Software Engineering

[LAB MANUAL]

2011

INDIRA GANDHI INSTITUTE OF TECHNOLOGY
(IGIT)

Version No.	**Comments**	**Modification Date**	**Modification Done By**
1.0	Created Lab Manual – added list of experiments, case studies and conceptual references	8th Feb. 2011	**Ankita Singh,** Assistant Professor, Information Technology Dept., IGIT.

INDEX

LAB WORK-I

Title: Design of Flow Chart
Objective To perform the Requirement analysis of the specified problem and draw a flow chart

----

References: An Integrated Approach To Software Engineering Pankaj Jalote
Software Engineering K.K Aggarwal & Yogesh Singh
Software Engineering Roger Pressman McGraw Hill
Visual Modeling with Rational Rose 2000 and UML Terry Quatrani.

Theory:

Flowcharts are the ideal diagrams for visually representing business processes. For example, if you need to show the flow of a custom-order process through various departments within your organization, you can use a flowchart. This paper provides a visual representation of basic flowchart symbols and their proposed use in communicating the structure of a well-developed web site, as well as their correlation in developing on-line instructional projects. A typical flowchart from older Computer Science textbooks may have the following kinds of symbols:

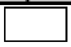
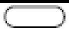
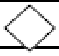
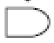
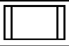
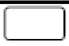
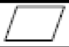
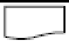
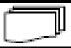
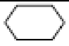
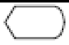
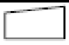
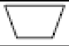
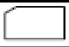
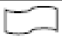
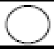
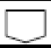
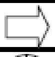
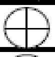
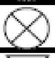
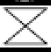
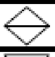
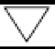
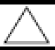
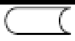
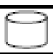
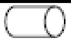
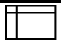
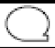
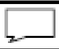
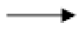
- **Start** and **end** symbols, represented as lozenges, ovals or rounded rectangles, usually containing the word "Start" or "End", or another phrase signaling the start or end of a process, such as "submit enquiry" or "receive product".
- **Arrows** showing what's called "flow of control" in computer science. An arrow coming from one symbol and ending at another symbol signifies flow passes to the symbol the arrow points to.
- **Processing steps**, represented as rectangles. Examples: "Add 1 to X"; "replace identified part"; "save changes" or similar.
- **Input/ Output** represented as a parallelogram. Examples: Get X from the user; display X.
- **Conditional** (or **decision**) represented as a diamond (rhombus). These typically contain a Yes/No question or True/False test. This symbol is unique in that it has two arrows coming out of it, usually from the bottom point and right point, one corresponding to Yes or True, and one corresponding to No or False. The arrows should always be labeled. More than two arrows can be used, but this is normally a clear indicator that a complex decision is being taken, in which case it may need to be broken-down further, or replaced with the "pre-defined process" symbol.

A number of other symbols that have less universal currency, such as:

- A **Document** represented as a rectangle with a wavy base;
- A **Manual input** represented by a rectangle, with the top irregularly sloping up from left to right. An example would be to signify data-entry from a form;

- A **Manual operation** represented by a trapezoid with the longest parallel side upmost, to represent an operation or adjustment to process that can only be made manually.
- A **Data File** represented by a cylinder

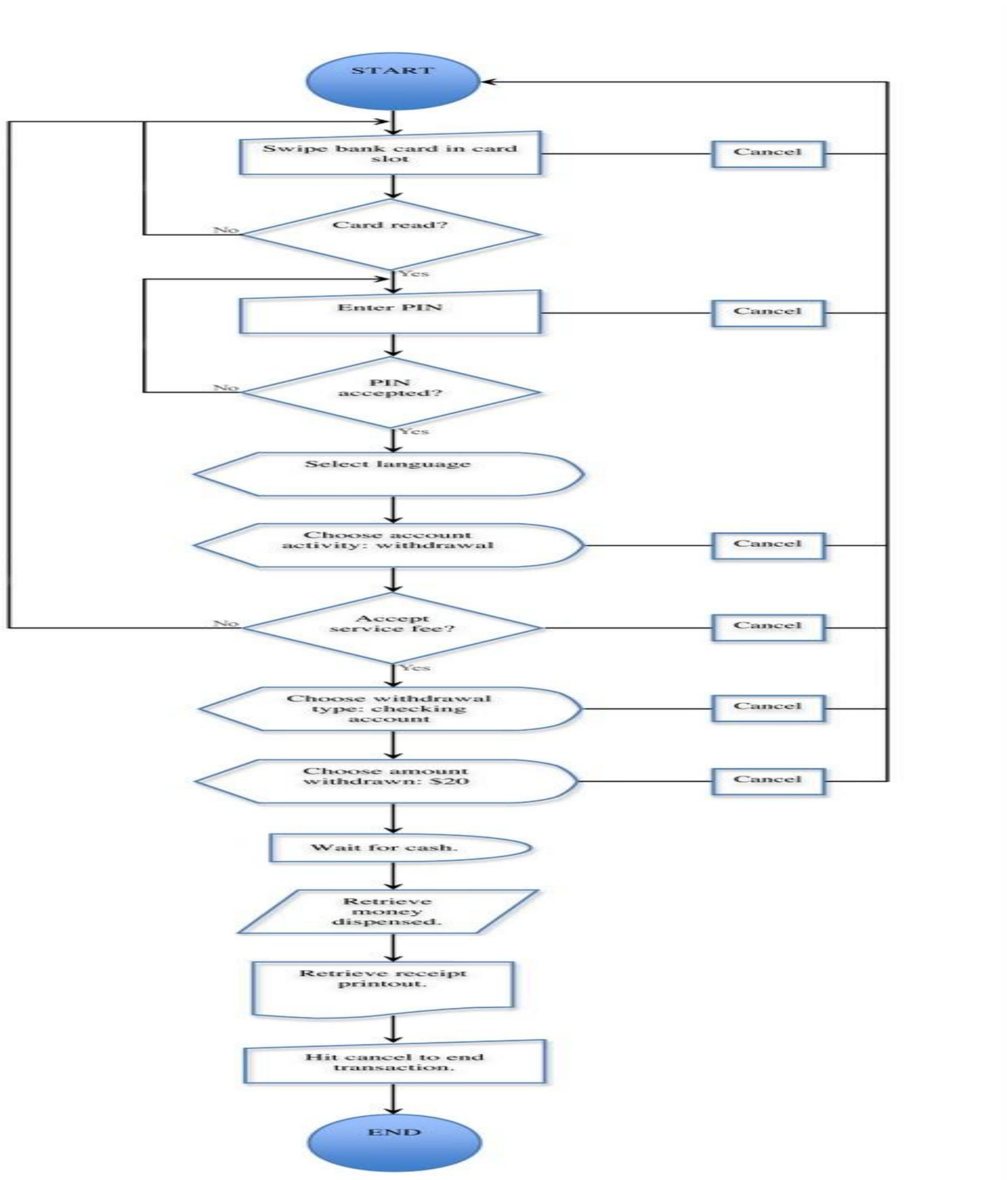
Flowchart Symbol Cheat Sheet

Flowchart Symbol	Name (Alternates)	Description
	Process	An operation or action step.
	Terminator	A start or stop point in a process.
	Decision	A question or branch in the process.
	Delay	A waiting period.
	Predefined Process	A formally defined sub-process.
	Alternate Process	An alternate to the normal process step.
	Data (I/O)	Indicates data inputs and outputs to and from a process.
	Document	A document or report.
	Multi-Document	Same as Document, except, well, multiple documents.
	Preparation	A preparation or set-up process step.
	Display	A machine display.
	Manual Input	Manually input into a system.
	Manual Operation	A process step that isn't automated.
	Card	A old computer punch card.
	Punched Tape	An old computer punched tape input.
	Connector	A jump from one point to another.
	Off-Page Connector	Continuation onto another page.
	Transfer	Transfer of materials.
	Or	Logical OR
	Summing Junction	Logical AND
	Collate	Organizing data into a standard format or arrangement.
	Sort	Sorting of data into some pre-defined order.
	Merge (Storage)	Merge multiple processes into one. Also used to show raw material storage.
	Extract (Measurement) (Finished Goods)	Extract (split processes) or more commonly - a measurement or finished goods.
	Stored Data	A general data storage flowchart symbol.
	Magnetic Disk (Database)	A database.
	Direct Access Storage	Storage on a hard drive.
	Internal Storage	Data stored in memory.
	Sequential Access Storage (Magnetic Tape)	An old reel of tape.
	Callout	One of many callout symbols used to add comments to a flowchart
	Flow Line	Indicates the direction of flow for materials and/or information

Procedure:

1) Identify various processes, input/output and decision making statement of the system and analyse it.
2) Use them to draw flow chart.

