Module 2

4 Software Requirements Analysis and Modeling

Syllabus

Requirement Engineering, Requirement Modeling, Data flow diagram, Scenario based model, Software Requirement Specification document format(IEEE)

4.1 Introduction

- Requirements engineering helps software engineers and software developers to better understand the problem and the nature of the problem they are going to work on. It includes the set of tasks that lead to understanding of :
  - What will be business impact of the software ?
  - What the customer wants exactly ?
  - How end user will interact with the system ?
- Software engineers (sometimes also called as system engineer or analyst) and other project stakeholders (managers, customers and users), all participate in requirements engineering.
- Designing and building the elegant computer software will not satisfy customer's requirements. If requirements analysis is wrong, then design will be wrong.
- Ultimately coding will be wrong. Finally, software expectations will not match with outcomes. Hence requirements engineering should be carried out carefully.

4.2 Requirement Engineering

- The requirements engineering is carried out through the execution of following functions. Some of these requirements engineering functions occur in parallel.
- All these seven functions focus on the customer's needs and care must be taken to satisfy it. All these functions collectively form the strong base for software design and construction. These functions are :

1. Inception
2. Elicitation
3. Elaboration
4. Negotiation
5. Specification
6. Validation
7. Requirement management

4.2.1 Inception

- Inception is beginning and it is usually said that, 'well beginning is half done'. But it is always problematic for the developer that what should be the starting point i.e. 'from where to start'.
- The customer and developer meet and they decide the overall scope and nature of the problem statement. The aim is :
  - To have the basic understanding of the problem.
  - To know the people who will use the software.
  - To know exact nature of problem that is expected from customer's side.
  - To maintain effectiveness of preliminary communication.
  - To have collaboration between customer and developer.
- The requirements engineering is a 'communication intensive' activity because a requirement gathering is an initial step for design. Hence exact requirements are gathered with customer communication.
- The developer must ask following questions:
  - Who is behind the request for this work?
  - Who will use the solution ?
  - What will be the economical benefits of a successful solution ?
  - Is there another source of solution for the same problem ?

4.2.2 Elicitation

- Elicitation means, 'to draw out the truth or reply from anybody'. In relation with requirements engineering, elicitation is a task that helps the customer to define what is required.
- To know the objectives of the system or the project to be developed is a critical job. Why it is difficult to get a clear understanding of customer wants? To answer this question, we have a number of problems like :

Problems of scope

Many times customer states unnecessary technical details. The unnecessary details may confuse developer instead of giving clarity of overall system objectives.

Problems of understanding

Sometimes both customer as well as developer has poor understanding of :

- What is needed ?
- Capabilities and limitations of the computing environment.
- Understanding of problem domain.
- Specify requirements those conflict with other customers and users.

Problems of volatility

Volatility means 'change from one state to another'. The customer's requirements may change time to time. Th is also a major problem while deciding fixed set of requirements.

4.2.3 Elaboration

- Elaboration means 'to work out in detail'. The information obtained during inception and elicitation is expande and modified during elaboration.
- Now requirements engineering activity focuses on developing the technical model of the software that wil include :
  - Functions
  - Features
  - Constraints
- Thus, elaboration is an 'analysis modeling' action. This model focuses on 'how the end user will interact with the system'.

4.2.4 Negotiation

- Here, 'Negotiation' means 'discussion on financial and other commercial issues'.
- In this step customer, user and other stakeholders discuss to decode :
  - To rank the requirements
  - To decide priorities
  - To decide risks
  - To finalize the project cost
  - The impact of above issues on project cost and delivery date.

4.2.5 Specification

- The specification is the final work product produced by requirement engineer. The specification may take different forms :
  - A written document
  - A set of graphical model
  - A formal mathematical model
  - A collection of scenarios
  - A prototype
  - Any combination of above
- The specification is considered as the foundation of all subsequent software engineering activities.

- The specification describes the function and performance of the computer based systems. The specification also describes the constraints are necessary in its development.

4.2.6 Validation

- All previous work completed will be just useless and meaningless if it is not validated against the customers expectations.
- The requirement validation checklist includes :
  - All requirements are stated clearly ?
  - Are the requirements misinterpreted ?
  - Does the requirements violate any system domain constrains?
  - Is the system requirement traceable to the system model that is created?
  - Are the requirements associated with performance, behaviour and operational characteristics ?

4.2.7 Requirement Management

- Requirement management is a software engineering task that helps the project team members to identify the requirements, control the requirements, track the requirements and changes to requirements at any time as the project exceeds.
- The requirement management task starts with identification and each of the requirements is assigned a unique identifier.
- After the requirements finalization, the traceability table is developed. Traceability table relates the requirements to one or more aspects of the system or it is environment. The traceability tables are used as requirements database and are useful in understanding how change in one particular requirement will affect other parts or aspects of the system.

4.2.8 Initiating the Requirement Engineering Process

- In requirements engineering process, following are considerable issues :
  - Customers may be located at different cities or countries.
  - Customers do not have clear idea of product requirements.
  - Different customers may have different and conflicting opinions about the system to be built.
  - Customers may have limited technical knowledge.
  - Customers may have only limited time to interact with requirement engineer.
- Hence, considering all these issues, following are the steps required to initiate the requirements engineering process :

Identifying the Stakeholders

- Stakeholder is anyone who has direct interest in or benefits from the system that is to be developed. Hence, we can list following people as stakeholders :

- Business operations manager
- Product managers
- Marketing people
- Internal and external customers
- End users
- Consultants
- Product engineers
- Software engineers
- Support and maintenance engineers
- Others

- All these people have different view of the system regarding the system that is to be developed.

Recognizing Multiple Viewpoints

- As many stakeholders exist, they all have different views regarding the system that is to be developed.

For example

- Marketing people are interested in functions and features having potential market.
- Business people are interested in functions and features those can be set within minimum budget.
- End users are interested in functions and features those will be easy to understand and use.
- Software engineers are interested in functions and features those support their existing infrastructure.
- Support engineer may focus on the maintainability of the software.
- It's a duty of software engineer to consider all these viewpoints of all the stakeholders and find the consistent and balanced set of requirements.

Working towards collaboration

Customers must collaborate with themselves and software engineering practitioners to get successful system. The job of requirement engineer is to find :

1. **Common areas :** The requirements for which all stakeholders agree.
2. **Conflict areas :** The requirements that are purposed by one stakeholder but opposed by another stakeholder.

   Now, the higher authorities like business manager or senior technologist can take the decision for the requirements either to forward or to reject these requirements.

Asking the First Questions

The requirements engineering is a 'communication intensive' activity because 'requirements gathering' is an initial step for design. Hence following three sets of questions are used to gain to understand the requirements :

1. **First set : The context free questions**
   - Who is behind the request for this work ?
   - Who will use the solution ?

- What will be the economical benefits of a successful solution ?
- Is there another source of solution for the same problem ?

2. **Second set : Questions to gain preliminary understanding of problem**
   - How will you characterize good output that will be generated by successful solution ?
   - What are the probable problems for this solution ?
   - What is the business environment in which this solution will be used ?
   - What are special performance issues and constraints that will affect the solution ?
3. **Third set : Questions those focus on effectiveness of the communication activity**
   - Are the questions official ?
   - Are the questions related to the problem ?
   - Are the questions 'too many' in quantity ?
   - Can anyone else provide additional information ?
   - Are some questions left to ask ?

