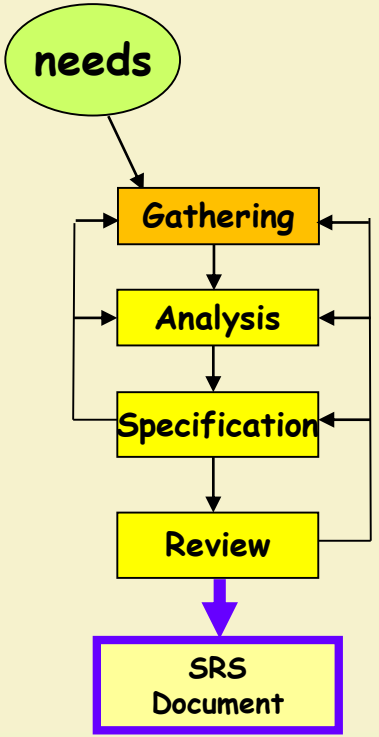
How to Gather Requirements?

- Observe existing (manual) systems,
- Study existing procedures,
- Discuss with customer and end-users,
- Input and Output analysis
- Analyze what needs to be done



Requirements Gathering Activities

- 1. Study existing documentation
- 2. Interview
- 3. Task analysis
- 4. Scenario analysis
- 5. Form analysis

Requirements Gathering (CONT.)

- In the absence of a working system,
  - Lot of imagination and creativity are required.
- Interacting with the customer to gather relevant data:
  - Requires a lot of experience.

Requirements Gathering (CONT.)

- Some desirable attributes of a good requirements analyst:
  - Good interaction skills,
  - Imagination and creativity,
  - Experience…

Case Study: Automation of Office Work at CSE Dept.

- The academic, inventory, and financial information at the CSE department:
  - At present carried though manual processing by two office clerks, a store keeper, and two attendants.
- Considering the low budget he had at his disposal:
  - The HoD entrusted the work to a team of student volunteers.

Case Study: Automation of Office Work at CSE Dept.

- The team was first briefed by the HoD:
  - Concerning the specific activities to be automated.
- The analysts first discussed with the two office clerks:
  - Regarding their specific responsibilities (tasks) that were to be automated.

Interview

- The analyst also interviewed student and faculty representatives who would also use the software.

Case Study: Automation of Office Work at CSE Dept.

- For each task that a user needs the software to perform, they asked:

  Task and Scenario Analysis

  - The steps through which these are to be performed.
  - The various scenarios that might arise for each task.

- Also collected the different types of forms that were being used.

  Form Analysis

Case Study: Automation of Office Work at CSE Dept.

- The analysts understood the requirements for the system from various user groups:

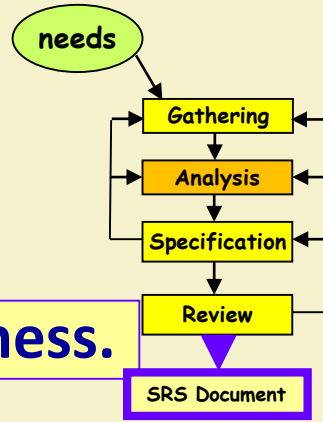
  Requirements Analysis

  - Identified inconsistencies, ambiguities, incompleteness.
- Resolved the requirements problems through discussions with users:
  - Resolved a few issues which the users were unable to resolve through discussion with the HoD.

  Requirements Specification
- Documented the requirements in the form of an SRS document.

Analysis of Gathered Requirements

- Main purpose of req. analysis:
  - Clearly understand user requirements,
  - **Detect inconsistencies, ambiguities, and incompleteness.**



- Incompleteness and inconsistencies:
  - Resolved through further discussions with the end-users and the customers.

Ambiguity

"All customers must have the same control field."

Do you notice any problems?

Multiple interpretations possible

1. All control fields have the same format.
2. One control field is issued for all customers.

Inconsistent Requirement

- Some part of the requirement:
  - contradicts some other requirement.
- **Example:**
  - One customer says turn off heater and open water shower when temperature > 100°C
  - Another customer says turn off heater and turn ON cooler when temperature > 100°C

- Some requirements not included:
  - Possibly due to oversight.

