional and behavioral.
- The requirement are formalize in both graphical and textual formats.

6. Validation

- The work product is built as an output of the requirement engineering and that is accessed for the quality through a validation step.
- The formal technical reviews from the software engineer, customer and other investors helps for the primary requirements validation mechanism.
- In this task developer ,customer will validate Product is accurate or not.(in short testing)

7. Requirement management

- It is a set of activities that help the project team to identify, **control and track the requirements** and changes can be made to the requirements at any time of the ongoing project.
- These tasks **start with the identification** and assign a **unique identifier** to each of the requirement.
- After finalizing the requirement **trackable table** is developed.
- The examples of **traceability table are the features, sources, dependencies, subsystems and interface of the requirement.**
- activities that help project team to identify, control, and track requirements and changes as project proceeds, similar to **software configuration management (SCM)** techniques

2.2 Eliciting Requirements

- Eliciting requirement helps the user for collecting the requirement.
- **Eliciting requirement steps are as follows:**
- 1. Collaborative requirements gathering.
- 2. Quality Function Deployment (QFD).
- 3. Usage scenarios.
- 4. Elicitation work product.

1. Collaborative requirements gathering

- Gathering the requirements by conducting the meetings between developer and customer.
- Fix the rules for **preparation** and **participation**.
- The main motive is to **identify the problem**, **give the solutions** for the elements, negotiate the different approaches and specify the primary set of solution requirements in an environment which is valuable **for achieving goal**.

2. Quality Function Deployment (QFD)

- In this technique, translate the customer need into the technical requirement for the software.
- QFD system designs a software according to the demands of the customer.
- **QFD consist of three types of requirement:**
- **1. Normal requirements**
- **2. Expected requirement**
- **3. Exciting requirements**

- **Normal requirements** – The objective and goal are stated for the system through the meetings with the customer.
- For the **customer satisfaction** these requirements should be there.
- **Expected requirement** –
- These are the basic requirement that not be **clearly told by the customer**, but also the customer expect that requirement.
- **Exciting requirements** – These features are beyond the **expectation of the customer**.
- The developer adds some additional features or unexpected feature into the software to make the customer more satisfied.
  For example, the mobile phone with standard features, but the developer adds few additional functionalities like voice searching, multi-touch screen etc. then the customer more exited about that feature.

3. Usage scenarios

- Till the software team does not understand how the features and function are used by the end .
- To achieve above problem the software team produces a **set of structure that identify** the usage for the software.
- This structure is called as '**Use Cases**'.

4. Elicitation work product

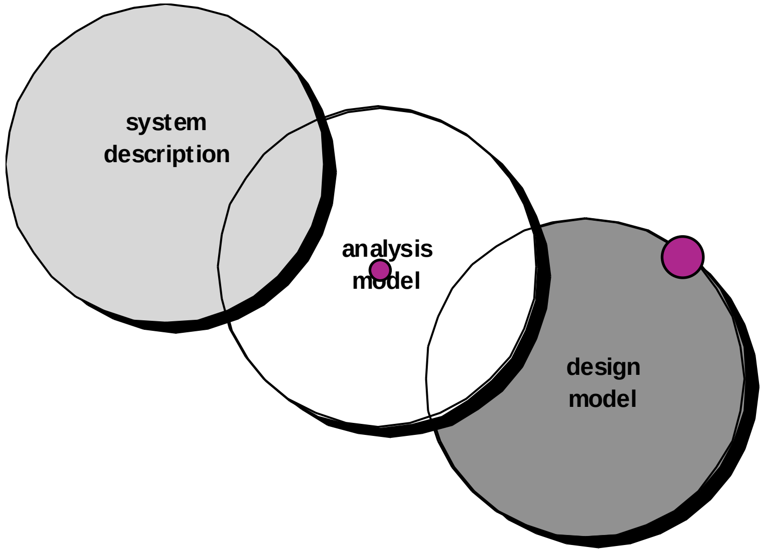
- The work product created as a result of requirement elicitation that is depending on the size of the system or product to be built.
- The work product consists of a statement need, feasibility, statement scope for the system.
- It also consists of a list of users participate in the requirement elicitation.

2.3 Requirements Analysis

- Requirements analysis
  - specifies software’s ***operational*** characteristics
  - indicates software's ***interface*** with other system elements
  - establishes ***constraints*** that software must meet
- Requirements analysis allows the software engineer (called an *analyst* or *modeler* in this role) to:
  - ***elaborate*** on basic requirements established during earlier requirement engineering tasks
  - build ***models*** that show user scenarios, functional activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed.

A Bridge

Writing the Software Specification



Everyone knew exactly what had to be done until someone wrote it down!

Software Requirements

- Broadly software requirements should be categorized in these categories:
- 1. Functional Requirements
- 2. Non-Functional Requirements .
- 3. User Interface requirements .

Functional Requirements

- Requirements, which are related to functional aspect of software fall into this category.
- They define functions and functionality within and from the software system.
- EXAMPLES –
  - Search option given to user to search from various invoices.
  - User should be able to mail any report to management.
  - Users can be divided into groups and groups can be given separate rights.
  - Should comply business rules and administrative functions.
  - Software is developed keeping downward compatibility intact.

Non-Functional Requirements

- Requirements, which are not related to functional aspect of software, fall into this category.
- Non-functional requirements include –
  - Security
  - Logging
  - Storage
  - Configuration
  - Performance
  - Cost
  - Interoperability
  - Flexibility
  - Disaster recovery
  - Accessibility

- Requirements are categorized logically as:
- **Must Have** : Software cannot be said operational without them. (Core Features)
- **Should have** : Enhancing the functionality of software. (For better system)
- **Could have** : Software can still properly function with these requirements.
- **Wish list** : These requirements do not map to any objectives of software.
- While developing software, ‘Must have’ must be implemented, ‘Should have’ is a matter of debate with stakeholders, whereas ‘Could have’ and ‘Wish list’ can be kept for software updates

User Interface requirements

- User Interface (UI) is an important part of any software or hardware . A software is widely accepted if it is –
  - easy to operate
  - quick in response
  - effectively handling operational errors
  - providing simple yet consistent user interface

- User interface requirements are briefly mentioned below –

- Content presentation
- Easy Navigation
- Simple interface
- Responsive
- Consistent UI elements
- Feedback mechanism
- Default settings
- Purposeful layout
- Strategically use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

Requirement Engineering Process

- It is a four step process, which includes –
  - Feasibility Study
  - Requirement Gathering
  - Software Requirement Specification(SRS)
  - Software Requirement Validation (SRV)

Feasibility study

- When the client approaches the organization for getting the desired product developed, it comes up with a rough idea about what all functions the software must perform and which all features are expected from the software.
- Referencing to this information, the **analysts** do a **detailed study** about whether the desired system and its functionality are feasible to develop.
- This feasibility study is focused towards goal of the organization.

- This study analyzes whether the software product can be practically Devloped in terms of implementation, contribution of project to organization, cost constraints, and as per values and objectives of the organization.
- The output of this phase should be a **feasibility study report** that should contain acceptable comments and recommendations for management after the project should be started.

Requirement Gathering

- If the feasibility report is positive towards starting the project, next phase starts with gathering requirements from the user. **Analysts and engineers communicate with the client and end-users** to know their ideas on what the software should provide and **which features they want the software to include**

Software Requirement Specification (SRS)

- SRS is a document created by system analyst after the requirements are collected from various investors.
- SRS defines how the intended software will interact with hardware, external devices, speed of operation, response time of system, portability of software across various platforms, maintainability, Security, Quality, Limitations etc.
- The requirements received from client are written in natural language. **It is the responsibility of the system analyst to document the requirements in technical language so that they can be understood and used by the software development team.**

- SRS should come up with the following features:
  - User Requirements are expressed in natural language.
  - Technical requirements are expressed in structured language, which is used inside the organization.
  - Design description should be written in Pseudo code. (Pseudocode is an artificial and informal language that helps programmers develop algorithms. Pseudocode is a "text-based" detail (algorithmic) design tool.)

    ex :-

    ```
     If student's grade is greater than or equal to 60
            • Print "passed"
    else
            • Print "failed"
    ```
  - Format of Forms and GUI screen prints.
  - Conditional and mathematical notations for DFDs etc.