4.3 Requirement Modelling

Q. List out Requirement models. Explain any one of them. **MU - Dec.19, 5 Marks**

- Analysis model uses a combination of text and diagrammatic forms to depict requirements of data, functions and behavior. So that it will be easy to understand overall requirements of the software to be built.
- The **'Software Engineer'** also called as **'Analyst'** builds the model using requirements stated by the customer.
- Analysis model validates the software requirements and represent the requirements in multiple dimensions.
- Basic aim of analysis modeling is to create the model that represents the information, functions and behavior of the system to be built.
- These information, functions and behavior of the system are translated into architectural, interface and component level designs in design modeling.
- Information, functions and behavior of the system are represented using number of different diagrammatic formats.

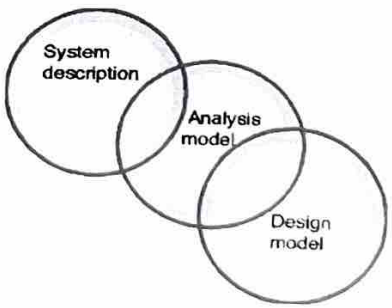


Fig. 4.3.1 : Analysis model, bridge in system description and design model

- As stated previously, the 'analysis model' acts as a bride between 'system description' and the 'design model'.
- The system description describes overall system functionality of the system including software, hardware, databases, human interfaces and other system elements. And the software design mainly focuses on application architectural, user interface and component level designs.
- Thus, all elements of analysis model are directly traceable to the parts of design model. Hence, the analysis model must have following three objectives :
  - To state clearly what customer wants exactly.
  - To establish the basis of the 'design model'. The information, functions and behaviour of the system defined by 'analysis model' are translated into architectural, interface and component level designs in 'design modeling'.
  - To bridge the gap between a system level description and software design.

4.4 Data Flow Diagram

4.4.1 Flow-Oriented Modelling

They provide necessary information that how data objects are transformed by processing the functions. Flow oriented elements are :

1. Data Flow Diagrams
2. Control Flow Diagrams
3. Control Specifications
4. Process Specifications

4.4.1(A) Data Flow Model

- Data Flow Diagram (DFD) is also called as 'Bubble chart' is a graphical technique, which is used to represent information flow, and transformers those are applied when data moves from input to output.
- DFD represents system requirements clearly and identify transformers those becomes programs in design. DFD consists of series of bubbles joined by lines.
- DFD may be further partitioned into different levels to show detailed information flow e.g. level 0, level 1, level 2 etc.

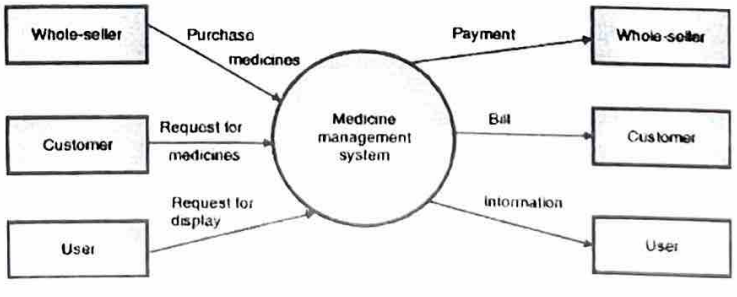


Fig. 4.4.1 : Data flow diagram for medicine managements system

- DFD focuses on the fact 'what data flow' rather 'how data is processed'.

Example of data flow diagram

The data flow diagram developed for **medicine management system** includes the external entities : Whole-seller, Customer and User. The DFD shows following requirements with proper flow of data from one entity to other.

- The medicines are purchased from whole-seller.
- The customer can request for the medicines in medicine store.
- User can request for display of the current status of the items in medicine.
- When medicines are purchased from whole-seller, the system generates the payment for whole-seller.
- When medicines are sold to customer, the system generates the bill for customer.
- The system also presents the required information to the user as per his requirements.

Developing level 1 DFD

- DFDs are used to represent information flow, and the transformers those are applied when data moves from input to output.
- To show the data flow with more details, the DFD is further extended to level 1, level 2, level 3 etc. as per the requirements.
- The typical value for the DFD is seven. Any system can be well represented with details up to seven levels.
- The level 0 and level 1 DFD for the 'Food Ordering System' in the restaurant' are shown in Figs. 4.4.2 and 4.4.3.

Notations used in DFD

- The circle represents the process.
- The rectangle represents the external entity like customer, whole-seller etc.
- The labelled arrows indicate incoming and outgoing data flow.
- The open rectangle shows the database or file.

DFD for food ordering system

Here customer, kitchen and restaurant managers are external entities.

- The 'food order system' accepts the food order from the customer and forwards the order to the kitchen.
- When the service is provided to the customer, the system generates the bill.
- A copy of customer bill can be submitted to the manager as a part of restaurant management report.

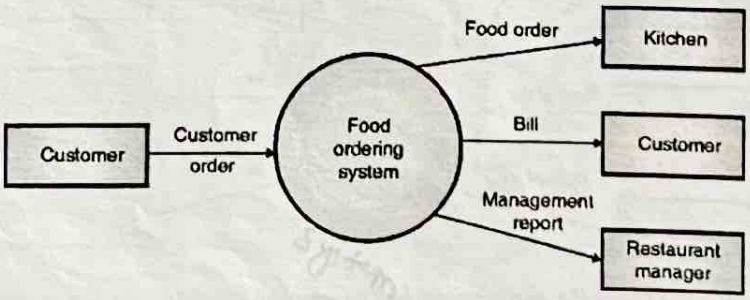


Fig. 4.4.2 : Level 0 DFD for 'food ordering system' in restaurant

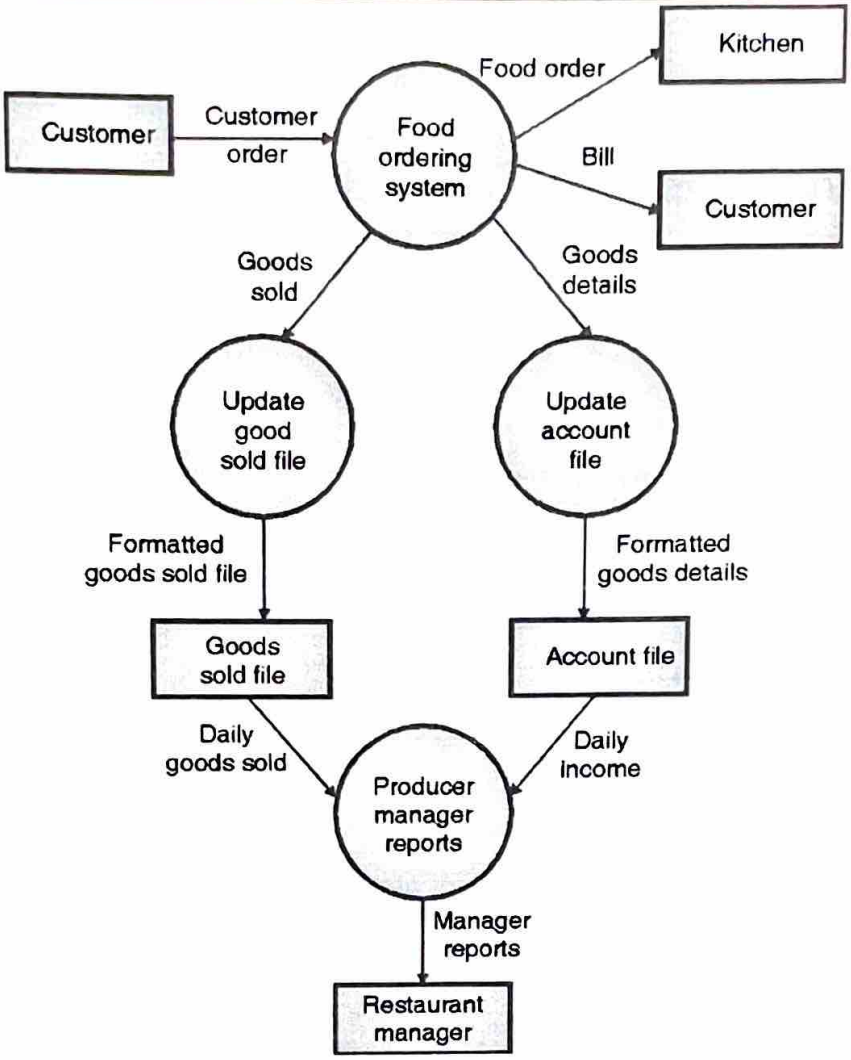


Fig. 4.4.3 : Level 1 DFD for 'food ordering system' in restaurant

This level 0 DFD can be extended to level 1 DFD to show more details showing exact data flow and processes (i.e. transformers).