Sample Output:


START
Swipe bank card in card slot
Cancel
Card read?
No
Yes
Enter PIN
Cancel
PIN accepted?
No
Yes
Select language
Choose account activity: withdrawal
Cancel
Accept service fee?
No
Yes
Cancel
Choose withdrawal type: checking account
Cancel
Choose amount withdrawn: $20
Cancel
Wait for cash.
Retrieve money dispensed.
Retrieve receipt printout.
Hit cancel to end transaction.
END

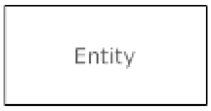
LAB WORK-II

Title: Draw E-R diagram for the specified system
Objective: To draw the ER-Diagram with generalization, specialization and aggregation.

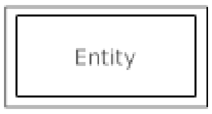
References: An Integrated Approach To Software Engineering Pankaj Jalote
Software Engineering K.K Aggarwal & Yogesh Singh
Software Engineering Roger Pressman McGraw Hill
Visual Modeling with Rational Rose 2000 and UML Terry Quatrani

Theory: An entity-relationship model (ERM) is an abstract and conceptual representation of data. Entity-relationship modeling is a database modeling method, used to produce a type of conceptual schema or semantic data model of a system, often a relational database, and its requirements in a top-down fashion. Diagrams created by this process are called entity-relationship diagrams, ER diagrams, or ERDs. The building blocks: entities, relationships, and attributes

An entity may be defined as a thing which is recognized as being capable of an independent existence and which can be uniquely identified. An entity may be a physical object such as a house or a car, an event such as a house sale or a car service, or a concept such as a customer transaction or order. Entities can be thought of as nouns. Examples: a computer, an employee, a song, a mathematical theorem. Entity sets are drawn as rectangles, relationship sets as diamonds. If an entity set participates in a relationship set, they are connected with a line.

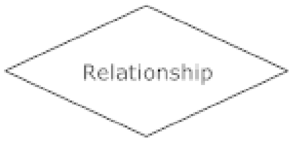


Weak Entity:
A weak entity is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone.



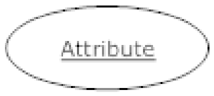
A relationship captures how two or more entities are related to one another. Relationships can be thought of as verbs, linking two or more nouns. Examples: a *performs* relationship between an artist and a song, a *proved* relationship between a mathematician and a theorem.

Relationships illustrate how two entities share information in the database structure.

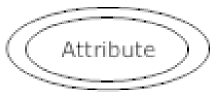


Attributes are drawn as ovals and are connected with a line to exactly one entity or relationship set

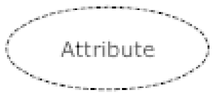
Key attribute
A key attribute is the unique, distinguishing characteristic of the entity. For example, an employee's social security number might be the employee's key attribute.



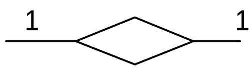
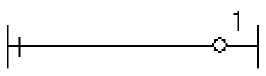
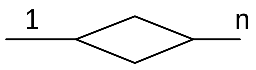
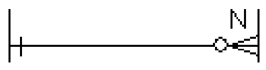
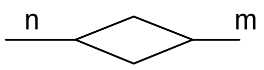
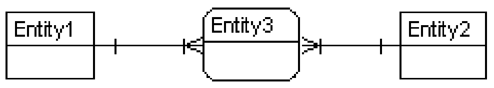
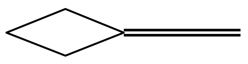
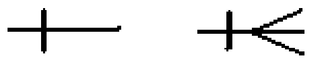
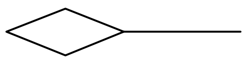
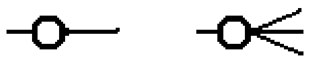
Multivalued attribute*:*
A multivalued attribute can have more than one value. For example, an employee entity can have multiple skill values.



Derived attribute:
A derived attribute is based on another attribute. For example, an employee's monthly salary is based on the employee's annual salary.

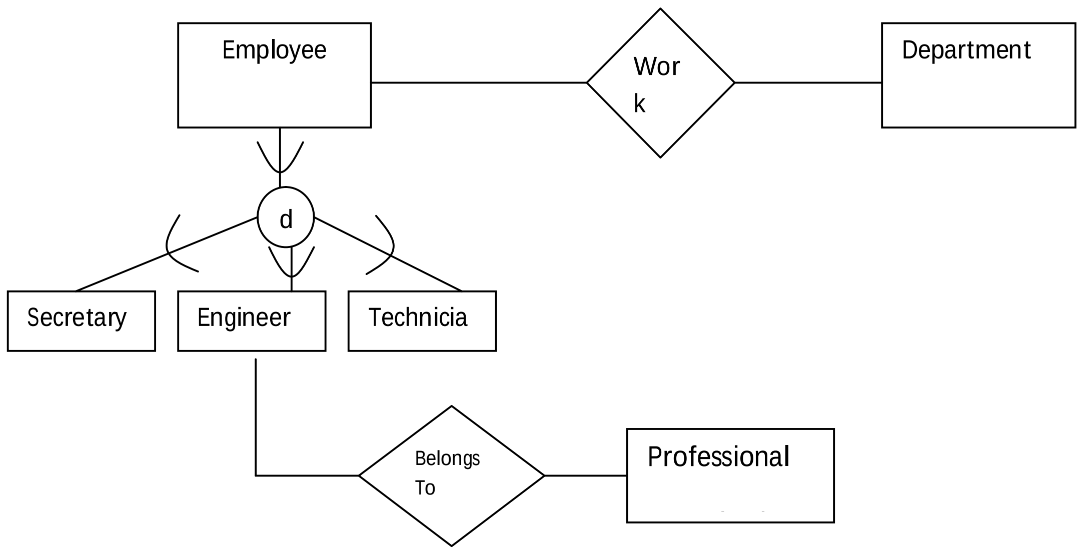


Cardinality:
Cardinality specifies how many instances of an entity relate to one instance of another entity.
Ordinality is also closely linked to cardinality. While cardinality specifies the occurences of a relationship, ordinality describes the relationship as either mandatory or optional. In other words, cardinality specifies the maximum number of relationships and ordinality specifies the absolute minimum number of relationships.

Cardinality		
One-to-one		
One-to-many		
Many-to-Many		
Participation		
Total (Mandatory)		
Partial (Optional)		

Specialization and Generalization:
Specialization is the procedure of defining a set of subclasses. The entity of which these subclasses are parts is called the superclass of the specialization.
Allows for

- Defining set of subclasses of entity type
- Create additional specific attributes for each sub class
- Create additional specific relationship types between each sub class and other entity types or other subclasses.



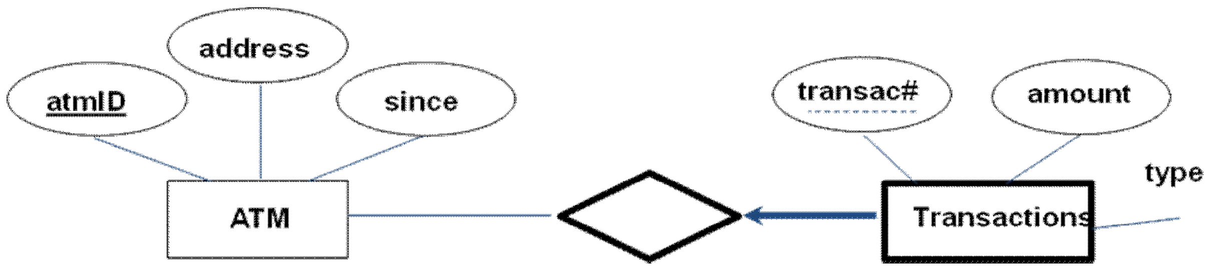
Generalization:
This can be thought of as the reverse procedure. This time we suppress the differences among several entity types and identify their common features. For example, the entity types Car and Truck share common attributes License, PlateNo, VehicleID and Price, therefore they can be generalized into the super class Vehicle.