Value of Good SRS

- A basic purpose of the SRS is to bridge this communication gap so they have a shared vision of the software being built. Hence, one of the main advantages of a good SRS is:
  - An SRS establishes the basis for agreement between the client and the supplier on what the software product will do.
  - An SRS provides a reference for validation of the final product.
  - A high-quality SRS is a prerequisite to high-quality software.
  - A high-quality SRS reduces the development cost.

Properties of Good SRS

- The important properties of a good SRS document are the following:
- **Short.** The SRS document should be short and at the same time clear, reliable, and complete. Talkative
- **Structured.** It should be well-structured. A well-structured document is easy to understand and modify. In practice, the SRS document undergoes several revisions to manage with the customer requirements. Often, the customer requirements evolve over a period of time. Therefore, in order to make the modifications to the SRS document easy, it is important to make the document well-structured.

- **Black-box view.** It should only specify what the system should do .This means that the SRS document should specify the external behavior of the system and not discuss the implementation issues. The SRS document should view the system to be developed as black box, and should specify the externally visible behavior of the system. For this reason, the SRS document is also called the black-box specification of a system.
- **Conceptual integrity.** It should show conceptual integrity so that the reader can easily understand it.
- **Response to undesired events.** It should characterize acceptable responses to undesired events. These are called system response to exceptional conditions.

- **Verifiable.** All requirements of the system as documented in the SRS document should be verifiable. This means that it should be possible to determine whether or not requirements have been met in an implementation.

Problems without a SRS document

- Without developing the SRS document, the system would not be implemented according to customer needs.
- Software developers would not know whether what they are developing is what exactly required by the customer.
- Without SRS document, it will be very much difficult for the maintenance engineers to understand the functionality of the system.
- It will be very much difficult for user document writers to write the users' manuals properly without understanding the SRS document.

Problems with an unstructured specification

- It would be very much difficult to understand that document.
- It would be very much difficult to modify that document.
- Conceptual integrity in that document would not be shown.
- The SRS document might be unambiguous and inconsistent.

Characteristics of SRS

- Correct
- Clear-cut
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Components of SRS

- Completeness of specifications is difficult to **achieve** and even more difficult to **verify**. Having guidelines about what different things an SRS should specify will help in completely specifying the requirements. Here we describe some of the system properties than an SRS should specify.
- **The basic issues an SRS must address**
  - 1. Functionality
  - 2. Performance
  - 3. Design constraints imposed on an implementation
  - 4. External interfaces

Software Requirement Validation

- After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements by mistake.
- Requirements can be checked against following conditions -
  - **If they can be practically implemented**
  - **If they are valid and as per functionality and domain of software**
  - **If there are any doubts**
  - **If they are complete**
  - **If they can be demonstrated**

2.4 Elements of analysis model

- Analysis model operates as a link between the 'system description' and the 'design model'.
- In the analysis model, information, functions and the behaviour of the system is defined and these are translated into the architecture, interface and component level design in the 'design modeling'.

Elements of the analysis model

- 1. Scenario based element
- 2. Class based elements
- 3. Behavioral elements
- 4. Flow oriented elements

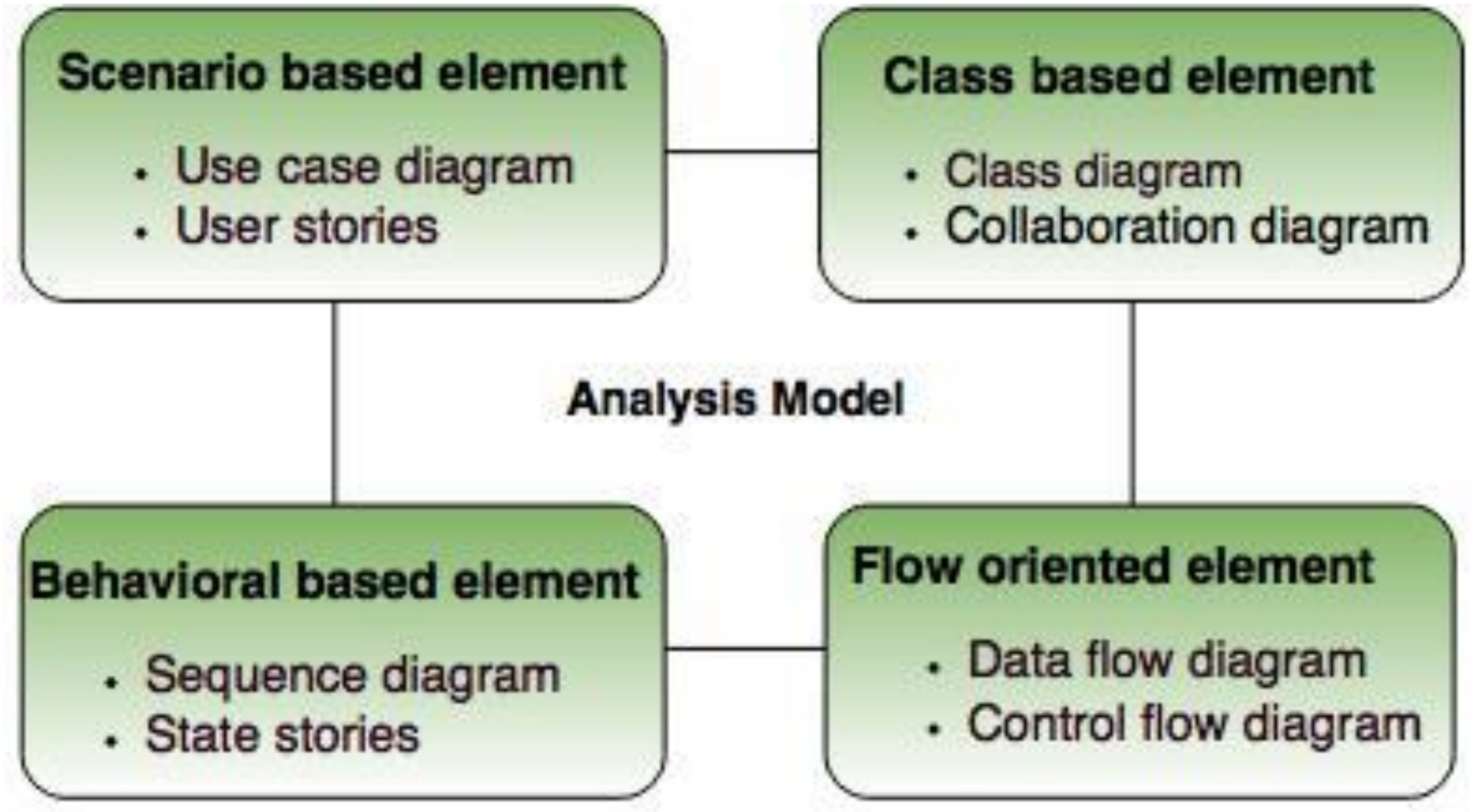
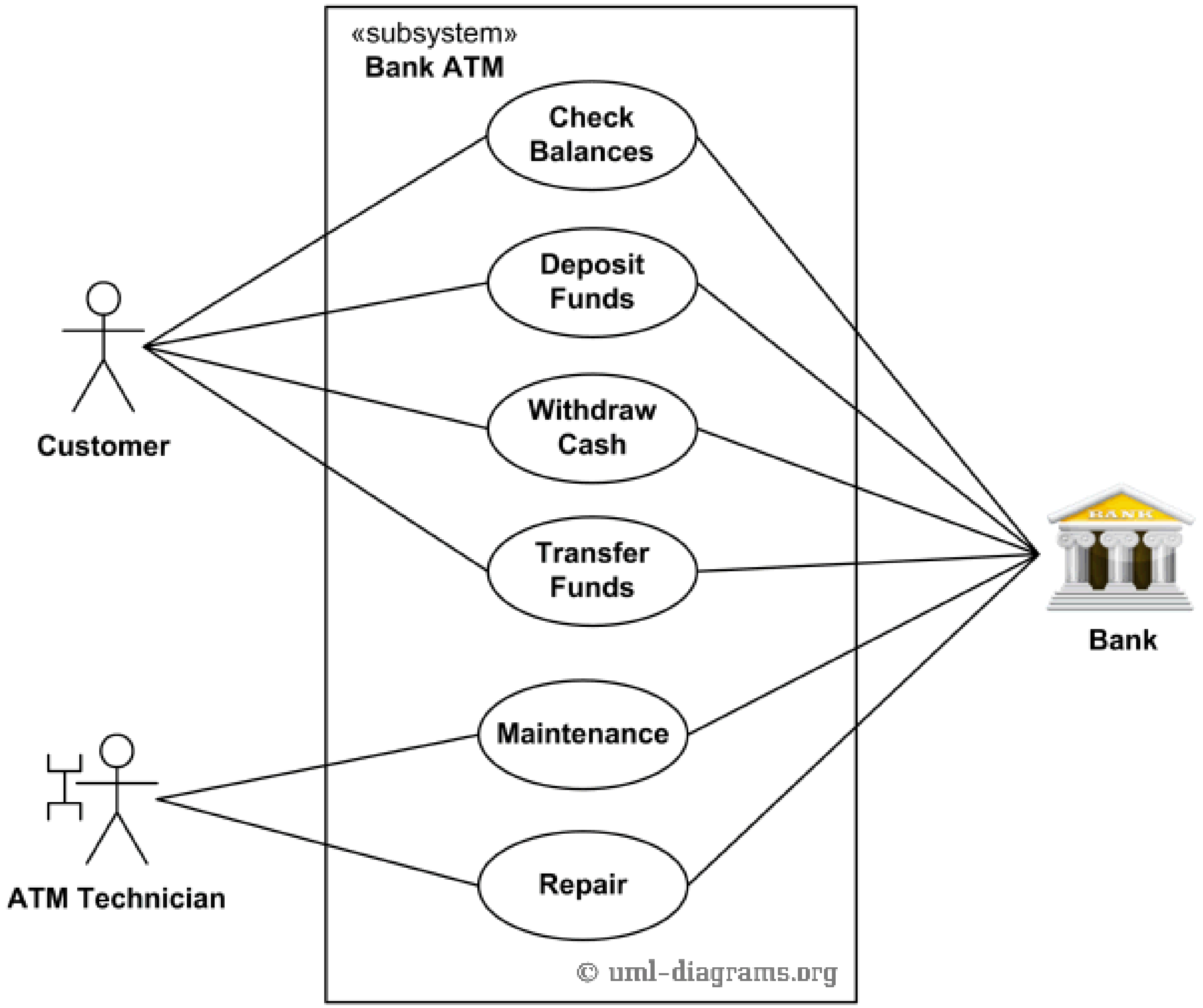