4.4.1(B) Control Flow Model

The large class of applications having following characteristics requires control flow modeling :

- The applications those are driven by the events rather than data.
- The applications those produce control information rather than reports or displays.
- The applications those process information in specific time.

The control item or event is implemented as Boolean value. For example, true or false, on or off, 1 or 0.

4.4.1(C) Control Specifications

The control specification represents the behaviour of the system. The behaviour of the system can be represented using 'state transition diagram' which is also called as 'state chart diagram'.

A state chart diagram

A state chart diagram shows the state machine that consists of states, transitions, events and activities. This diagram shows how system makes the transition from one state to another state on occurrence of particular event.

State

A state of an object is defined as the condition that is satisfied in the life of an object and once the condition is satisfied, the object performs some activity and it waits for some events to occur.

Event

An event causes the system to make transition from one state to another. The notations used in state chart diagram are :

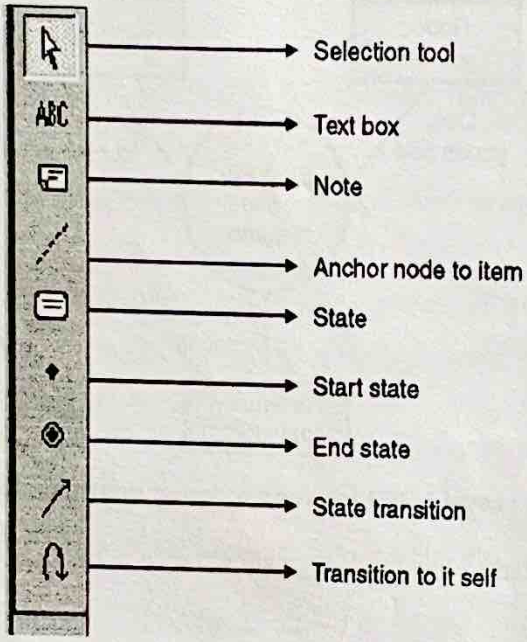


Fig. 4.4.4 : Notations in state chart diagram

Example of state chart diagram

Problem statement : To develop the state chart diagram for 'washing machine':

Explanation

- First, fill the water in washing machine. Then take detergent in. Now, motor is in running state.
- The 'motor running' state can go in ' motor stops' state in one of the two cases :
  - (i) When user of washing machine will press the stop switch or
  - (ii) Time expires
- When 'washing process' is completed, the transitions will take place for 'drying process'. When drying process is going on the machine reaches to end state in one of the two cases :

(i) When user of washing machine will press the stop switch or

(ii) Time expires

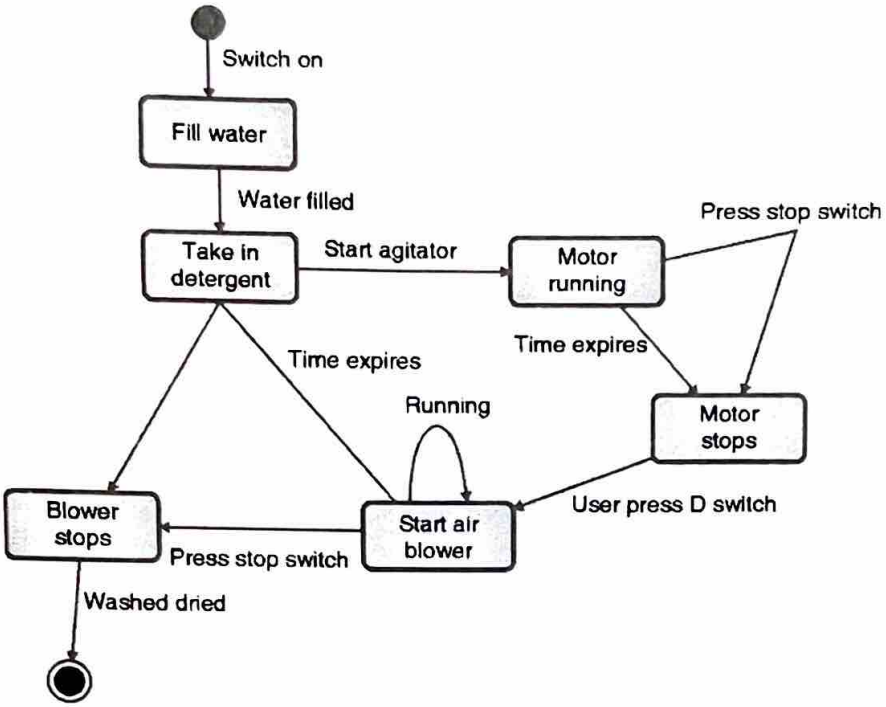


Fig. 4.4.5 : A state chart diagram for washing machine

4.4.1(D) Process Specifications (PSPEC)

The process specification is used to describe all flow model processes. The content of process specifications include Program Design Language (PDL). The PDL is also called as **'pseudo code'** or **'Structured English'**. PDL uses the vocabulary of English and syntax of some other language. PDL looks like the modern programming language but it cannot be compiled. PDL processors are used to convert PDL into graphical representation.

Example of process specification

Consider following part of DFD with process 'analyze triangle'.