Constraints on Specialization and Generalization

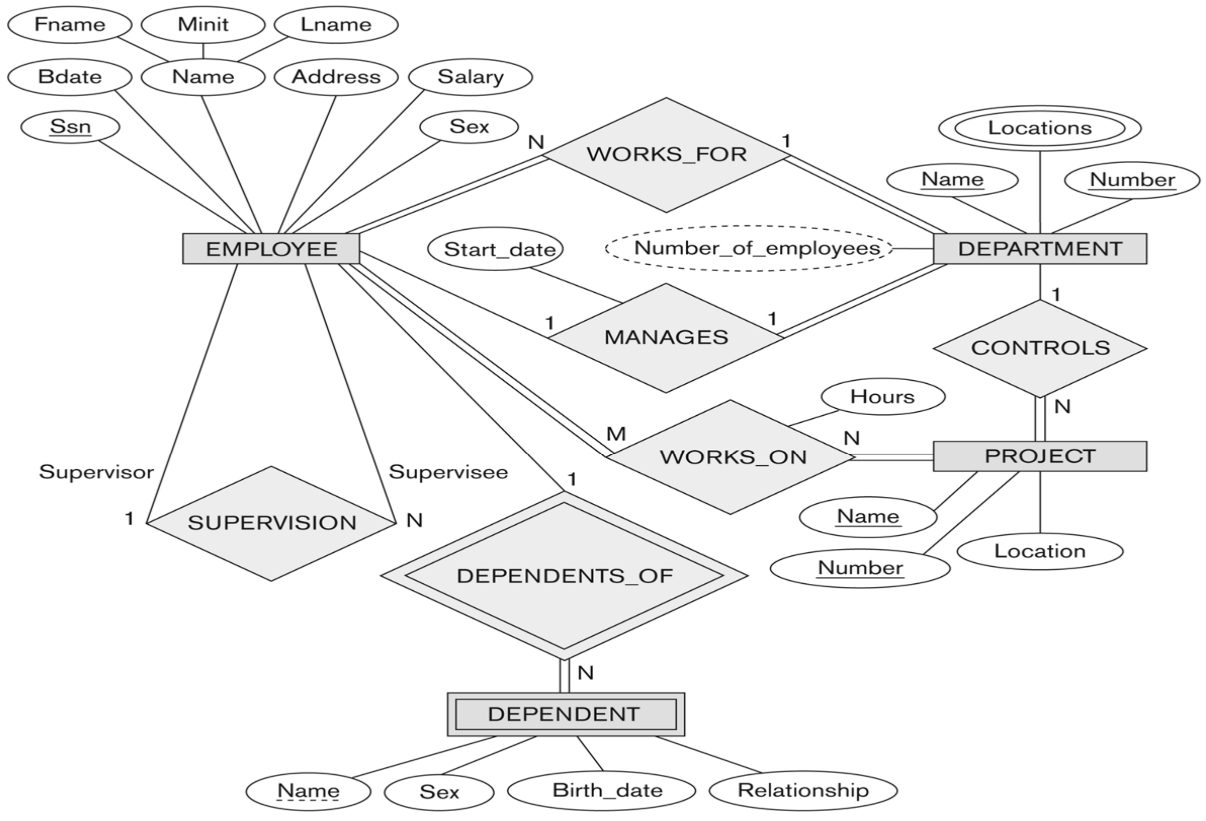
- Several specializations can be defined on an entity type.
- Entities may belong to subclasses in each of the specializations.
- The specialization may also consist of a single subclass, such as the manager specialization; in this case we don't use the circle notation.

Sample Output: A software training program is divided into training module, and each module is described by module name and approximate practice time .Each module sometimes has prerequisite modules. Model this situation of training programs and modules with an E-R diagram.

ER diagram for ATM:



Another Example (COMPANY DATABASE): ER diagram for Company Database:



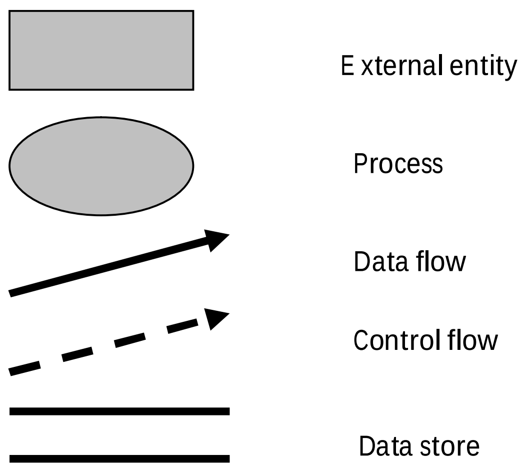
LAB WORK-III

Title: Design of the System modeling.
Objective: Understanding of System modeling: Functional modeling: DFD level 0 i.e. Context Diagram and perform it.

References: An Integrated Approach To Software Engineering Pankaj Jalote
Software Engineering K.K Aggarwal & Yogesh Singh
Software Engineering Roger Pressman McGraw Hill
Visual Modeling with Rational Rose 2000 and UML Terry Quatrani

Theory:
Data flow diagram:
Data flow diagrams are versatile diagramming tools. With only four symbols, data flow diagrams can represent both physical and logical information systems. The four symbols used in DFDs represent data flows, data stores, processes, and sources / sinks (or external entities).

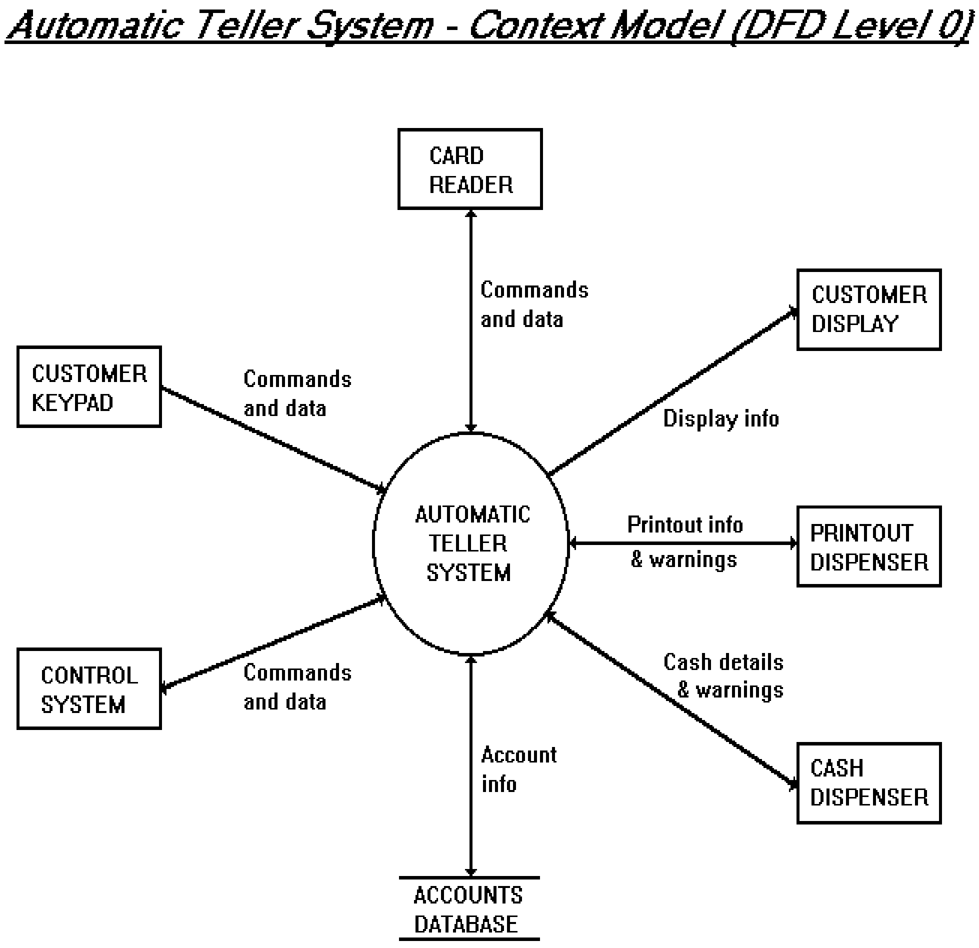
Symbols of DFD:
A data flow is depicted as an arrow.
A rectangle or square is used for sources / sinks and its name states what the external agent is, such as customer, teller, EPA office, or inventory control system.
The symbol for a process is a rectangle with rounded corners.
The symbol for a data store is a rectangle with the right vertical line missing. Its label includes the number of data store.