Fig. - Elements of analysis model

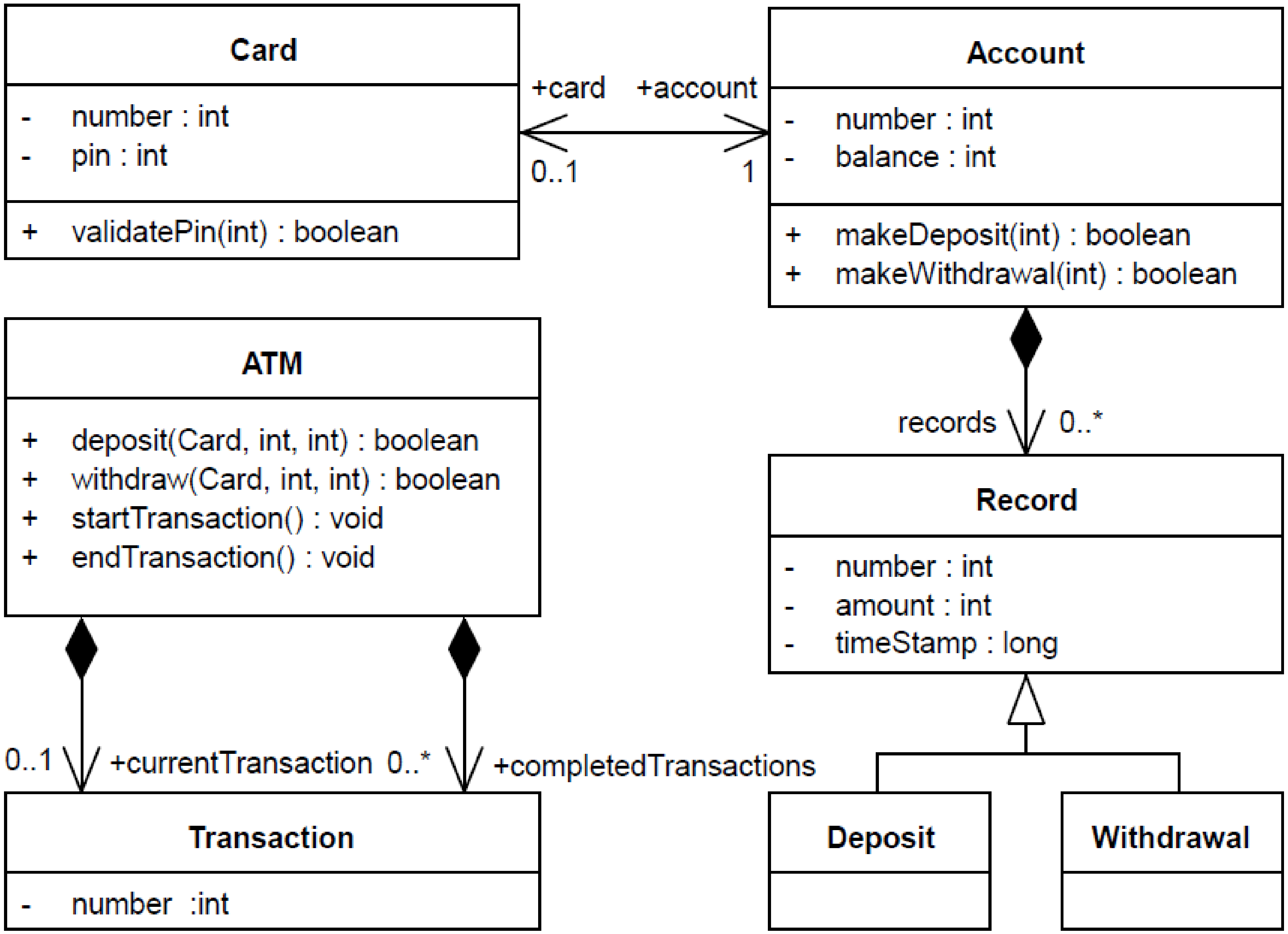
1. Scenario based element

- This type of element represents the system user point of view.
- Scenario based elements are use case diagram, user stories.


«subsystem»
Bank ATM
Check Balances
Deposit Funds
Withdraw Cash
Transfer Funds
Maintenance
Repair
Customer
ATM Technician
BANK
Bank
© uml-diagrams.org