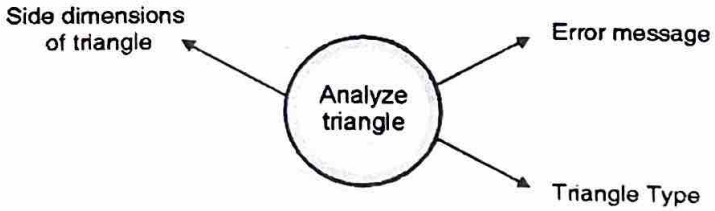


Fig. 4.4.6

4.5 Scenario Based Modelling : UML Models

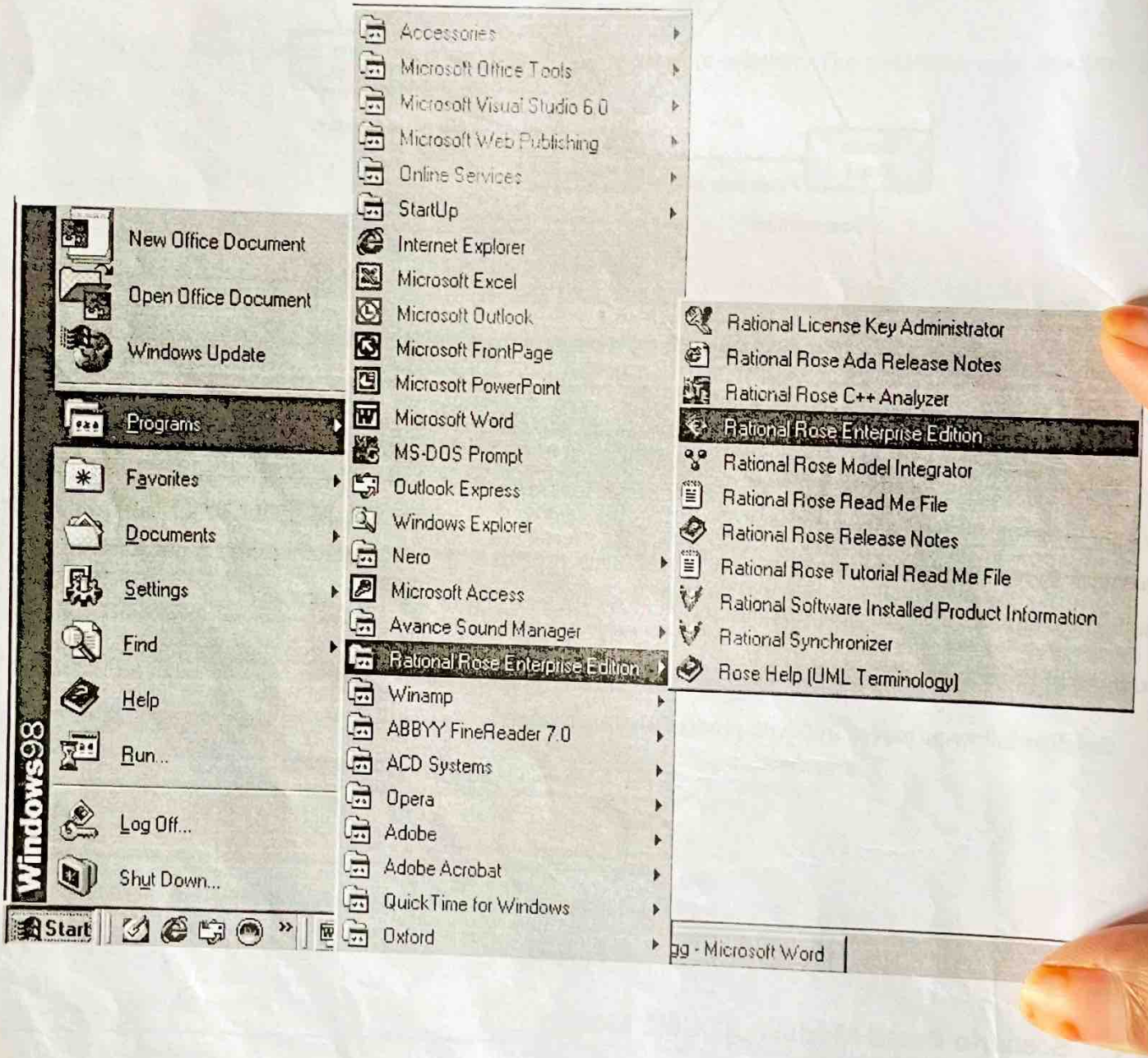
- Analysis modeling with UML begins with the creation of scenarios.
- In Scenario Base modeling the system is represented in user's point of view. Scenario-based elements are :

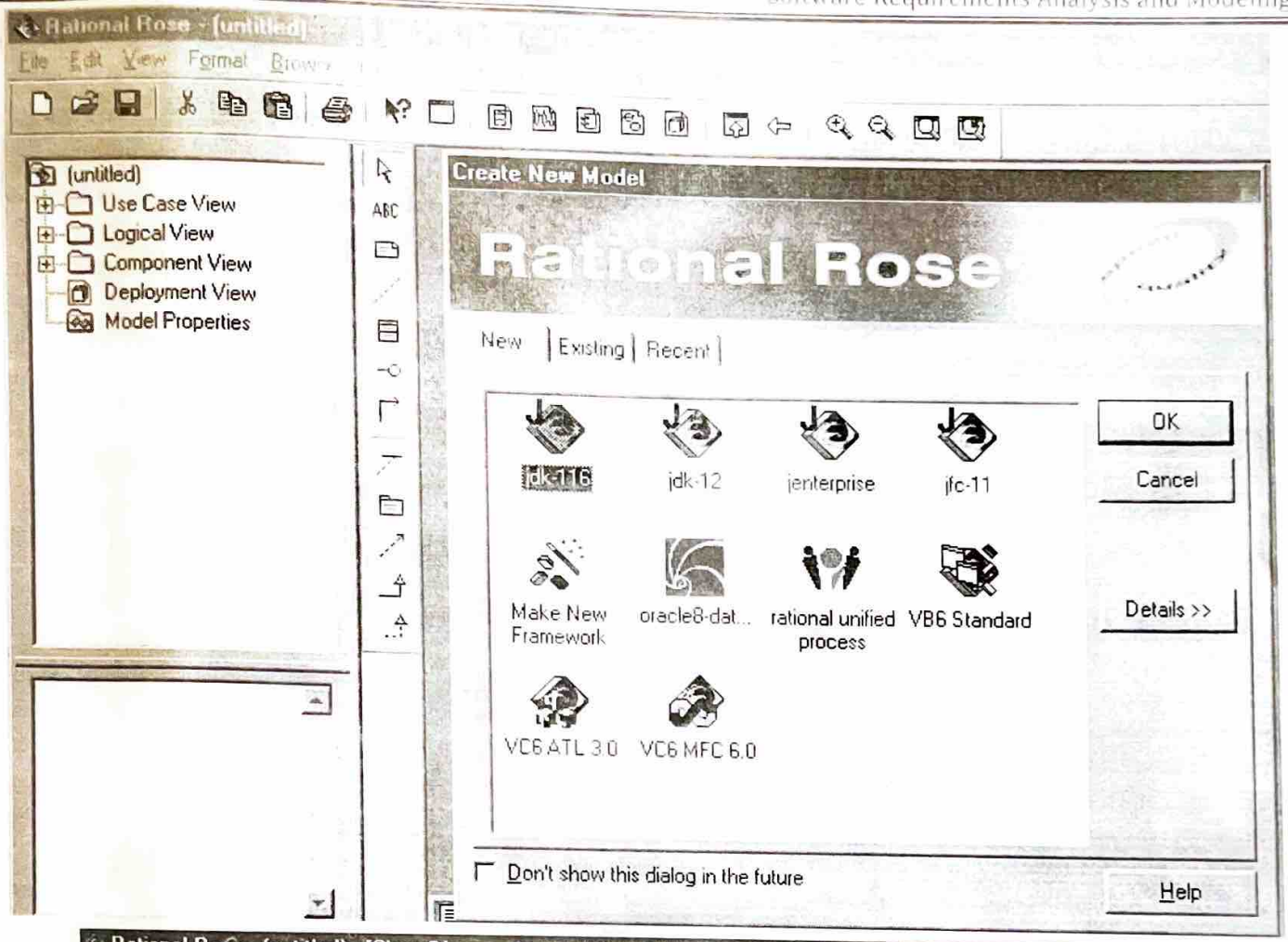
(1) Use case diagrams (2) Activity diagrams (3) Swim lane diagrams