Procedure:

1) Identify various processes, data store, input, output etc. of the system and analyse it.
2) Use processes at various levels to draw the DFDs.
3) Identify various modules, input, output etc. of the system and ask students to analyze.

Sample Output:



LAB WORK-IV

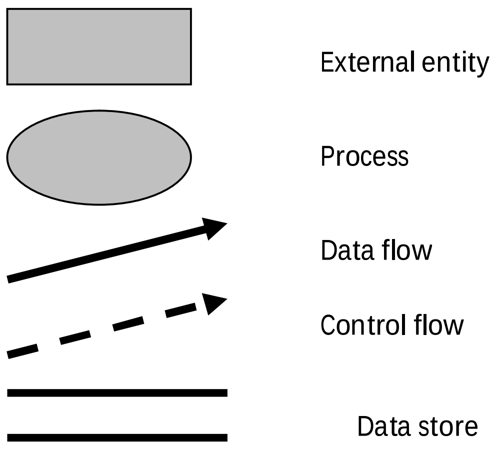
Title: Design of the System modeling.
Objective: Understanding of System modeling: Functional modeling: DFD level 1 and perform it.

References: An Integrated Approach To Software Engineering Pankaj Jalote
Software Engineering K.K Aggarwal & Yogesh Singh
Software Engineering Roger Pressman McGraw Hill
Visual Modeling with Rational Rose 2000 and UML Terry Quatrani

Tools/Apparatus: Rational Rose

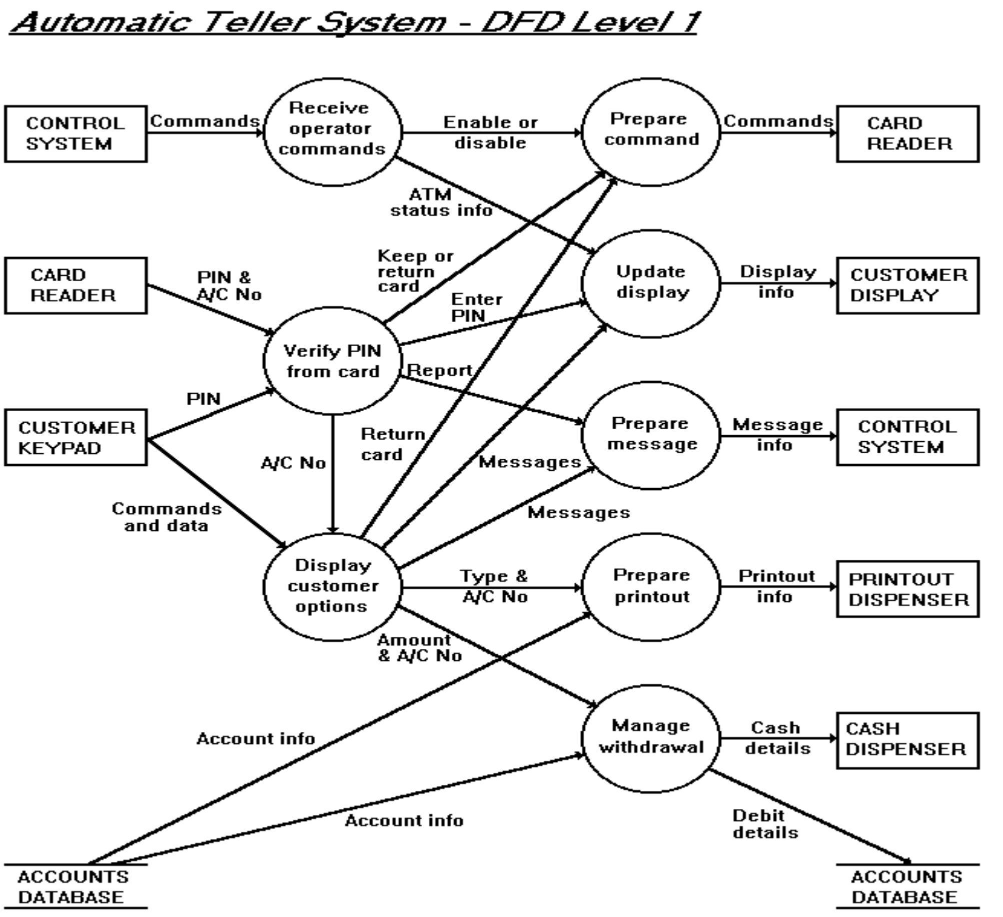
Theory:
Data flow diagram
Data flow diagrams are versatile diagramming tools. With only four symbols, data flow diagrams can represent both physical and logical information systems. The four symbols used in DFDs represent data flows, data stores, processes, and sources / sinks (or external entities).
Symbols of DFD

- A data flow is depicted as an arrow.
- A rectangle or square is used for sources / sinks and its name states what the external agent is, such as customer, teller, EPA office, or inventory control system.
- The symbol for a process is a rectangle with rounded corners.
- The symbol for a data store is a rectangle with the right vertical line missing. Its label includes the number of data store.



Procedure:
1) Identify various processes, data store, input, output etc. of the system and analyse it.
2) Use processes at various levels to draw the DFDs.
3) Identify various modules, input, output etc. of the system and ask students to analyse.

Sample Output:



This diagram shows data entering and leaving the system. Input data is received from the hardware elements on the left. Various types of data are processed by different parts of the software system. Output data is sent to the elements of hardware on the right.

LAB WORK-V

Title: To perform the user's view analysis

Objective: Understanding different actors and use cases in detail of the specified problem statement and draw it using Rational Rose

---

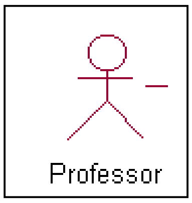
References: An Integrated Approach To Software Engineering Pankaj Jalote
Software Engineering K.K Aggarwal & Yogesh Singh
Software Engineering Roger Pressman McGraw Hill
Visual Modeling with Rational Rose 2000 and UML Terry Quatrani

Tools/Apparatus: Rational Rose

Theory:

Actors

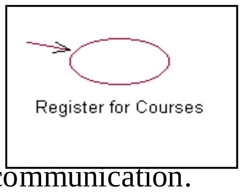
- Are NOT part of the system – they represent anyone or anything that must interact with the system.
- Only input information to the system.
- Only receive information from the system.
- Both input to and receive information from the system.
- Represented in UML as a stickman.



Use Case

- A sequence of transactions performed by a system that yields a measurable result of values for a particular actor
- A use case typically represents a major piece of functionality that is complete from beginning to end. A use case must deliver something of value to an actor

Use Case Relationships



- Between actor and use case.
- Association / Communication.
- Arrow can be in either or both directions; arrow indicates who initiates communication.
- Between use cases (generalization):

Uses: Where multiple use cases share pieces of same functionality.

Extends

- Optional behavior.
- Behavior only runs under certain conditions (such as alarm).
- Several different flows run based on the user's selection.

Procedure:

1) Identify various processes, use-cases, actors etc. of the system and analyse it.
2) Use processes at various levels .

Sample Output:

Actors (based on the nouns):

The ATM is the system under design so we scan the requirements to identify entities that interact with the ATM.

Customer	Customer needs to perform banking transactions using the ATM machine.
Bank	The bank serving this ATM.

Use-cases (based on the verbs):

Identify the Use Cases

Now we go through the requirements to identify the use cases that will give us a good coverage of the requirements:

• Customer withdraws cash
• Customer deposits cash
• Customer repeatedly enters invalid PIN

The final use case entity organization is shown below:

The Actors are further classified as Primary and Secondary Actors. The System is split into Input, Output and Controller.