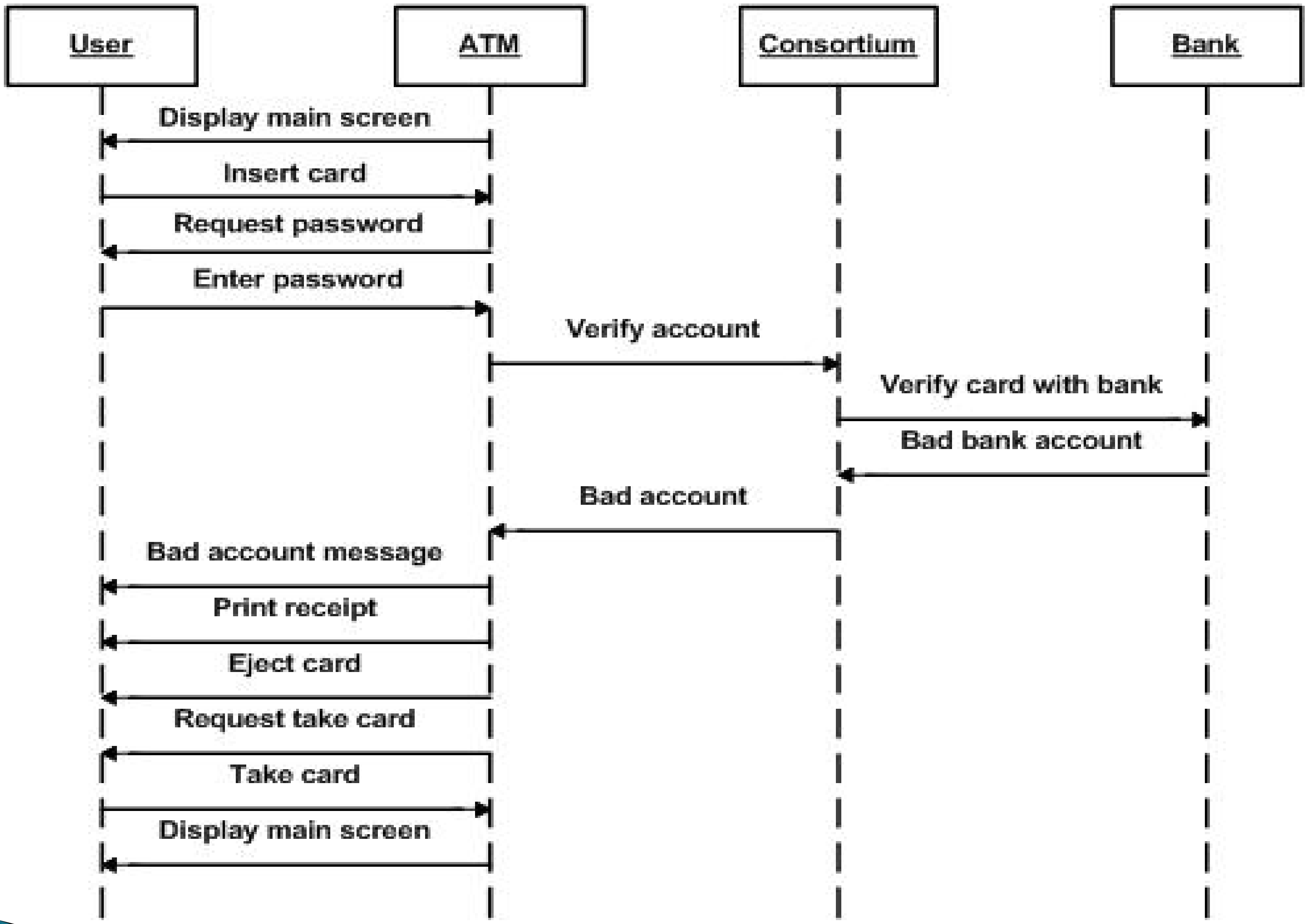
2. Class based elements

- The object of this type of element manipulated by the system.
- It defines the object , attributes and relationship.
- The collaboration is occurring between the classes.
- Class based elements are the class diagram, collaboration diagram.


Card
- number : int
- pin : int
+ validatePin(int) : boolean
+card
+account
0..1
1
Account
- number : int
- balance : int
+ makeDeposit(int) : boolean
+ makeWithdrawal(int) : boolean
records
0..*
Record
- number : int
- amount : int
- timeStamp : long
ATM
+ deposit(Card, int, int) : boolean
+ withdraw(Card, int, int) : boolean
+ startTransaction() : void
+ endTransaction() : void
0..1
+currentTransaction
0..*
+completedTransactions
Transaction
- number :int
Deposit
Withdrawal