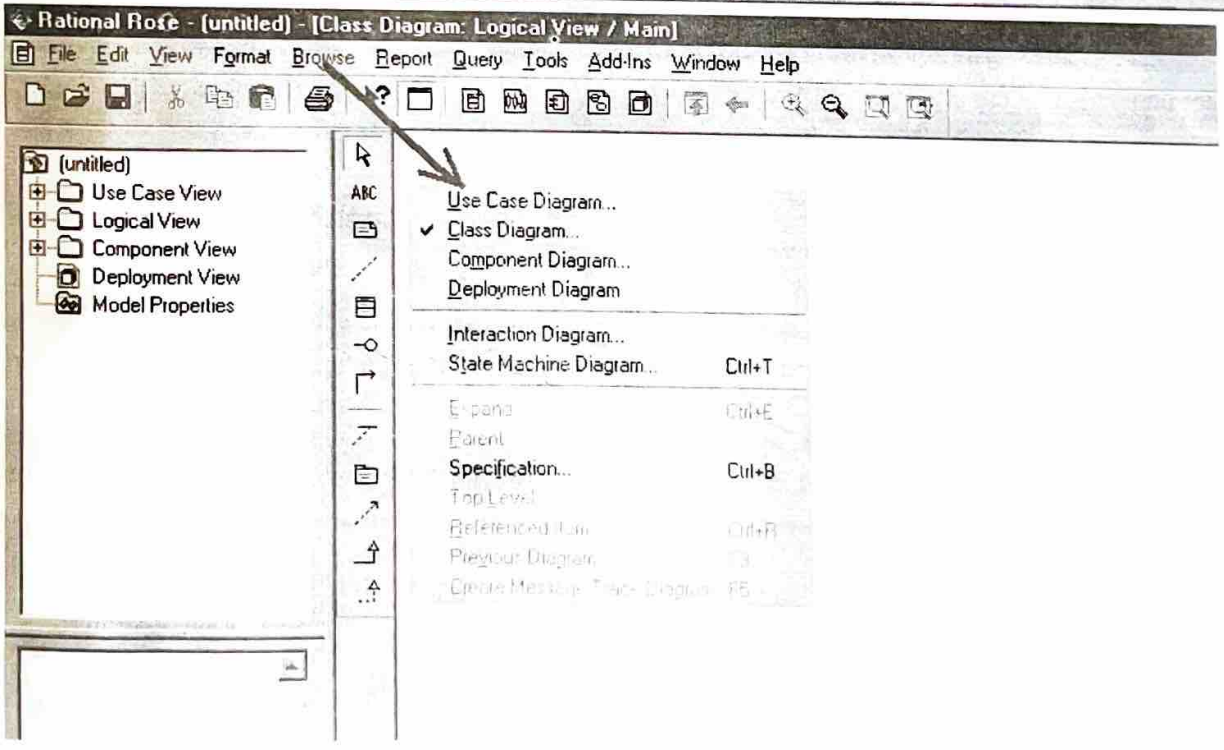
4.5.1 Diagramming in UML

During 1990 Rumbaugh, Booch and Jacobson combinely presented a Unified Modeling Language (UML) that includes notations for modeling and development of OO systems. By 1997 UML became a industry standard for Object Oriented software development and Rational corporation developed automated tools to support UML methods (i.e. Rational Rose software).

Using Rational Rose for UML diagrams




Rational Rose - [untitled]
File Edit View Format Browse
(untitled)
Use Case View
Logical View
Component View
Deployment View
Model Properties
Create New Model
Rational Rose
New
Existing
Recent
jdk-116
jdk-12
jenterprise
jfc-11
OK
Cancel
Make New Framework
oracle8-dat...
rational unified process
VB6 Standard
Details >>
VC6 ATL 3.0
VC6 MFC 6.0
Don't show this dialog in the future
Help


Rational Rose - (untitled) - [Class Diagram: Logical View / Main]
File Edit View Format Browse Report Query Tools Add-Ins Window Help
(untitled)
Use Case View
Logical View
Component View
Deployment View
Model Properties
Use Case Diagram...
Class Diagram...
Component Diagram...
Deployment Diagram
Interaction Diagram...
State Machine Diagram... Ctrl+T
Expand Ctrl+E
Parent
Specification... Ctrl+B
Top Level
Referenced Item Ctrl+R
Previous Diagram
Create Message Trace Diagram F5

4.5.2 Developing Use Cases Diagram

- To start developing a set of use-cases, the functions or the activities performed by actor are listed. The list of Use-cases may be obtained from one of the following sources :
  - From customer communication.
  - End user communication.
  - By evaluating the activity diagrams.
- Different notations used for use case diagrams are :

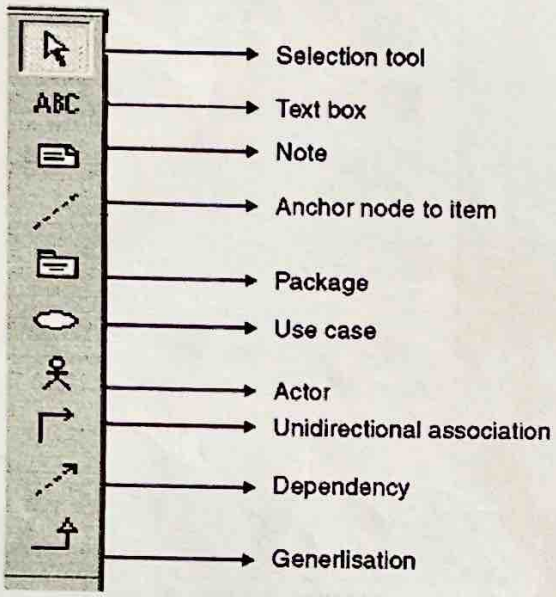


Fig. 4.5.1 : Notations in use case diagram

Example of use case diagram

Problem statement : To develop the Use case diagram for 'food ordering system' in hotel.

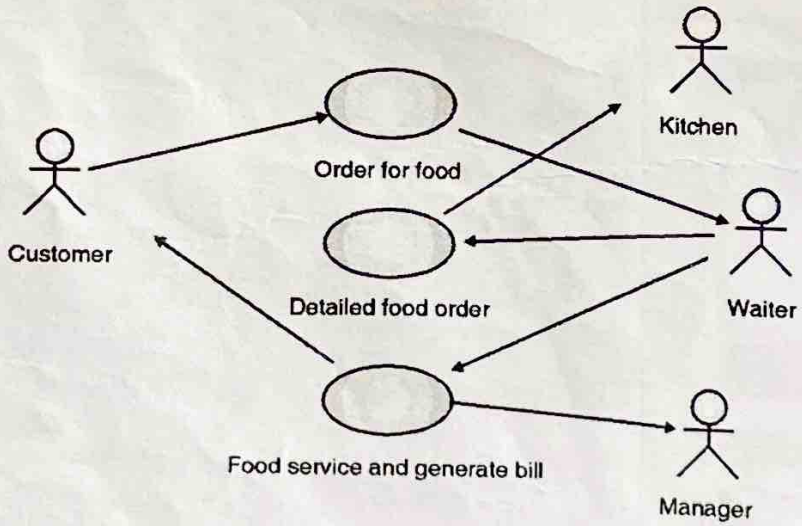


Fig. 4.5.2 : Use case diagram for food ordering system

Explanation

Customer orders for the food. The waiter receives this order. The waiter then forms detailed order. The detailed order is submitted to the kitchen and desired service is provided to the customer. Finally bill is produced. The bill is submitted to customer and bill report is sent to manager.

4.5.3 Developing Activity Diagram

Activity diagrams are same similar to that of flowcharts. The notations used in activity diagram are :

- Rounded rectangles imply specific system functions.
- Arrows represent flow through the system.
- Decision diamonds are used to depict branching decisions.