Actors		**System**								
Primary Actors	Secondary Actors	Input			Output		Controller			
Customer	Bank	Card Reader	Key pad	Envelope Dispenser	Display	Printer	ATM	Session	Transaction	Logger

LAB WORK-VI

Title: To perform the user's view analysis
Objective: To draw Use case diagram using Rational Rose

References: An Integrated Approach To Software Engineering Pankaj Jalote
Software Engineering K.K Aggarwal & Yogesh Singh
Software Engineering Roger Pressman McGraw Hill
Visual Modeling with Rational Rose 2000 and UML Terry Quatrani

Tools/Apparatus: Rational Rose

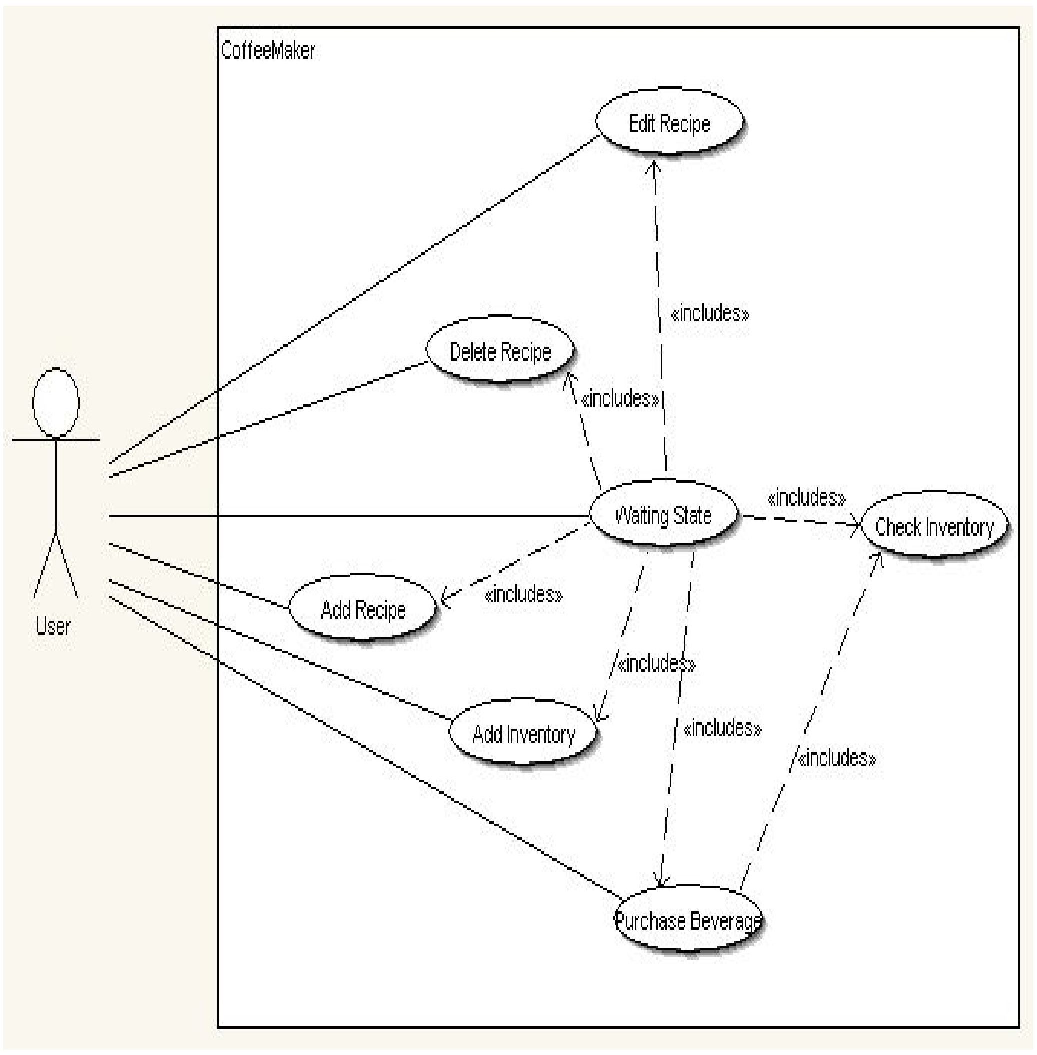
Theory: The use-case diagram can provide the user's view for designing of the software product. And it can also be tested by matching up the requirements with the use-cases.

When to Use: Use Cases Diagrams
Use cases are used in almost every project. The are helpful in exposing requirements and planning the project. During the initial stage of a project most use cases should be defined, but as the project continues more might become visible.

Procedure:

1) Identify various processes, use-cases, actors etc. of the system and analyse it.
2) Use processes at various levels and draw use case diagram.

Sample Output:



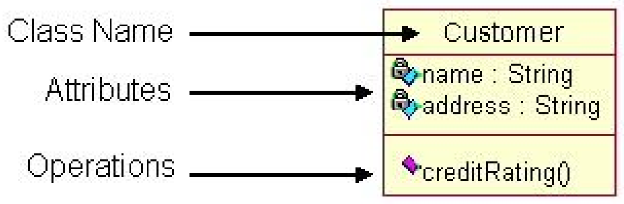
LAB WORK-VII

Title: To perform the structural view diagram
Objective: To perform Class diagram

References: An Integrated Approach To Software Engineering Pankaj Jalote
Software Engineering K.K Aggarwal & Yogesh Singh
Software Engineering Roger Pressman McGraw Hill
Visual Modeling with Rational Rose 2000 and UML Terry Quatrani

Tools/Apparatus: Rational Rose.

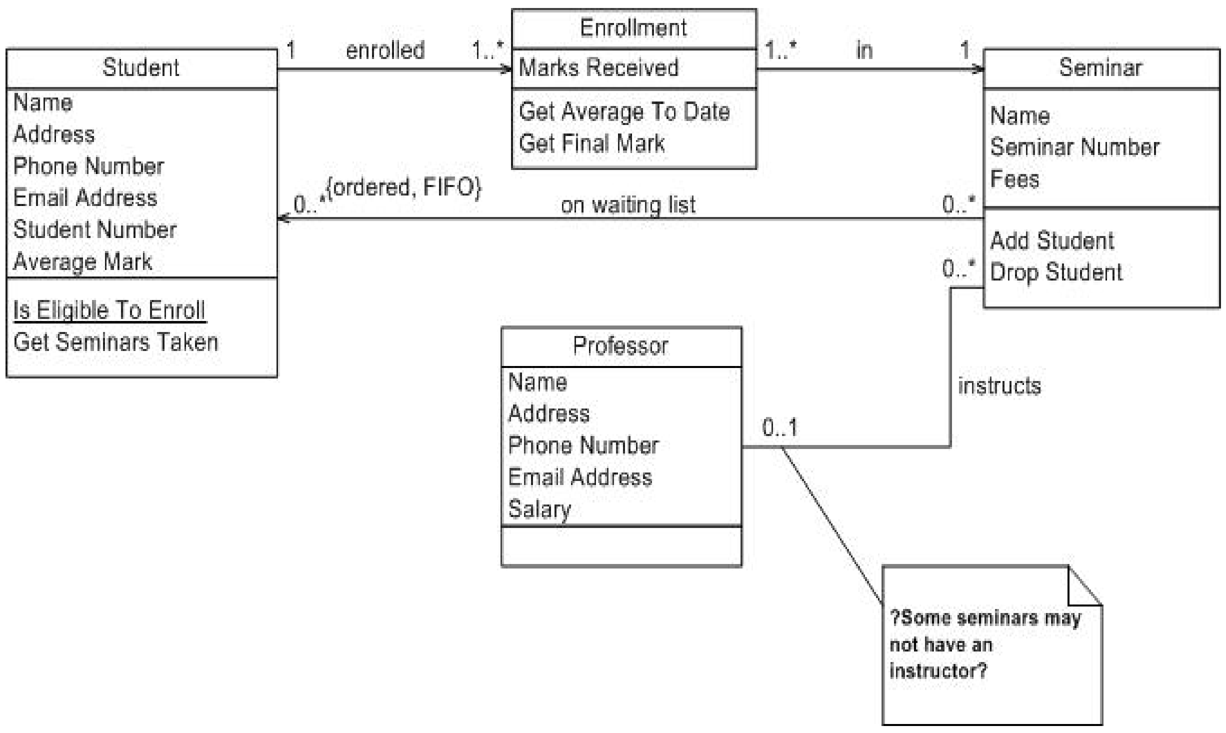
Theory: Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects.[2] Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation.[1] These perspectives become evident as the diagram is created and help solidify the design. This example is only meant as an introduction to the UML and class diagrams. If you would like to learn more see the Resources page for more detailed resources on UML.Classes are composed of three things: a name, attributes, and operations. Below is an example of a class.