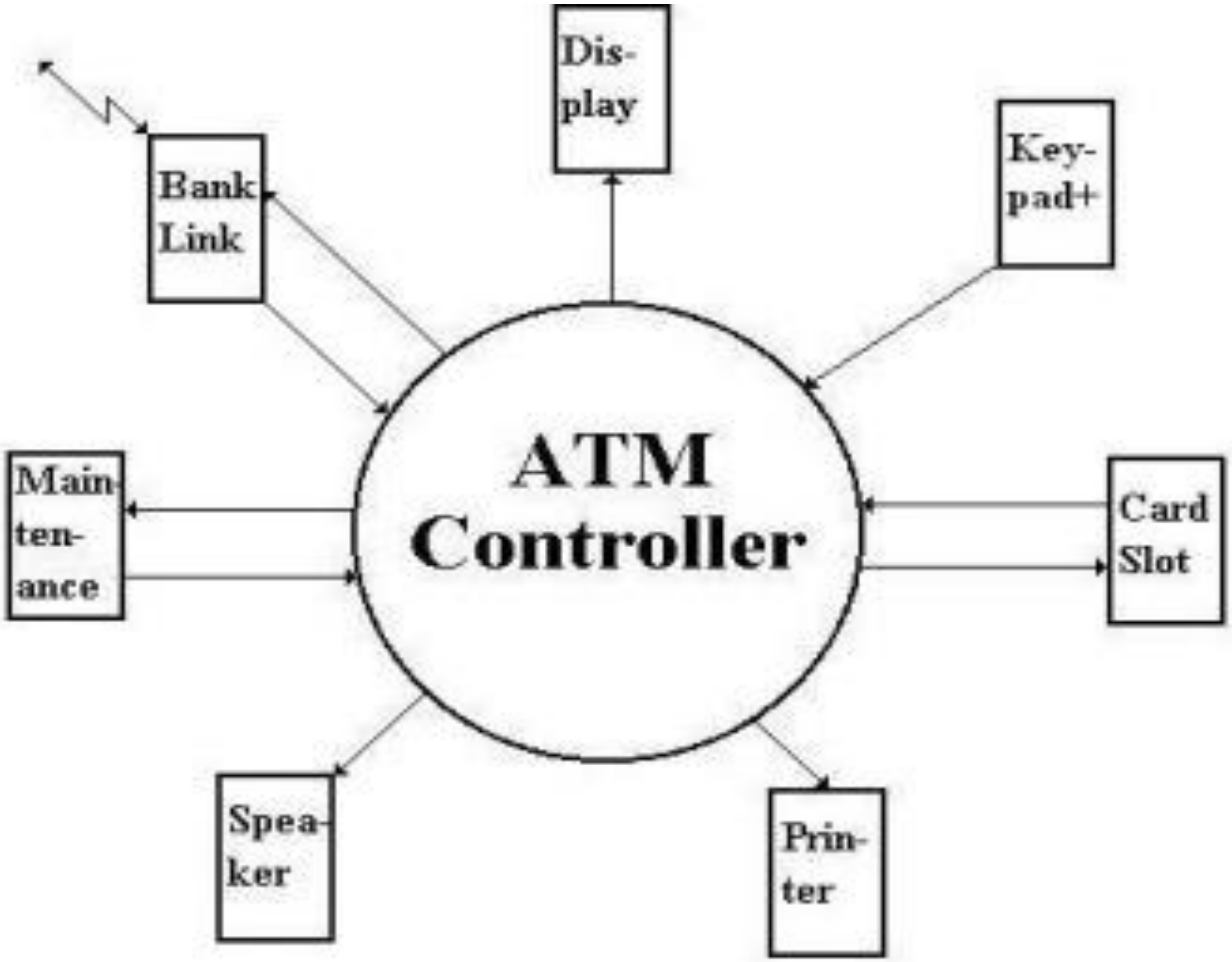
3. Behavioral elements

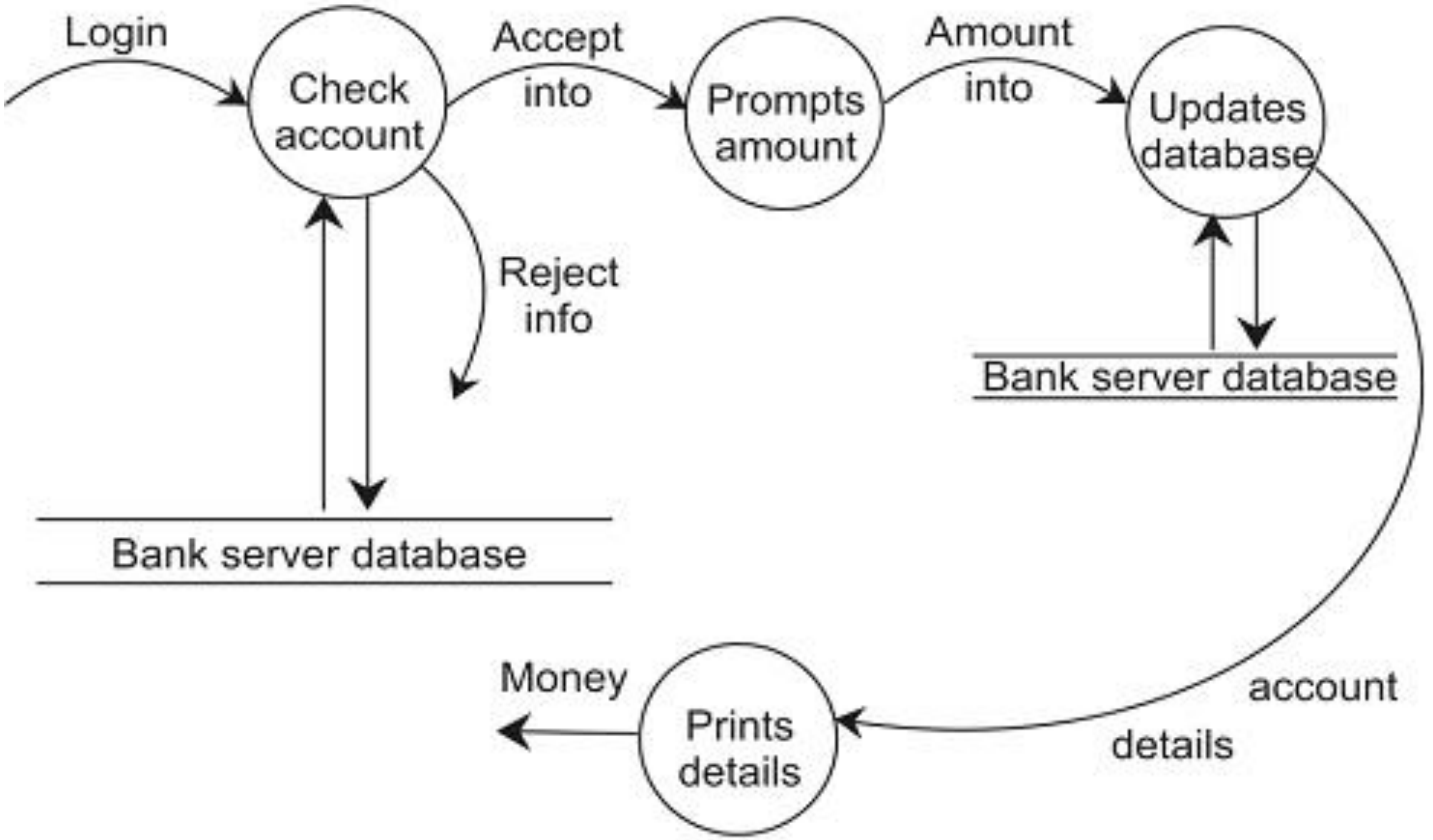
- Behavioral elements represent state of the system and how it is changed by the external events.
- The behavioral elements are **sequenced diagram, state diagram.**


User
ATM
Consortium
Bank
Display main screen
Insert card
Request password
Enter password
Verify account
Verify card with bank
Bad bank account
Bad account
Bad account message
Print receipt
Eject card
Request take card
Take card
Display main screen

4. Flow oriented elements

- An information flows through a computer-based system it gets transformed.
- It shows how the data objects are transformed while they flow between the various system functions.
- The flow elements are data flow diagram, control flow diagram.


Dis-
play
Bank
Link
Key-
pad+
ATM
Controller
Main-
ten-
ance
Card
Slot
Spea-
ker
Prin-
ter


Login
Check account
Accept into
Prompts amount
Amount into
Updates database
Reject info
Bank server database
Bank server database
account details
Money
Prints details

DFD - 1 - Level

2.5 Analysis Rules of Thumb

- The rules of thumb that must be followed while creating the analysis model.

  The rules are as follows:
- The model focuses on the requirements in the business domain. The level of abstraction must be high i.e there is no need to give details.
- Every element in the model helps in understanding the software requirement and focus on the information, function and behaviour of the system.
- The consideration of infrastructure and nonfunctional model delayed in the design.

- **For example,** the database is required for a system, but the classes, functions and behavior of the database are not initially required. If these are initially considered then there is a delay in the designing.
- Throughout the system minimum coupling is required. The interconnections between the modules is known as 'coupling'.
- The analysis model gives value to all the people related to model.
- The model should be simple as possible. Because simple model always helps in easy understanding of the requirement.

2.6 Data Modeling

- Analysis modeling starts with the data modeling.
- The software engineer defines all the data object that proceeds within the system and the relationship between data objects are identified.'
- It Contains following:
  - Data Objects
  - Data attributes
  - Relationship

Data objects

- The data object is the representation of composite information.
- The composite information means an object has a number of different properties or attribute.
  For example, Height is a single value so it is not a valid data object, but dimensions contain the height, the width and depth these are defined as an object.

What is a Data Object?

Object —something that is described by a set of attributes (data items) and that will be manipulated within the software (system)

- each instance of an object (e.g., a book) *can be identified uniquely* (e.g., ISBN #)
- each plays a *necessary* role in the system i.e., the system could not function without access to instances of the object
- each is described by *attributes* that are themselves data items

Data Attributes

- Each of the data object has a set of attributes.

object: automobile

attributes:
- make
- model
- body type
- price
- options code

object: mobile

attributes:
- Camera
  - features
  - applications
  - price
  - other things

Data object has the following

- **characteristics:**Name an instance of the data object.
- Describe the instance.
- Make reference to another instance in another table.

Relationship

- Relationship shows the relationship between data objects and how they are related to each other.
- Relationship is expressed as :
  - One to one (1:1)
  - One to many (1:M)
  - Many to many (M: M)
  - Modality (0)

- **One to one (1:1)**
  One event of an object is related to one event of another object.
  For example, one employee has only one ID.

- **One to many (1:M)**
  One event of an object is related to many events.
  For example, One collage has many departments.

- **Many to many(M:M)**
  Many events of one object are related to many events of another object.
  For example, many customer place order for many products.
- **Modality**
- If an event relationship is an optional then the modality of relationship is zero.
- If an event of relationship is compulsory then modality of relationship is one.

2.7 Functional modelling

- Functional Modelling gives the process perspective of the object-oriented analysis model and an overview of what the system is supposed to do.
- It defines the function of the internal processes in the system with the aid of Data Flow Diagrams (DFDs).

- The *functional model* addresses two processing elements of the WebApp, each
- representing a different level of procedural abstraction: **(1) user-observable functionality**
- that is delivered by the WebApp to end users, and (2) the operations contained within **analysis classes** that implement behaviors associated with the class.