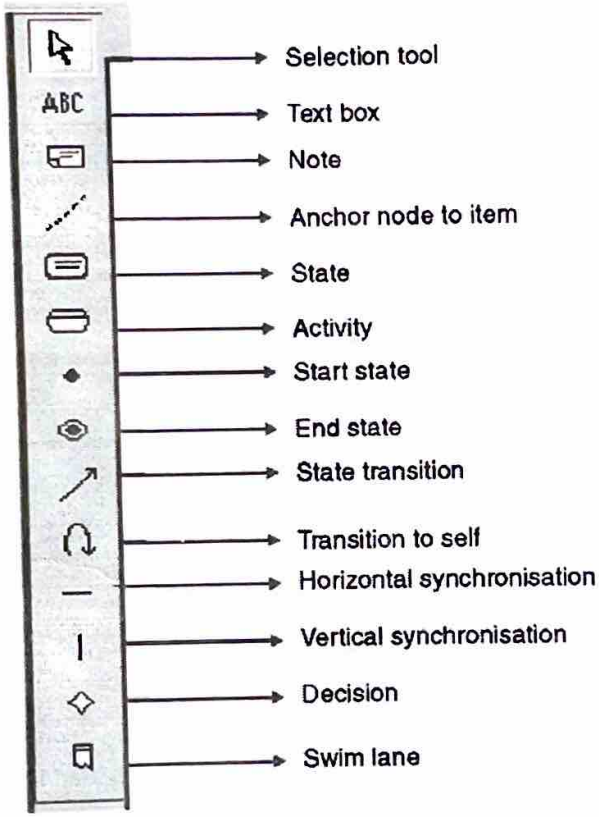


Fig. 4.5.3 : Notations in activity diagram

Example of activity diagram

Problem statement : To develop the activity diagram for 'Building Plan'.

Explanation

- First 'dig foundation' activity will be completed. After completing foundation, two activities 'build walls' and 'connect service' are two activities those can run parallel.

- After completing these activities 'build roof' activity can be started.
- While 'wiring' and 'carpentry' is going on sequentially, at the same time 'fit windows' activity can be completed parallel. Finally, 'painting' activity is completed.

4.5.4 Swim Lane Diagram

- Swim lane diagrams are the variation in activity diagrams. Swim lane diagrams allows to represent, the flow of activities described by use cases.
- At the same time these diagrams indicate which actors are involved in specific functions.
- We can also show the interaction between these different activities shown in different lanes.
- To show such parallel activities, diagram is divided vertically into lanes, similar to lanes of swimming pool. Hence these diagrams are called as 'swim lane' diagram.

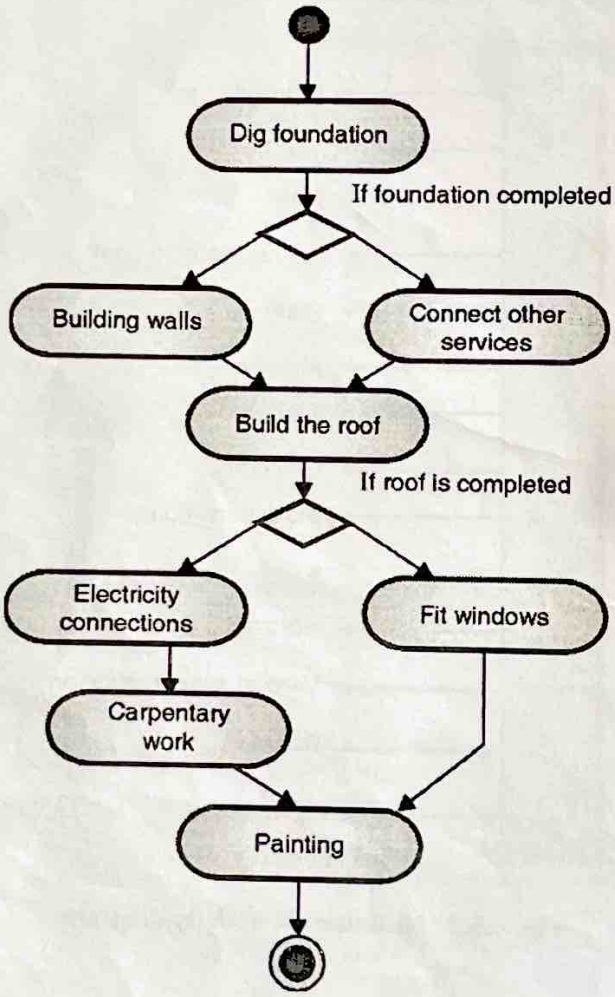


Fig. 4.5.4

Example of swim lane diagram

Problem statement : To develop the swim lane diagram for 'purchase equipment order'.

Explanation

- First, Head of the department will submit the requirement for purchasing the equipments

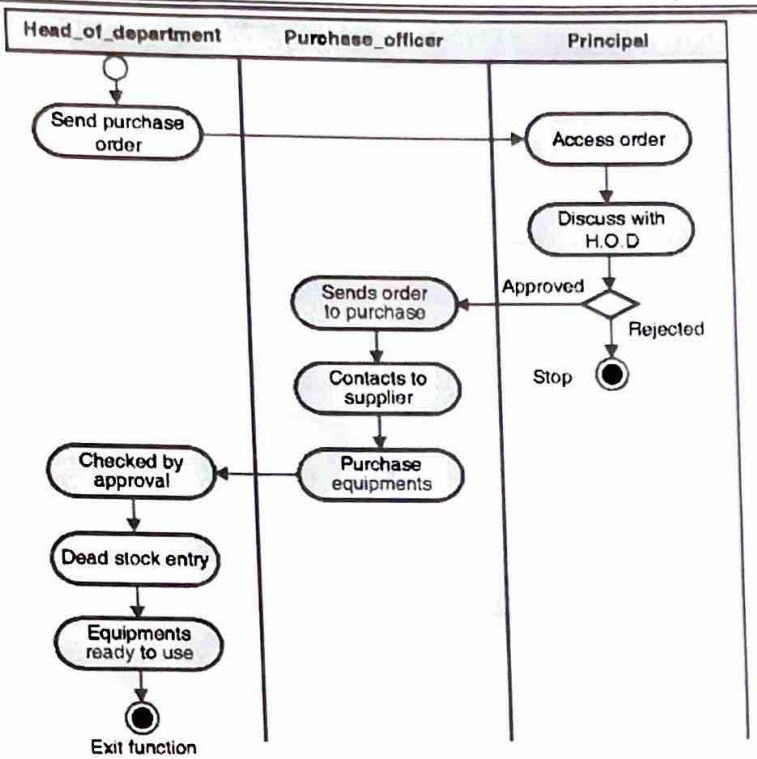


Fig. 4.5.5

- The order is reviewed by the principal.
- If the principal permits the proposal, the order is forwarded to purchase officer.
- Purchase officer now contacts to supplier.
- After purchasing the required equipments, they are checked by H.O.D.
- Finally, the dead stock entry is done in dead stock of the department.
- Purchased equipments are ready for use.

4.5.5 Class Diagram

Example class diagram

The class diagram for computer based system is shown with two important symbols :

Aggregation

Aggregation denotes super class to subclasses relationship.

For example in Fig. 4.5.6, we can say that Monitor, Printer and Plotter are subclasses of super class 'Output Device'.

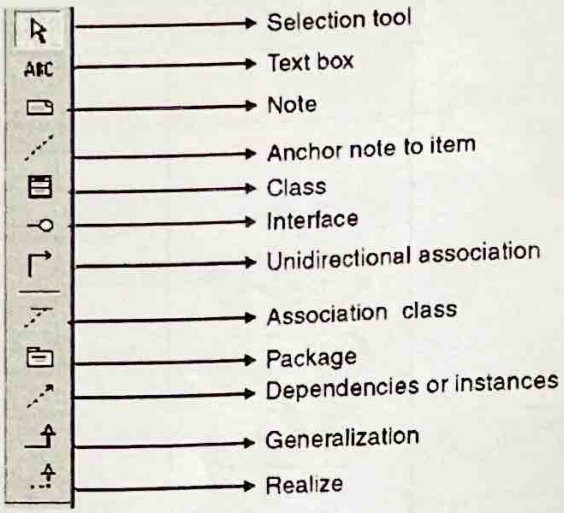


Fig. 4.5.6 : Notations in class Diagram

Generalization

- This symbol is used to denote 'is part of' or 'is composed of' relation.
- **For example :** In Fig. 4.5.7 we can say that class CPU is composed of classes RAM, Control Unit and ALU.