Procedure:

1) Identify various elements such as classes, member variables, member functions etc.of the class diagram
2) Draw the class diagram as per the norms.

Sample Output:



LAB WORK-VIII

Title: To perform the behavioral view diagram

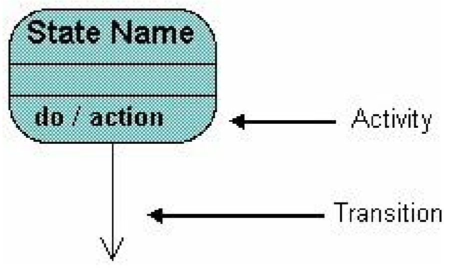
Objective: To draw State-chart diagram, Activity diagram of specified problem statement using Rational Rose

----

References: An Integrated Approach To Software Engineering Pankaj Jalote
Software Engineering K.K Aggarwal & Yogesh Singh
Software Engineering Roger Pressman McGraw Hill
Visual Modeling with Rational Rose 2000 and UML Terry Quatrani

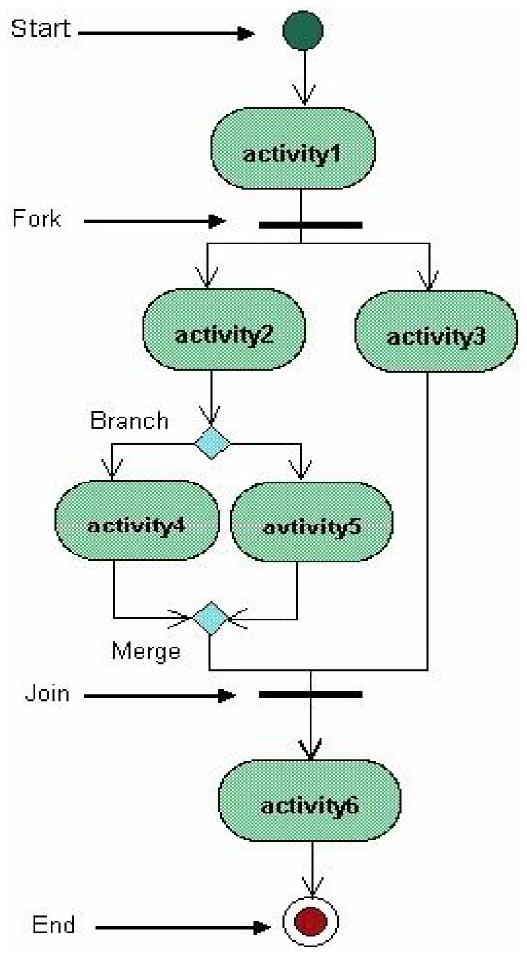
Tools/Apparatus: Rational Rose

Theory: _State diagrams are used to describe the behavior of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system. **Statechart** diagrams represent the behavior of entities capable of dynamic behavior by specifying its response to the receipt of event instances. Typically, it is used for _describing the behavior of classes, but statecharts may also describe the behavior of _other model entities such as us e-cases, actors, subsystems, operations, or methods. Use state diagrams to demonstrate the behavior of an object through many use cases of the system. Only use state diagrams for classes where it is necessary to understand the behavior of the object through the entire system. Not all classes will require a state diagram and state diagrams are not useful for describing the collaboration of all objects in a use case.



Activity diagrams describe the workflow behavior of a system. Activity diagrams are similar to state diagrams because activities are the state of doing something. The diagrams describe the state of activities by showing the sequence of activities performed. Activity diagrams can show activities that are conditional or parallel. Activity diagrams show the flow of activities through the system. Diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities. A fork is used when multiple activities are occurring at the same time. The diagram below shows a fork after activity1. This indicates that both activity2 and activity3 are occurring at the same time. After activity2 there is a branch. The branch describes what activities will take place based on a set of conditions. All branches at some point are followed by a merge to indicate the end of the conditional behavior started by that branch.

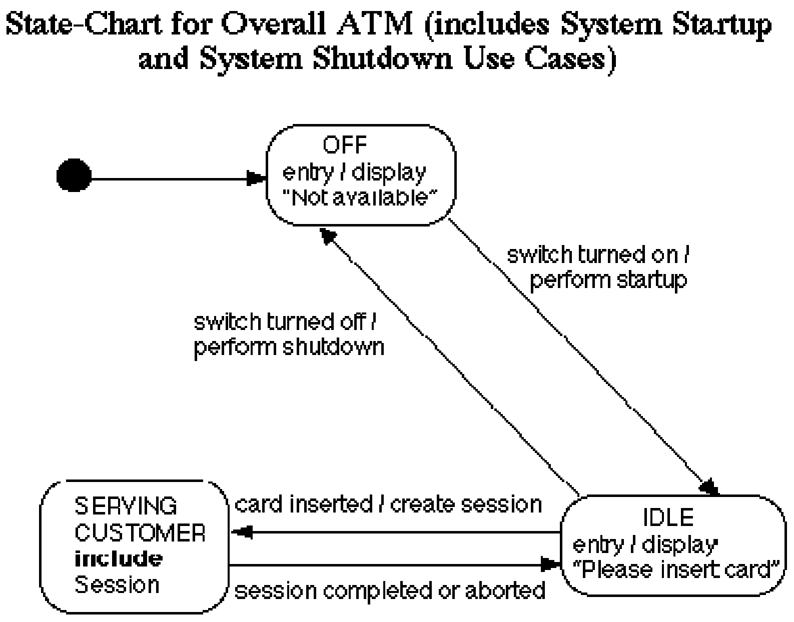
After the merge all of the parallel activities must be combined by a join before transitioning into the final activity state.



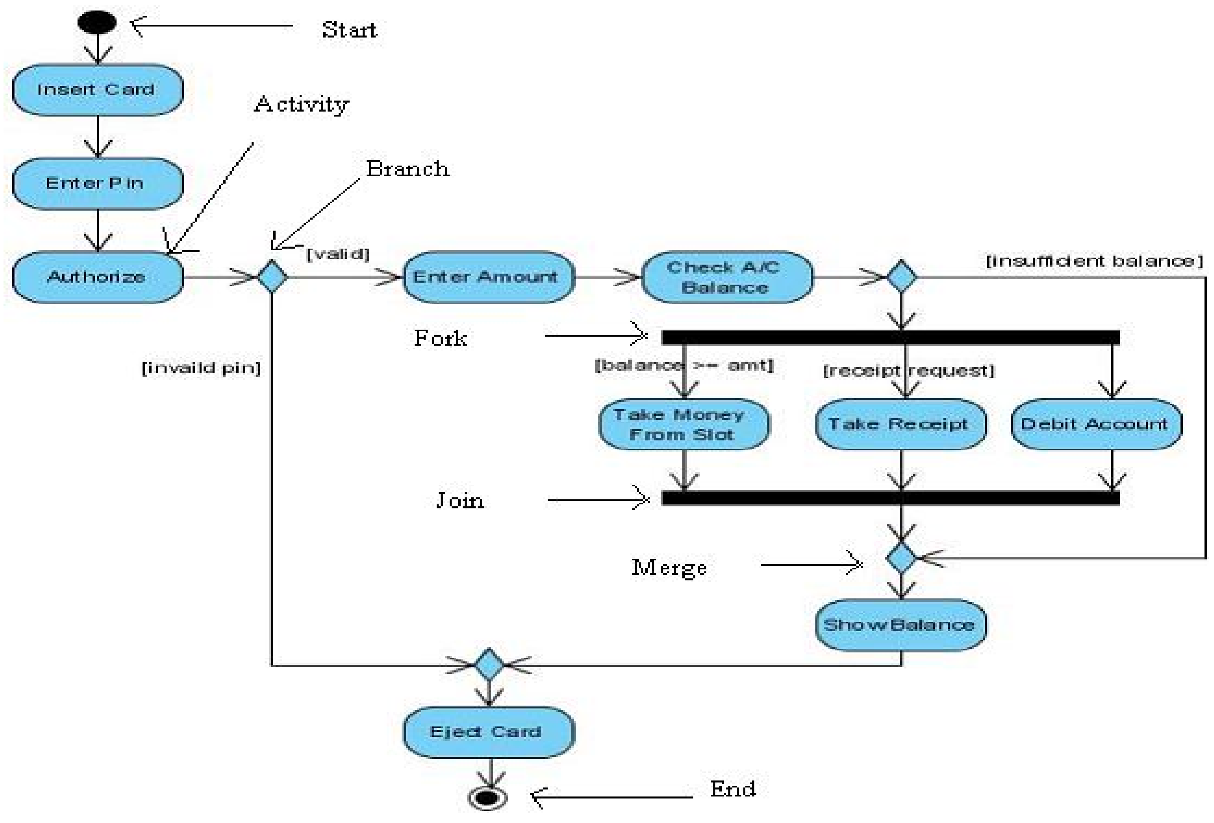
Procedure:

1) Identify various elements states and their different transition of the state-chart diagram
2) Draw the state-chart diagram as per the norms.
3) Identify various elements such as different activity their boundaries etc. of the activity diagram
4) Draw the activity diagram as per the norms.