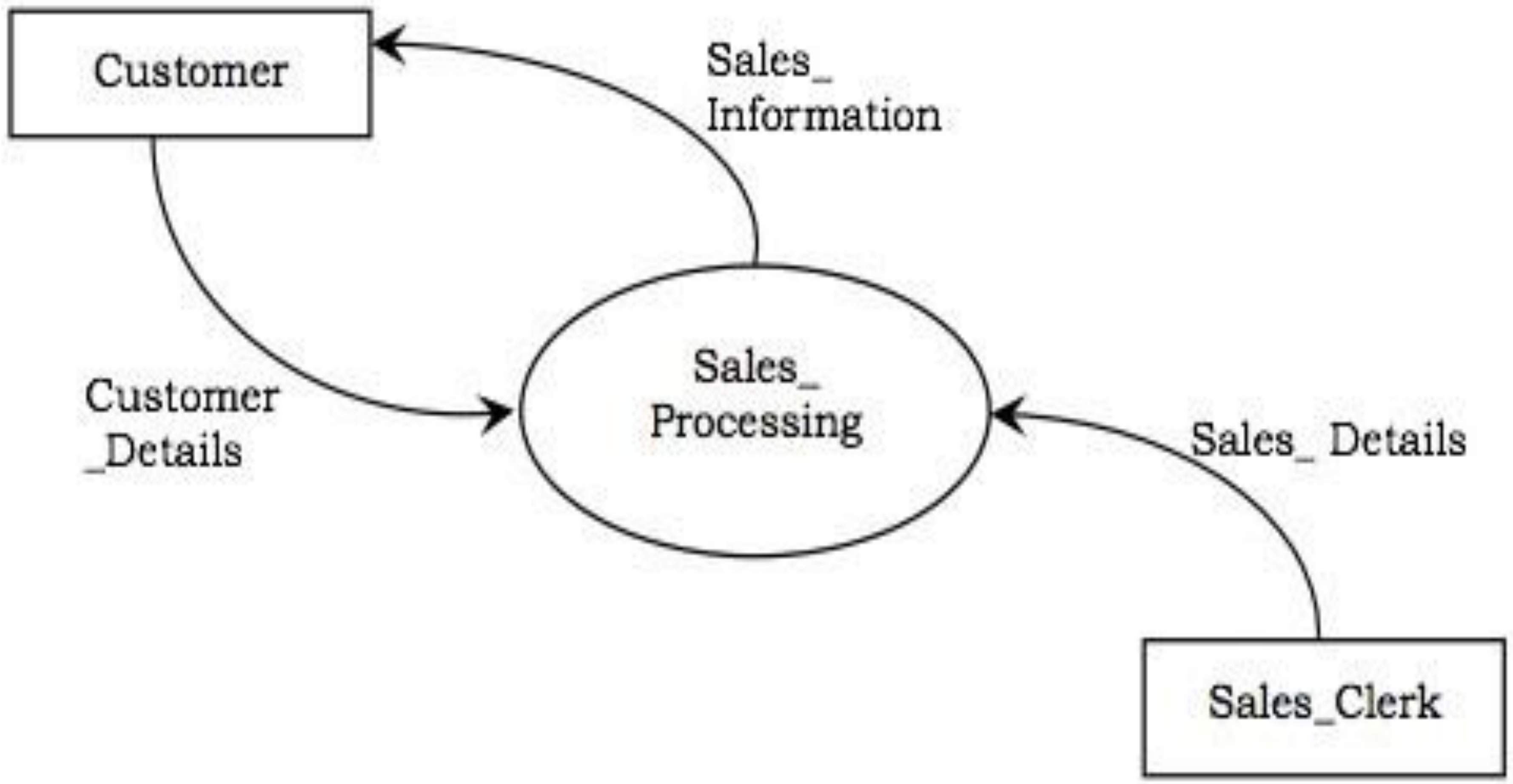
Data Flow Diagrams

- Functional Modelling is represented through a hierarchy of DFDs.
- The DFD is a graphical representation of a system that shows the inputs to the system, the processing upon the inputs, the outputs of the system as well as the internal data stores.
- DFDs illustrate the series of transformations or computations performed on the objects or the system, and the external controls and objects that affect the transformation.

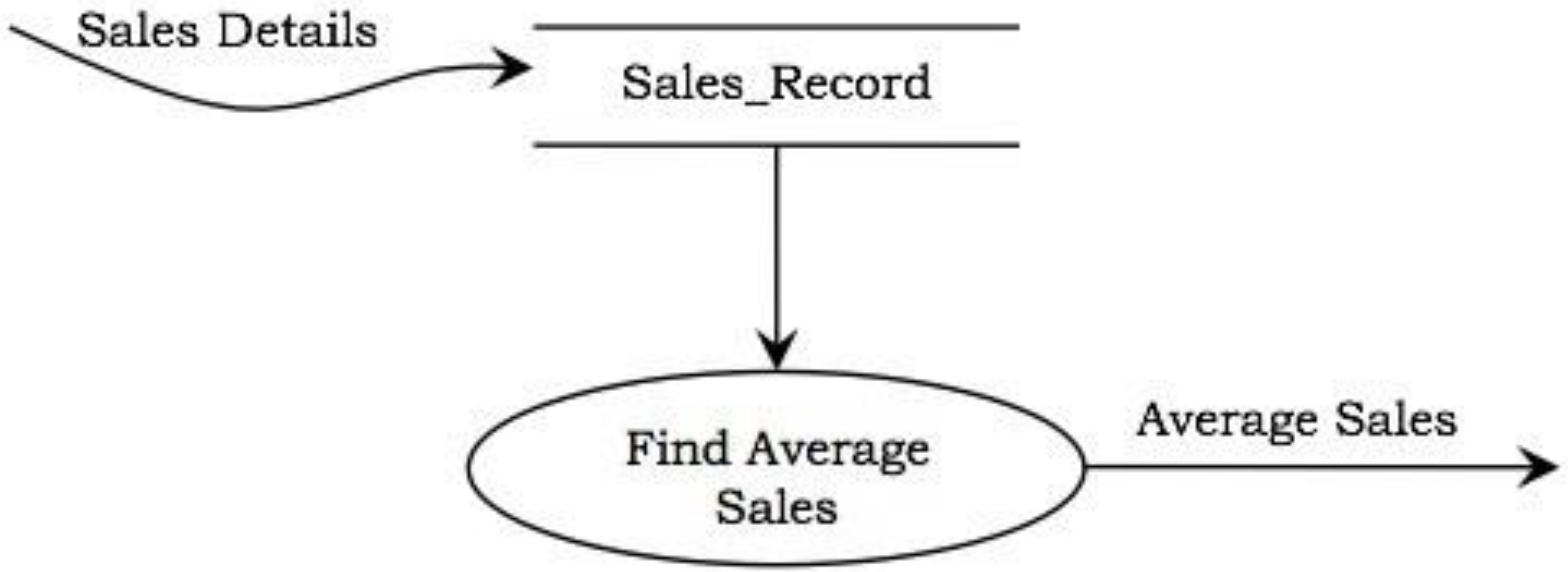
Features of a DFD

- **Processes** – Processes are the computational activities that transform data values. A whole system can be visualized as a high-level process. A process may be further divided into smaller components. The lowest-level process may be a simple function.
- **Representation in DFD** − A process is represented as an ellipse with its name written inside it and contains a fixed number of input and output data values.
- **Example** − The following figure shows a process Compute_HCF_LCM that accepts two integers as inputs and outputs their HCF (highest common factor) and LCM (least common multiple).

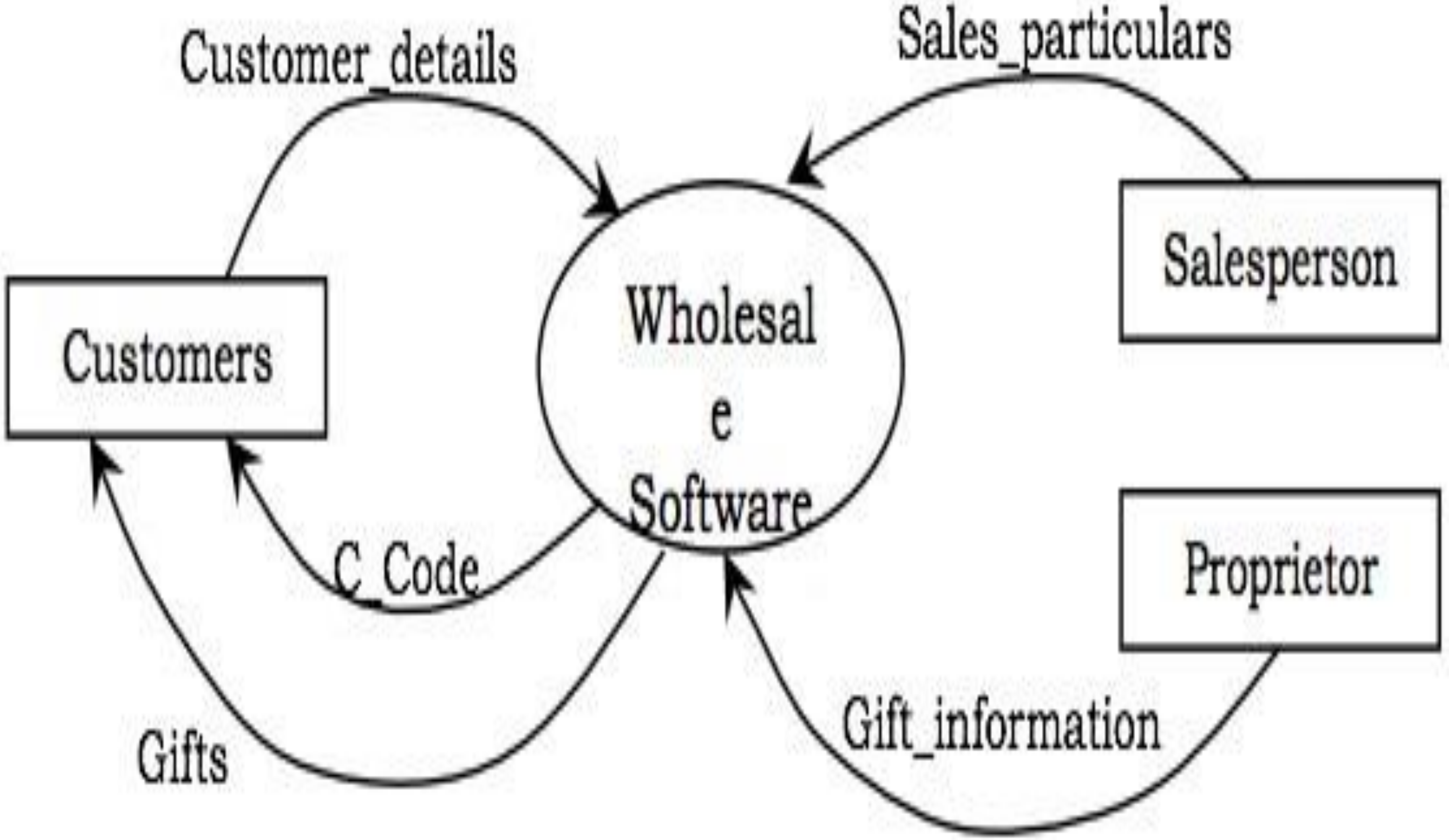
- **Data Flows** – Data flow represents the flow of data between two processes. It could be between an actor and a process, or between a data store and a process. A data flow denotes the value of a data item at some point of the computation. This value is not changed by the data flow.
- **Actors** – Actors are the active objects that interact with the system by either producing data and inputting them to the system, or consuming data produced by the system. In other words, actors serve as the sources and the sinks of data.