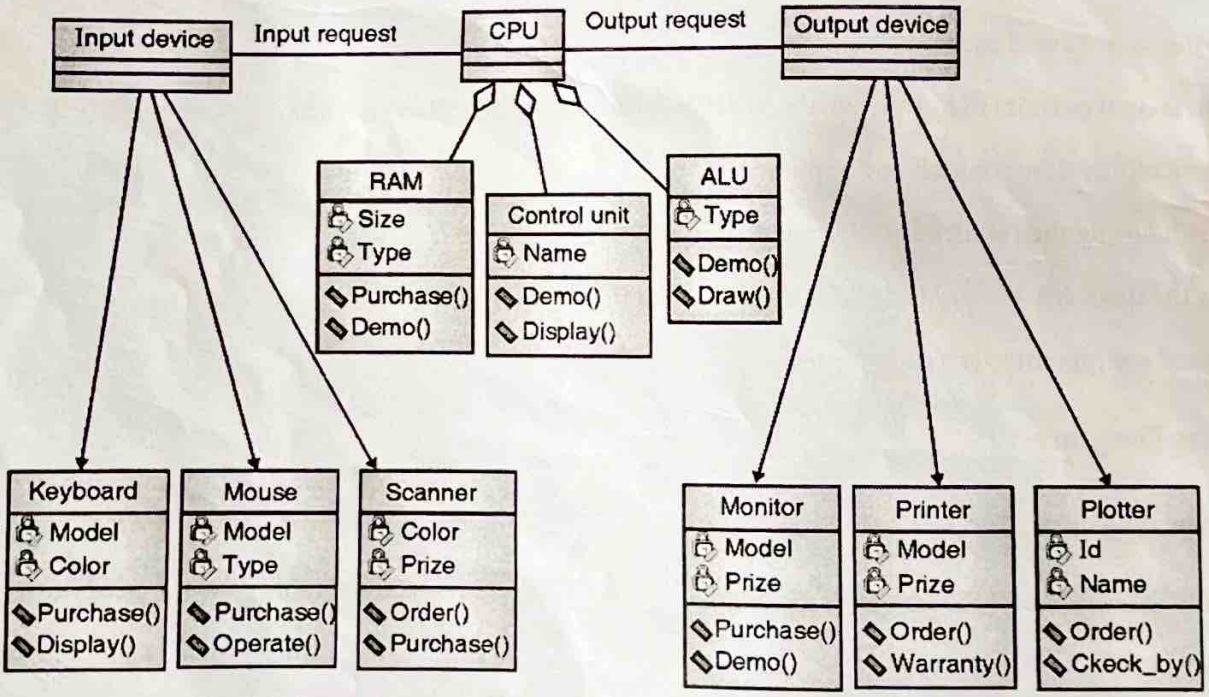


Fig. 4.5.7

Associations and Dependency

Associations

Two classes may be related to one another. This relation is called as 'association'. For example : In computer based system shown in Fig. 4.5.8, class 'input device' is associated with class 'CPU' and class 'CPU' is associated with class 'output device'.

Multiplicity

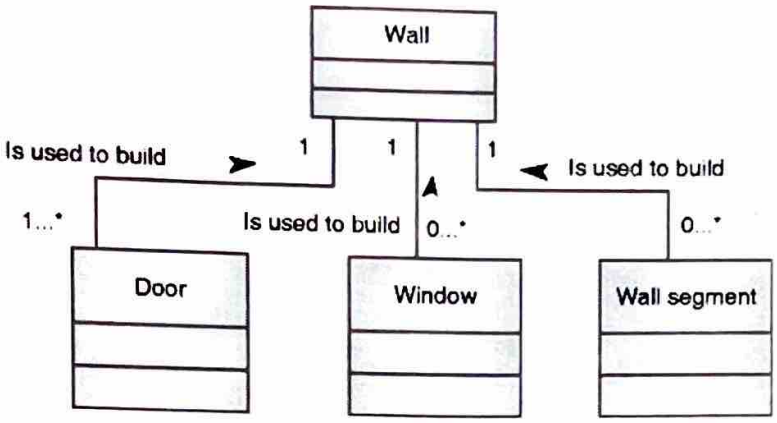


Fig. 4.5.8

The association is further defined by indicating multiplicity. It shows how many occurrences of one class ar related to how many occurrences of another class. For example, from Fig. 4.5.8, we can read that 1...* doors are use in building one wall.

Dependency

Sometimes the application requires showing the dependencies between the classes.

For example : The librarian will get access to the 'server system' if he enters the correct 'password'. Thi dependency is shown in Fig. 4.5.9.

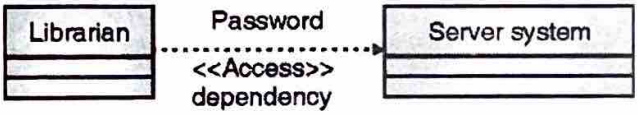


Fig. 4.5.9

Analysis Packages

- **Package :** The package is used to assemble a collection of related classes.
- **For example** : The analysis model of video game may include following packages :
    - Environment
    - Characters
- In package environment we can have following related classes :
    - Tree
    - Landscape
    - Road
    - Wall
    - Bridge

- Building
- Visual effect
- Scene

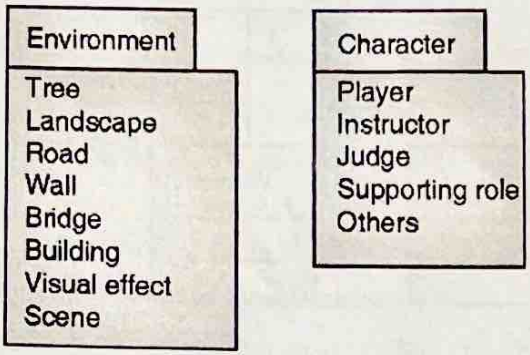


Fig. 4.5.10 : Environment and characters packages in video game

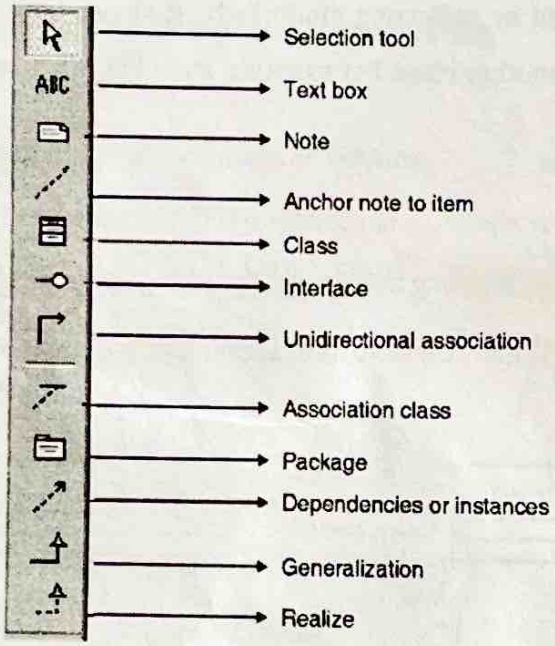


Fig. 4.5.11 : Notations in package diagram

4.6 Software Requirements Specification (SRS) Document Format(IEEE)

- Basically SRS or software requirement specification is an official document or the statement of what the system developers should implement.
- It includes both :
  - System requirement.
  - User requirement.
- The SRS has a set of users like senior management of the organization, engineers responsible for developing the software.