Incomplete Requirement

- **Example:**
  - The analyst has not recorded that when temperature falls below 90°C :
    - heater should be turned ON
    - water shower turned OFF.

Analysis of the Gathered Requirements

- Requirements analysis involves:
  - Obtaining a clear, in-depth understanding of the software to be developed
  - Remove all ambiguities and inconsistencies from the initial customer perception of the problem.

Analysis of the Gathered Requirements (CONT.)

- It is quite difficult to obtain:
  - A clear, in-depth understanding of the problem:
    - Especially if there is no working model of the problem.

Analysis of the Gathered Requirements (CONT.)

- Experienced analysts take considerable time:
  - **Clearly understand the exact requirements the customer has in his mind.**

Analysis of the Gathered Requirements (CONT.)

- Experienced systems analysts know - often as a result of painful experiences ---
  - **"Without a clear understanding of the problem, it is impossible to develop a satisfactory system."**

Analysis of the Gathered Requirements

- Several things about the project should be clearly understood:
  - What is the problem?
  - What are the possible solutions to the problem?
  - What complexities might arise while solving the problem?

Analysis of the Gathered Requirements

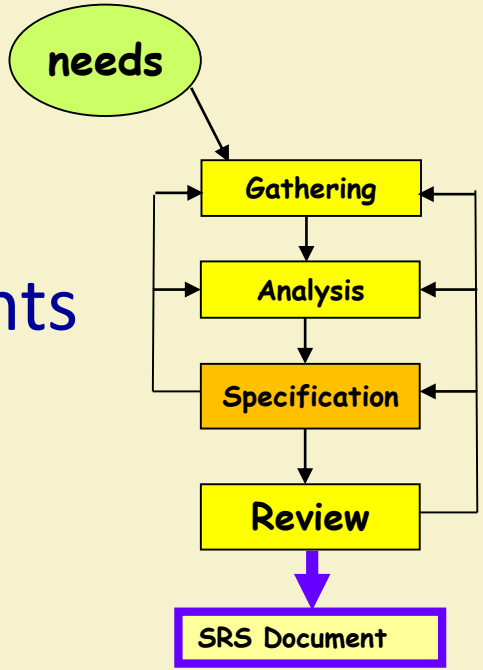
- Some anomalies and inconsistencies can be very subtle:
  - Escape even most experienced eyes.
  - If a formal specification of the system is constructed,
    - Many of the subtle anomalies and inconsistencies get detected.

Analysis of the Gathered Requirements(CONT.)

- After collecting all data regarding the system to be developed,
  - Remove all inconsistencies and anomalies from the requirements,
  - Systematically organize requirements into a Software Requirements Specification (SRS) document.

Software Requirements Specification

- Main aim:
  - Systematically organize the requirements arrived during requirements analysis.
  - Document requirements properly.



SRS Document

- As already pointed out--- useful in various contexts:
  - **Statement of user needs**
  - **Contract document**
  - **Reference document**
  - **Definition for implementation**

SRS Document (CONT.)

- SRS document is known as **black-box specification**:
  - The system is considered as a black box whose internal details are not known.

  - **Only its visible external (i.e. input/output) behavior is documented.**

SRS Document (CONT.)

- SRS document concentrates on:
  - What needs to be done in terms of input-output behaviour
  - Carefully avoids the solution ("how to do") aspects.

SRS Document (CONT.)

- The requirements at this stage:
  - Written using end-user terminology.
- If necessary:
  - Later a formal requirement specification may be developed from it.

- **It should be concise**
  - and at the same time should not be ambiguous.
- **It should specify what the system must do**
  - and not say how to do it.
- **Easy to change.,**
  - i.e. it should be well-structured.
- **It should be consistent.**
- **It should be complete.**

Properties of a Good SRS Document

Properties of a Good SRS Document (cont…)

- **It should be traceable**
  - You should be able to trace which part of the specification corresponds to which part of the design, code, etc and vice versa.
- **It should be verifiable**
  - e.g. “system should be user friendly” is not verifiable

SRS should not include...