Customer
Sales_ Information
Customer _Details
Sales_ Processing
Sales_ Details
Sales_Clerk

- **Data Stores** – Data stores are the passive objects that act as a repository of data. Unlike actors, they cannot perform any operations. They are used to store data and retrieve the stored data. They represent a data structure, a disk file, or a table in a database.


Sales Details
Sales_Record
Find Average Sales
Average Sales

- **Example** – Let us consider a software system, Wholesaler Software, that automates the transactions of a wholesale shop. The shop sells in bulks and has a clientele comprising of merchants and retail shop owners. Each customer is asked to register with his/her particulars and is given a unique customer code, C_Code. Once a sale is done, the shop registers its details and sends the goods for dispatch. Each year, the shop distributes Christmas gifts to its customers, which comprise of a silver coin or a gold coin depending upon the total sales and the decision of the proprietor.


Customer_details
Sales_particulars
Salesperson
Customers
Wholesal
e
Software
C_Code
Proprietor
Gifts
Gift_information

2.8 Behavioural modelling

- The *behavioral model* indicates how software will respond to external events. To create the model, you should perform the following steps:
- **1.** Evaluate all use cases to fully understand the sequence of interaction within the system.
- **2.** Identify events that drive the interaction sequence and understand how these events relate to specific objects.
- **3.** Create a sequence for each use case.
- **4.** Build a state diagram for the system.
- **5.** Review the behavioral model to verify accuracy and consistency.

How do we Represent them?

- (Design) Sequence Diagrams
- **Communication Diagrams** *or* **collaboration diagram**
- State Diagrams *or* **state machine diagram** *or* **state chart**

1. Identifying Events with the Use Case

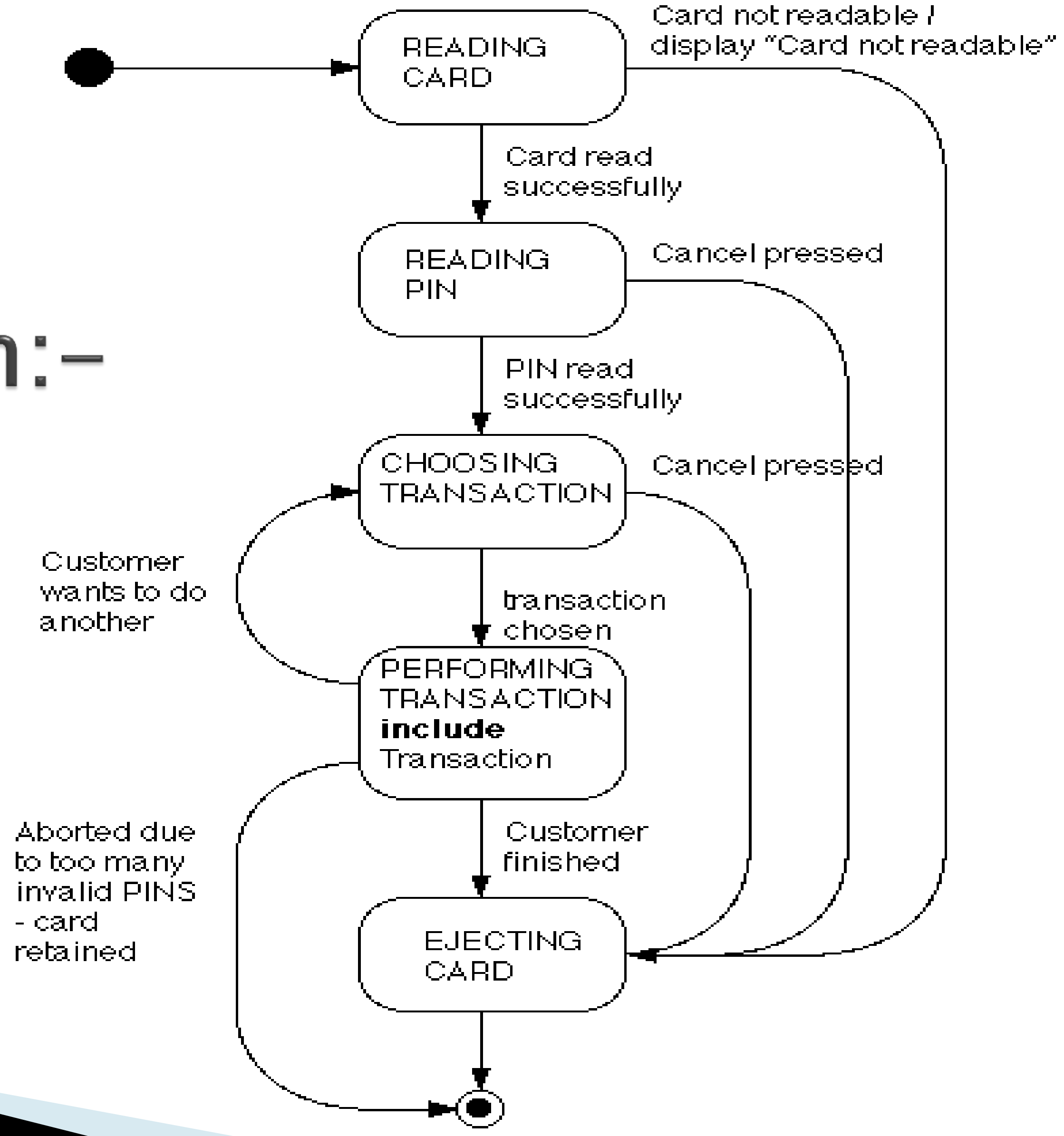
2. State Representations

- In the context of behavioral modeling, two different characterizations of states must
- be considered: (1) **the state of each class as the system performs its function** and
- (2) **the state of the system as observed from the outside as the system performs its function**