Sample Output:
For ATM card Problem State-chart diagram would be:


State-Chart for Overall ATM (includes System Startup and System Shutdown Use Cases)
OFF
entry / display "Not available"
switch turned on / perform startup
switch turned off / perform shutdown
SERVING CUSTOMER include Session
card inserted / create session
session completed or aborted
IDLE
entry / display "Please insert card"

For ATM card Problem Activity Diagram would be:



LAB WORK-IX

Title: Design Test Cases.
Objective: To perform Boundary value analysis and Equivalence Class Testing**.**

References: An Integrated Approach To Software Engineering Pankaj Jalote
Software Engineering K.K Aggarwal & Yogesh Singh
Software Engineering Roger Pressman McGraw Hill
Visual Modeling with Rational Rose 2000 and UML Terry Quatrani

Theory:
Boundary value analysis is a software testing technique in which tests are designed to include representatives of boundary values. The expected input and output values should be extracted from the Component specification. The input and output values to the software component are then grouped into sets with identifiable boundaries

Procedure:
Boundary values Testing

1. First try to identify different inputs and output of the problem Statement.
2. Write down Boundary/edge value i.e. minimum, minimum-, nominal, maximum and maximum+ value of each input
3. Create Test cases by putting each value at the edge / boundary value and other at nominal value for Boundary values Testing

Equivalence class testing

1. First try to identify different inputs and output of the problem Statement
2. Create as many valid and invalid input classes of identified inputs
3. Create Test cases by taking value from each valid and invalid input.

Sample Output:

i) Boundary/edge value of each input
ii)

Value	**Input 1(A)**	**Input 2(B)**	**Input 3(C)**
Min	**1**	**1**	**1**
Min+	2	2	2
Nominal	100	100	100
Max-	199	199	199
Max	200	200	200

Test case Description	**Test case Id**	**Input**			**Expected Output**	**Actual Output**
		A	**B**	**C**		
Keep side A at edge value and other at nominal value	**1**	**1**	**100**	**100**	Isosceles triangle	Isosceles triangle
Keep side B at edge value and other at nominal	2	100	1	100	Isosceles triangle	Isosceles triangle
Keep side C at edge value and other at nominal	3	100	100	1	Isosceles triangle	Isosceles triangle

Equivalence class testing:

I1=<a, b, c: three valid equal sides>
I1=<a ,b, c: two valid equal sides>
I1=< a, b, c: two valid equal sides and one invalid side>
O1=<a, b, c: Equilateral Triangle >
O2=<a, b, c: Isosceles Triangle >
O3=<a, b, c: Scalene Triangle>
O4=<a, b, c: Invalid Input>

Test case Description	**Test case Id**	**Input**			**Expected Output**	**Actual Output**
		a	**b**	**c**		
Three valid equal sides	1	7	7	7	Equilateral triangle	Equilateral triangle
Two equal valid sides	2	5	6	5	Isosceles triangle	Isosceles triangle
Two equal valid sides; Third side invalid	3	2	2	220	Error msg: invalid Input data value	Error msg: invalid Input data value

LAB WORK-X

Title: Design Test Cases.
Objective: To draw Flow graph, DD paths, calculation of cyclomatic complexity and find out all the independent paths from the DD paths graph.

References: An Integrated Approach To Software Engineering Pankaj Jalote
Software Engineering K.K Aggarwal & Yogesh Singh
Software Engineering Roger Pressman McGraw Hill
Visual Modeling with Rational Rose 2000 and UML Terry Quatrani

THEORY : Brief description of white box testing and cyclomatic complexity.

PROCEDURE:

STEP 1: Analyze the source code
STEP 2: Draw Flow Graph
STEP 3: Draw the following DD Path Graph
STEP 4: Create decision table to find out the independent paths.
STEP 5: Calculate Cyclomatic Complexity V(G)
STEP 6: Identify independent paths and execute it.

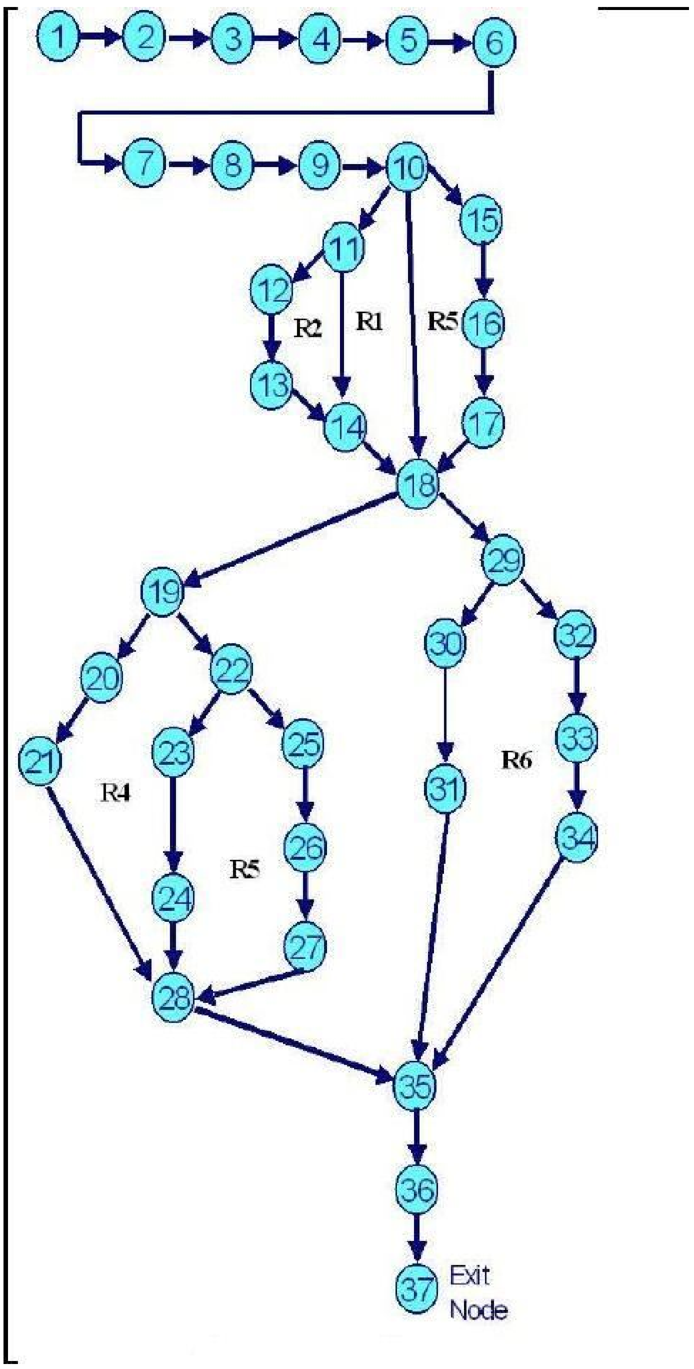
STEP 1: Analyze the source code

PROGRAM:

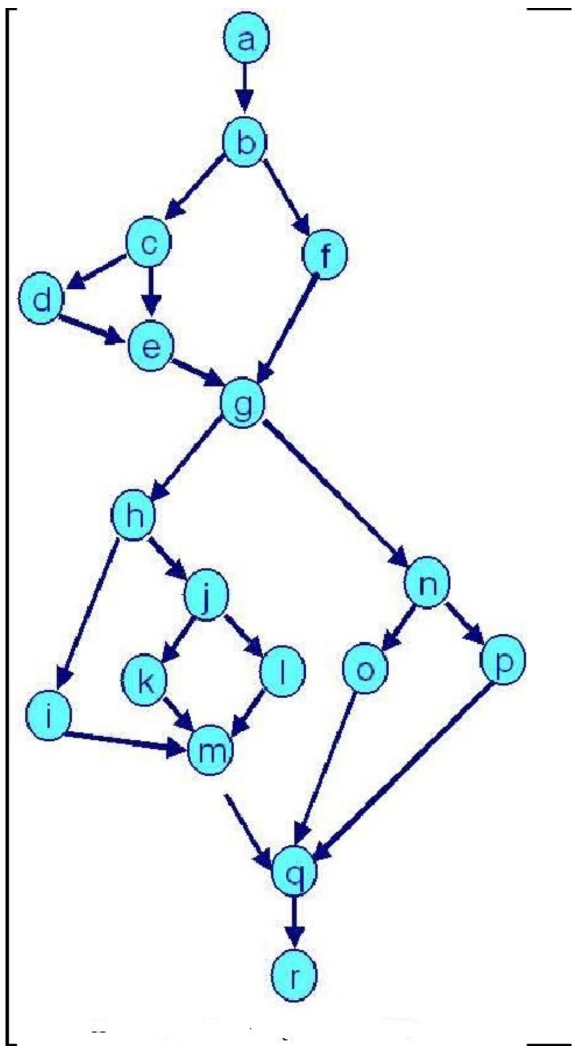
```
#include<stdio.h>
#include<conio.h>

(1). int main()
(2). {
(3). int a,b,c, boolean =0;
(4).printf("\n Enter the side A:");
(5). scanf("%d", &a);
(6).printf("\n Enter the side B:");
(7). scanf("%d", &b);
(8)..printf("\n Enter the side C:");
(9).  scanf("%d", &c);
(10). if((a>0) && (a<-100) && (b>0) && (b<100) && (c>0) && (c<=100)){
(11). if((a+b>c) && ((c+a)>b) && ((b+c)>a){
(12). boolean =1
(13).}
(14).}
(15). else {
```

```
(16) boolean = -1;
(17) }
(18) if (boolean = = 1) {
(19) if ((a = =b) && (b = =c)) {
(20) printf ("Triangle is equilateral");
(21) }
(22) else if ((a = =b) I I (b = = c) I I (c = = a)) {
(23) print ("Triangle is isosceles");
(24) }
(25) else {
(26) printf("Triangle is scalene”);
(27) }
(28) }
(29) else if (boolean = = 0) {
(30) printf ("Not a triangle");
(31) }
(32) else
(33) printf ("\n invalid input range");
(34) }
(35) getch ( );
(36) return -1;
(37) }
```

SAMPLE::
OUTPUT Draw Flow Graph of Triangle Problem



DD Path Graph: Since, nodes 1-9 are sequential nodes in the above flow graph, hence they are merged together as a single node – "a". Likewise we can go on deciding the merging of nodes & arrive at the following DD Path Graph


a
b
c
f
d
e
g
h
n
j
k
l
o
p
i
m
q
r

Following Decision Table::

Nodes in the flow Graph	**Corresponding Nodes of DD Path Graph**	**Justification**
1-9	a	Are Sequential nodes
10	b	Decision Nodes
11	C	Decision Nodes
12,13	d	Sequential nodes
14	e	Two Edge Combined
15,16,17	f	Sequential nodes
18	g	Decision Nodes
19	h	Decision Nodes
20,21	i	Sequential nodes
22	j	Decision Nodes
23,24	k	Sequential nodes
25,26,27	l	Sequential nodes
28	m	Three edge combined
29	n	Decision Nodes
30,31	o	Sequential nodes
32,33,34	p	Sequential nodes
35	q	Three edge combined

36,37	r	Sequential Exit Node

Calculation of **Cyclomatic Complexity** V (G) by three methods::

Method – 1: V(G) = e – n + 2 (Where “e” are edges & “n” are nodes)
V(G) = 23– 18+ 2 = 5 + 2 = 7

Method – 2: V(G) = P + 1 (Where P – No. of predicate nodes without degree = 2)
V(G) = 6 + 1 = 7 (Nodes b, c, g, h, j & n are predicate nodes with 2 outgoing edges)

Method – 3: V(G) = Number of enclosed regions + 1 = 6+1=7
(Here R1, R2, R3, R4, R5 & R6 are the enclosed regions and 1 corresponds to one outer region)
V (G) = 7 and is same by all the three methods.

Identification of the basis-set with Seven Paths
Path 1: a – b – f – g – n – p – q – r
Path 2: a – b – f – g – n – o – q – r

Path 3: a – b – c – e – g – n – p – q – r

Path 4: a – b – c – d – e – g – n – o – q – r

Path 5: a – b – f – g – h – i – m – q – r

CASE STUDY: Software Requirement Specification (SRS)

1. Objectives

- Gain a deeper understanding of the Software Requirement Specification phase and the Software Requirement Specification (SRS).
- Learn how to write requirements and specifications.
- Gain exposure to requirements management using RequisitePro.

2. Background

A requirement is a statement of a behavior or attribute that a system must possess for the system to be acceptable to a stakeholder.
Software Requirement Specification (SRS) is a document that describes the requirements of a computer system from the user's point of view. An SRS document specifies:

- The required behavior of a system in terms of: input data, required processing, output data, operational scenarios and interfaces.
- The attributes of a system including: performance, security, maintainability, reliability, availability, safety requirements and design constraints.

Requirements management is a systematic approach to eliciting, organizing and documenting the requirements of a system. It is a process that establishes and maintains agreement between the customer and the project team on the changing requirements of a system.
Requirements management is important because, by organizing and tracking the requirements and managing the requirement changes, you improve the chances of completing the project on time and under budget. Poor change management is a key cause of project failure.

2.1 Requirements Engineering Process

Requirements engineering process consists of four phases:

- **Requirements elicitation**: getting the customers to state exactly what the requirements are.
- **Requirements analysis:** making qualitative judgments and checking for consistency and feasibility of requirements
- **Requirements validation:** demonstrating that the requirements define the system that the customer really wants.
- **Requirements management:** the process of managing changing requirements during the requirements engineering process and system development, and identifying missing and extra requirements.

2.2 Writing Requirements

Requirements always need to be correct, unambiguous, complete, consistent, and testable.

2.2.1 Recommendations When Writing Requirements

- Never assume: others do now know what you have in mind.
- Use meaningful words; avoid words like: process, manage, perform, handle, and support.
- State requirements not features:

  - ➢ Feature: general, tested only for existence.
  - ➢ Requirement: specific, testable, measurable.
- Avoid:
  - ➢ Conjunctions: ask yourself whether the requirement should it be split into two requirements.
  - ➢ Conditionals: if, else, but, except, although.
  - ➢ Possibilities: may, might, probably, usually.

2.3 Writing Specifications

Specification is a description of operations and attributes of a system. It can be a document, set of documents, a database of design information, a prototype, diagrams or any combination of these things.

Specifications are different from requirements: specifications are sufficiently complete ─ not only what stakeholders say they want; usually, they have no conflicts; they describe the system as it will be built and resolve any conflicting requirements.

Creating specifications is important. However, you may not create specifications if:

- You are using a very incremental development process (small changes).
- You are building research or proof of concept projects.
- You rebuilding very small projects.
- It is not cheaper or faster than building the product.

2.4 Software Requirement Specification (SRS)

Remember that there is no “Perfect SRS”. However, SRS should be:

- Correct: each requirement represents something required by the target system.
- Unambiguous: every requirement in SRS has only one interpretation
- Complete: everything the target system should do is included in SRS (no sections are marked TBD-to be determined).
- Verifiable: there exists some finite process with which a person/machine can check that the actual as-built software product meets the requirements.
- Consistent in behavior and terms.
- Understandable by customers.
- Modifiable: changes can be made easily, completely and consistently.
- Design independent: doesn't imply specific software architecture or algorithm.
- Concise: shorter is better.
- Organized: requirements in SRS are easy to locate; related requirements are together.
- Traceable: each requirement is able to be referenced for later use (by the using paragraph numbers, one requirement in each paragraph, or by using convention for indication requirements)