- **Project development plans**
  - E.g. cost, staffing, schedules, methods, tools, etc
    - Lifetime of SRS is until the software is made obsolete
    - Lifetime of development plans is much shorter
- **Product assurance plans**
  - Configuration Management, Verification & Validation, test plans, Quality Assurance, etc
    - Different audiences
    - Different lifetimes
- **Designs**
  - Requirements and designs have different audiences
  - Analysis and design are different areas of expertise

SRS Document (CONT.)

- Four important parts:
  - **Functional requirements,**
  - **External Interfaces**
  - **Non-functional requirements,**
  - **Constraints**

Functional Requirements

- Specifies all the functionality that the system should support
  - **Heart of the SRS document:**
  - **Forms the bulk of the Document**
- Outputs for the given inputs and the relationship between them
- Must specify behavior for invalid inputs too!

Functional Requirement Documentation

- **Overview**
  - describe purpose of the function and the approaches and techniques employed
- **Inputs and Outputs**
  - sources of inputs and destination of outputs
  - quantities, units of measure, ranges of valid inputs and outputs
  - timing

Functional Requirement Documentation

- **Processing**
  - validation of input data
  - exact sequence of operations
  - responses to abnormal situations
  - any methods (eg. equations, algorithms) to be used to transform inputs to outputs

Nonfunctional Requirements

- Characteristics of the system which can not be expressed as functions:
  - **Maintainability,**
  - **Portability,**
  - **Usability,**
  - **Security,**
  - **Safety, etc.**

Nonfunctional Requirements

- Reliability issues
- Performance issues:
  - **Example:** How fast can the system produce results?
    - At a rate that does not overload another system to which it supplies data, etc.
    - Response time should be less than 1sec 90% of the time
    - Needs to be measurable (verifiability)

Constraints

- Hardware to be used,
- Operating system
  - or DBMS to be used
- Capabilities of I/O devices
- Standards compliance
- Data representations by the interfaced system

- User interfaces
- Hardware interfaces
- Software interfaces
- Communications interfaces with other systems
- File export formats

External Interface Requirements

Goals of Implementation

- Goals describe things that are desirable of the system:
  - But, would not be checked for compliance.
  - For example,
    - Reusability issues
    - Functionalities to be developed in future

IEEE 830-1998 Standard for SRS

- Title
- Table of Contents
- 1. Introduction
  - 1.1 Purpose
  - 1.2 Scope
  - 1.3 Definitions. Acronyms, and Abbreviations
  - 1.4 References
  - 1.5 Overview
- 2. Overall Description
- 3. Specific Requirements
- Appendices
- Index

•Describe purpose of the system
•Describe intended audience

•What the system will and will not do

•Define the vocabulary of the SRS (may also be in appendix)

•List all referenced documents and their sources SRS (may also be in appendix)

•Describe how the SRS is organized

IEEE 830-1998 Standard – Section 2 of SRS

- Title
- Table of Contents
- 1. Introduction
- 2. **Overall Description**
  - **2.1 Product Perspective**
  - **2.2 Product Functions**
  - **2.3 User Characteristics**
  - **2.4 Constraints**
  - **2.5 Assumptions and Dependencies**
- 3. Specific Requirements
- 4. Appendices
- 5. Index

•Present the business case and operational concept of the system
•Describe external interfaces: system, user, hardware, software, communication
•Describe constraints: memory, operational, site adaptation

•Summarize the major functional capabilities

•Describe technical skills of each user class

•Describe other constraints that will limit developer's options; e.g., regulatory policies; target platform, database, network, development standards requirements

IEEE 830-1998 Standard – Section 3 of SRS (1)

- …
- 1. Introduction
- 2. Overall Description
- 3. Specific Requirements
  - **3.1 External Interfaces**
  - **3.2 Functions**
  - **3.3 Performance Requirements**
  - **3.4 Logical Database Requirement**
  - **3.5 Design Constraints**
  - **3.6 Software System Quality Attributes**
  - **3.7 Object Oriented Models**
- 4. Appendices
- 5. Index

Specify software requirements in sufficient detail so that designers can design the system and testers can verify whether requirements met.