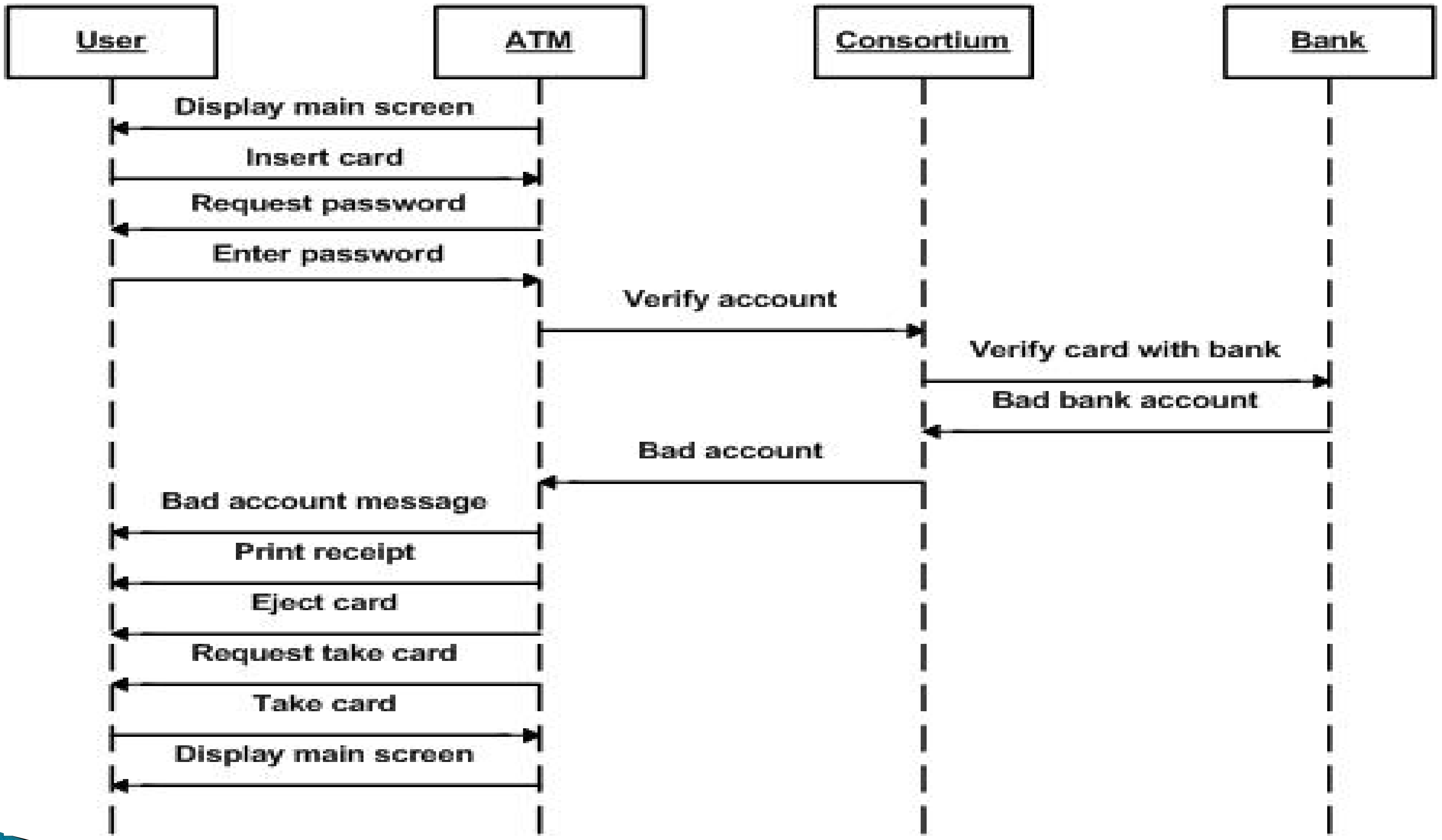
- The state of a class takes on both passive and active characteristics .
- A *passive state* is simply the current status of all of an object's attributes.(eg : a Player is Playing Video Game . would include the current position and orientation attributes of **Player** as well as other features of that are relevant to the game).
- The *active state* of an object indicates the current status of the object as it undergoes a continuing transformation or processing.(eg :- **Player** might have the following active states: *moving, at rest, injured, being smoked;trapped, lost,* and so onwards. An event (sometimes called a *trigger*) must occur to force an object to make a transition from one active state to another.

State Daigram:-

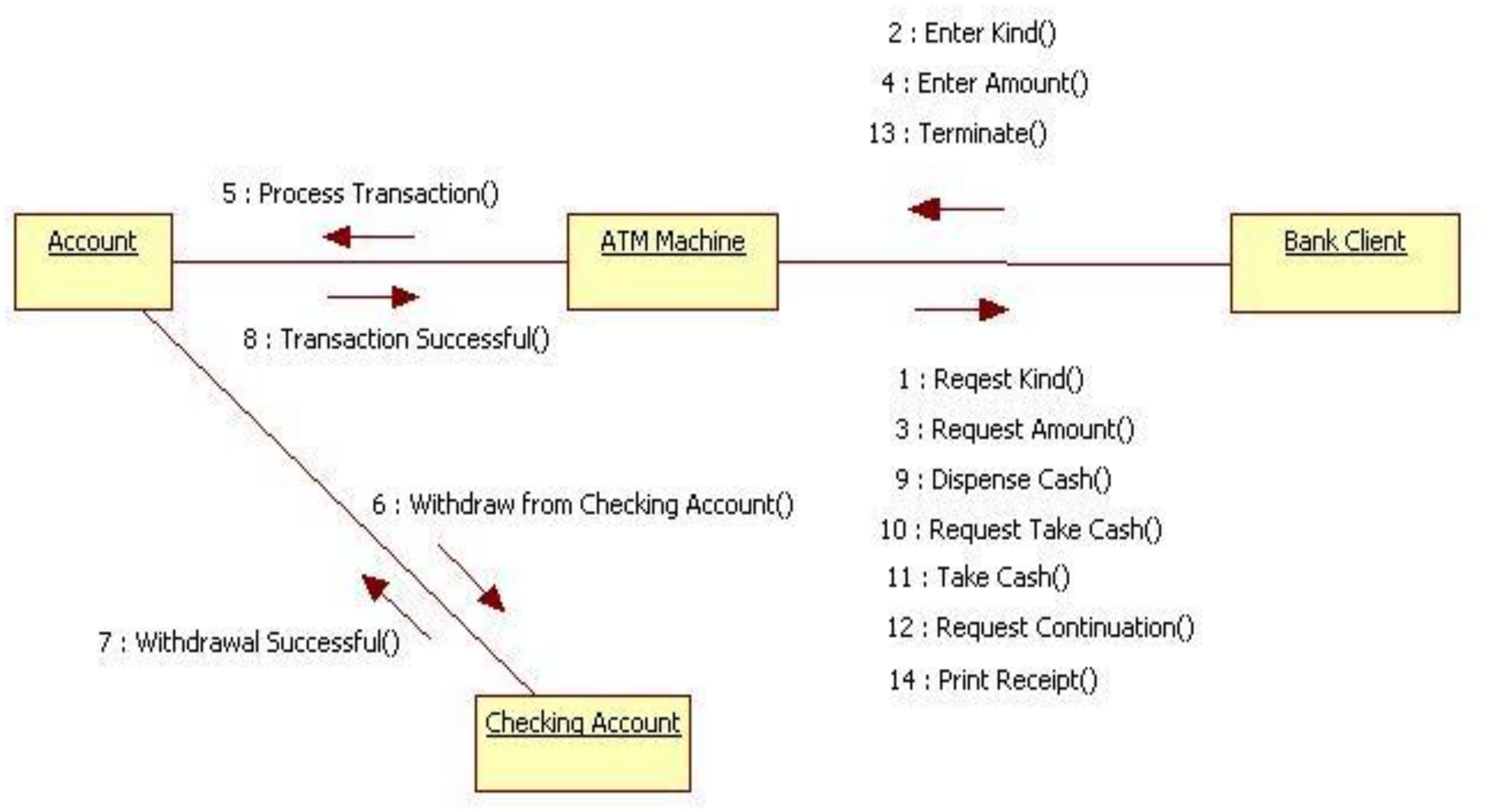
State-Chart for One Session


READING CARD
Card not readable / display "Card not readable"
Card read successfully
READING PIN
Cancel pressed
PIN read successfully
CHOOSING TRANSACTION
Cancel pressed
Customer wants to do another
transaction chosen
PERFORMING TRANSACTION include Transaction
Aborted due to too many invalid PINS - card retained
Customer finished
EJECTING CARD

Sequence Daigram:-



Communication Diagrams *or* collaboration diagram



2 : Enter Kind()
4 : Enter Amount()
13 : Terminate()
5 : Process Transaction()
Account
ATM Machine
Bank Client
8 : Transaction Successful()
1 : Reqest Kind()
3 : Request Amount()
9 : Dispense Cash()
10 : Request Take Cash()
11 : Take Cash()
12 : Request Continuation()
14 : Print Receipt()
6 : Withdraw from Checking Account()
7 : Withdrawal Successful()
Checking Account