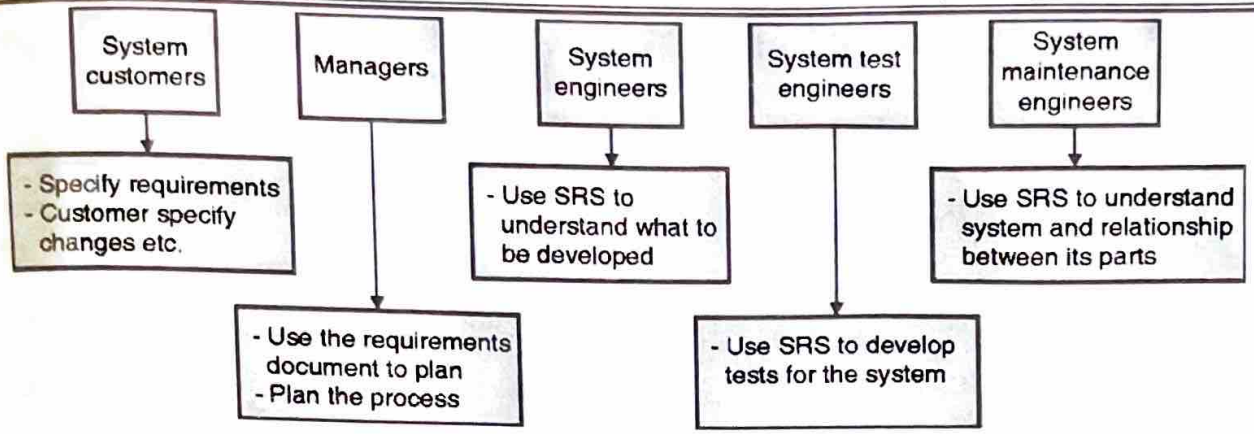


Fig. 4.6.1 : Users of SRS

- The details which are included in SRS depend on the type of system that is being developed and the type of the development process.
- A number of large organisations, such as IEEE have defined standards for software requirements document.
- The most widely used standard is IEEE/ANSI 830-1998. This standard suggests the following structure for requirements document :

1. Introduction
   - Purpose of SRS
   - Scope of product.
   - Definitions, acronyms, abbreviations.
   - References
   - Overview
2. General description
   - Product perspective
   - Product Functions
   - User characteristics
   - General constraints
   - Assumptions and dependencies
3. Specific requirements
4. Appendices
5. Index.

- SRS is very useful when an outside contractor is developing the software system.
- For business system, where requirements are unstable, SRS play an important role.

4.6.1 Writing Software Requirements Specifications

- Proper software requirement gathering is very important at the start of the software development. Generally it is done by professional software developers.

- Document is created in the requirement analysis which is generally referred as SRS or software requirement specification document.
- It is first project deliverable.
- SRS records the end user needs in certain format. This is SRS is needed when actual software development process starts.

4.6.2 What is a Software Requirements Specification?

- Before starting software development requirement is captured from the end users or clients or customer. This requirement is written in certain format which is called as SRS. Generally this document is created before starting the development work.
- It signifies both developers and client understood what should be implemented in the software.
- All capabilities and functionalities are stored in SRS.
- Requirement document needs in every steps of software development.
- Analyst uses this document for designing the software. Developer uses this document during the coding whereas software tester uses this document to check the functionalities of the software.
- All remaining project documents are depends upon requirement document because of which it is referred as parent of all document.
- It contains functional and non functional requirements.

Good SRS have accomplishes following goals

- It gives feedback to customer.
- The large problem can be divided into different small components.
- It acts as input to design which is part of design specification.
- It acts as product validation check.

4.6.3 What Kind of Information Should an SRS Include?

Following things are part of SRS :

- Interfaces
- Functional capabilities
- Performance levels
- Data structures/Elements
- Safety
- Reliability

Security/Privacy

Quality

Constraints and limitations

6.4 SRS Template

Different existing SRS templates can be used in writing SRS documents.

We can select the template which matches our requirement.

Template will just acts as guiding principles for writing template. Proper changes need to be done whenever necessary in template.

Table 4.6.1 shows SRS outline :

Table 4.6.1 : A sample of a basic SRS outline

1. **Introduction** : It contains different details like purpose of software, conventions used, target audiences, extra information, SRS team members, references.
2. **Overall Description** : Overall information of the project is given in this sections. It includes perspective, functions, different user classes, working environment, design constraints, assumptions about the software.
3. **External Interface Requirements :** It contains external interface information which includes user interface, hardware interface, software interfaces, communication protocols, etc.
4. **System Features :** Which system features needs to add is given in system features. It includes different features, features description, priority, action result, functionalities, etc.
5. **Other Non functional Requirements** : Non functional requirement of the project is given in this section. It includes performance requirement, safety concerns, security constraints, quality factors, documentation, user documentation.
6. **Other Requirements** : It includes Appendices, glossary of the software, etc.

4.6.5 Characteristics of an SRS

- **Complete :** All the project functionalities should be recorded by SRS.
- **Consistent :** SRS should be consistent
- **Accurate :** SRS defines systems functionality in real world. We need to record requirement very carefully.
- **Modifiable :** Logical and hierarchical modification should be allowed in SRS.
- **Ranked :** Requirement should be ranked using different factors like stability, security, ease or difficulty.
- **Testable :** The requirement should be realistic and able to implement.
- **Traceable :** Every requirement we must be able to uniquely identify.

- **Unambiguous :** Statement in the SRS document should have only one meaning.
- **Valid :** All the requirements should be valid.

4.6.6 Structured Specifications for an Insulin Pump Case Study

- Structured natural language is a way of writing system requirements where the freedom of the requirements writer is limited and all requirements are written in a standard way. This approach maintains most of the expressiveness and understandability of natural language but ensures that some uniformity is imposed on the specification. Structured language notations use templates to specify system requirements. The specification may use programming language constructs to show alternatives and iteration, and may highlight key elements using shading or different fonts.
- The Robertsons, a well known author, recommend that user requirements be initially written on cards, one requirement per card. They suggest a number of fields on each card, such as the requirements rationale, the dependencies on other requirements, the source of the requirements, supporting materials, and so on. This is similar to the approach used in the example of a structured specification shown in Table 4.6.2.

Table 4.6.2 : A structured specification of a requirement for an insulin pump

Function	**Compute insulin dose: Safe sugar level.**
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose—the dose in insulin to be delivered.
Destination	Main control loop.
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requirements	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

- To use a structured approach to specifying system requirements, you define one or more standard templates for requirements and represent these templates as structured forms. The specification may be structured around the objects manipulated by the system, the functions performed by the system, or the events processed by the system. An example of a form-based specification, in this case, one that defines how to calculate the dose of insulin to be delivered when the blood sugar is within a safe band, is shown in Table 4.6.2.

4.6.7 Tabular Specifications for an Insulin Pump Case Study

- Using structured specifications removes some of the problems of natural language specification. Variability in the specification is reduced and requirements are organized more effectively. However, it is still sometimes difficult to write requirements in a clear and unambiguous way, particularly when complex computations (e.g., how to calculate the insulin dose) are to be specified.
- To address this problem, you can add extra information to natural language requirements, for example, by using tables or graphical models of the system. These can show how computations proceed, how the system state changes, how users interact with the system, and how sequences of actions are performed.
- Tables are particularly useful when there are a number of possible alternative situations and you need to describe the actions to be taken for each of these. The insulin pump bases its computations of the insulin requirement on the rate of change of blood sugar levels. The rates of change are computed using the current and previous readings. Table 4.6.3 is a tabular description of how the rate of change of blood sugar is used to calculate the amount of insulin to be delivered.

Table 4.6.3 : Tabular specification of computation for an insulin pump

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase Decreasing ((r2 - r1) < (r1 - r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 — r1) ≥ (r1 — r0))	CompDose = round ((r2 — r1)/4) If rounded result = 0 then CompDose = MinimumDose