State requirements that are externally perceivable by users, operators, or externally connected systems

Requirements should include, at the least, a description of every input (stimulus) into the system, every output (response) from the system, and all functions performed by the system in response to an input

IEEE 830-1998 Standard – Section 3 of SRS (2)

- ...
- 1. Introduction
- 2. Overall Description
- 3. Specific Requirements
  - 3.1 External Interfaces
  - 3.2 Functions
  - 3.3 Performance Requirements
  - 3.4 Logical Database Requirements
  - 3.5 Design Constraints
  - 3.6 Software System Quality Attributes
  - 3.7 Object Oriented Models
- 4. Appendices
- 5. Index

•Detail all inputs and outputs
(complement, not duplicate, information presented in section 2)
•Examples: GUI screens, file formats

•Include detailed specifications of each use case, including collaboration and other diagrams useful for this purpose

Types of Data entities and their relationships

Standards compliance and specific software/hardware to be used

•Class Diagram, State and Collaboration Diagram, Activity Diagrams etc.

IEEE 830-1998 Standard – Templates

- Section 3 (Specific Requirements)can be organized in several different ways based on
  - **Modes**
  - **User classes**
  - **Concepts (object/class)**
  - **Features**
  - **Stimuli**

Example Section 3 of SRS of Academic Administration Software

- **SPECIFIC REQUIREMENTS**

  3.1 Functional Requirements

  3.1.1 Subject Registration

  – The subject registration requirements are concerned with functions regarding subject registration which includes students selecting, adding, dropping, and changing a subject.

- **F-001:**

  – The system shall allow a student to register a subject.

- **F-002:**

  – It shall allow a student to drop a course.

- **F-003:**

  – It shall support checking how many students have already registered for a course.

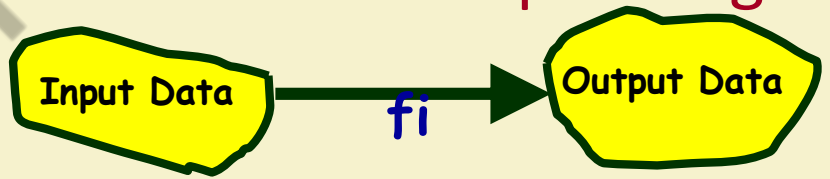
Design Constraints (3.2)

- **3.2 Design Constraints**
- **C-001:**
  - AAS shall provide user interface through standard web browsers.
- **C-002**:
  - AAS shall use an open source RDBMS such as Postgres SQL.
- **C-003**:
  - AAS shall be developed using the JAVA programming language

- **3.3 Non-Functional Requirements**
- **N-001:**
  - AAS shall respond to query in less than 5 seconds.
- **N-002**:
  - AAS shall operate with zero down time.
- **N-003**:
  - AAS shall allow upto 100 users to remotely connect to the system.
- **N-004**:
  - The system will be accompanied by a well-written user manual.

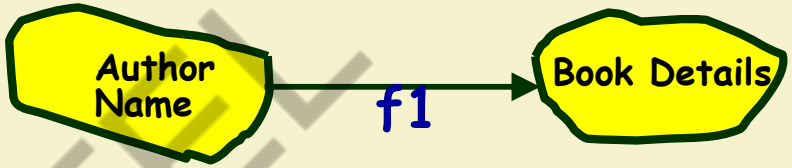
Functional Requirements

- It is desirable to consider every system as:
  - Performing a set of functions {fi}.
- Each function fi considered as:
  - Transforming a set of input data to corresponding output data.



Example: Functional Requirement

- F1: Search Book
  - Input:
    - an author's name:
  - Output:
    - details of the author's books and the locations of these books in the library.



- Functional requirements describe:
  - A set of high-level requirements
  - Each high-level requirement:
    - takes in some data from the user
    - outputs some data to the user
  - Each high-level requirement:
    - might consist of a set of identifiable sub-functions

Functional Requirements

Functional Requirements

- For each high-level requirement:
  - A function is described in terms of:
    - Input data set
    - Output data set
    - Processing required to obtain the output data set from the input data set.

Is it a Functional Requirement?

- A high-level function is one:
  - Using which the user can get some useful piece of work done.
- Can the receipt printing work during withdrawal of money from an ATM:
  - Be called a functional requirement?
- A high-level requirement typically involves:
  - Accepting some data from the user,
  - Transforming it to the required response, and then
  - Outputting the system response to the user.

Use Cases

- A use case is a term in UML:
  - Represents a high level functional requirement.
- Use case representation is more well-defined and has agreed documentation:
  - Compared to a high-level functional requirement and its documentation
  - Therefore many organizations document the functional requirements in terms of use cases

Example Functional Requirements

- Req. 1:
  - Once user selects the “search” option,
    - he is asked to enter the key words.
  - The system should output details of all books
    - whose title or author name matches any of the key words entered.
    - Details include: Title, Author Name, Publisher name, Year of Publication, ISBN Number, Catalog Number, Location in the Library.

Example Functional Requirements

- Req. 2:
  - When the “renew” option is selected,
    - The user is asked to enter his membership number and password.
  - After password validation,
    - The list of the books borrowed by him are displayed.
  - The user can renew any of the books:
    - By clicking in the corresponding renew box.

High-Level Function: A Closer View

- A high-level function:
  - Usually involves a series of interactions between the system and one or more users.
- Even for the same high-level function,
  - There can be different interaction sequences (or scenarios)
  - Due to users selecting different options or entering different data items.

Examples of Bad SRS Documents

- Unstructured Specifications:
  - **Narrative essay --- one of the worst types of specification document:**
    - Difficult to change,
    - Difficult to be precise,
    - Difficult to be unambiguous,
    - Scope for contradictions, etc.

Examples of Bad SRS Documents

- Noise:
  - Presence of text containing information irrelevant to the problem.
- Silence:
  - Aspects important to proper solution of the problem are omitted.

Examples of Bad SRS Documents

- **Overspecification:**
  - Addressing “how to” aspects
  - For example, “Library member names should be stored in a sorted descending order”
  - Overspecification restricts the solution space for the designer.
- **Contradictions:**
  - Contradictions might arise
    - if the same thing described at several places in different ways.

Examples of Bad SRS Documents

- **Ambiguity:**
  - Literary expressions
  - Unquantifiable aspects, e.g. “good user interface”
- **Forward References:**
  - References to aspects of problem
    - defined only later on in the text.
- **Wishful Thinking:**
  - Descriptions of aspects
    - for which realistic solutions will be hard to find.

Suggestions for Writing Good Quality Requirements

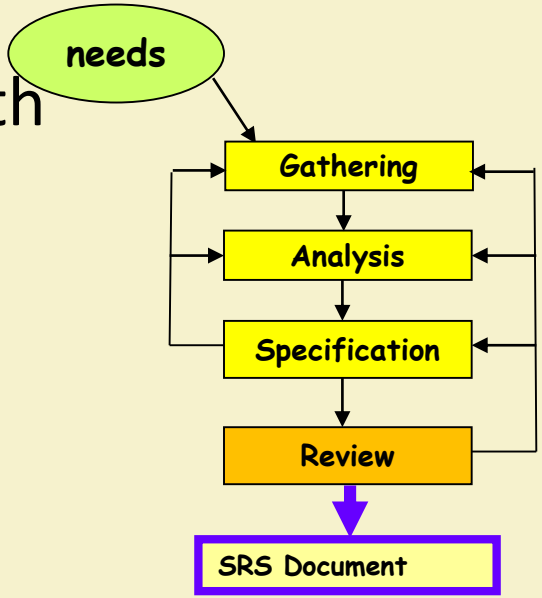
- Keep sentences and paragraphs short.
- Use active voice.
- Use proper grammar, spelling, and punctuation.
- Use terms consistently and define them in a glossary.
- To see if a requirement statement is sufficiently well defined,
  - Read it from the developer's perspective

Suggestions for Writing Good Quality Requirements

- Split a requirement into multiple sub-requirements:
  - Because each will require separate test cases and because each should be separately traceable.
  - If several requirements are strung together in a paragraph, it is easy to overlook one during construction or testing.

SRS Review

- Review done by the Developers along with the user representatives.
- To verify that SRS confirms to the actual user requirements
- To detect defects early and correct them.
- Review typically done using standard inspection process:
  - Checklists.

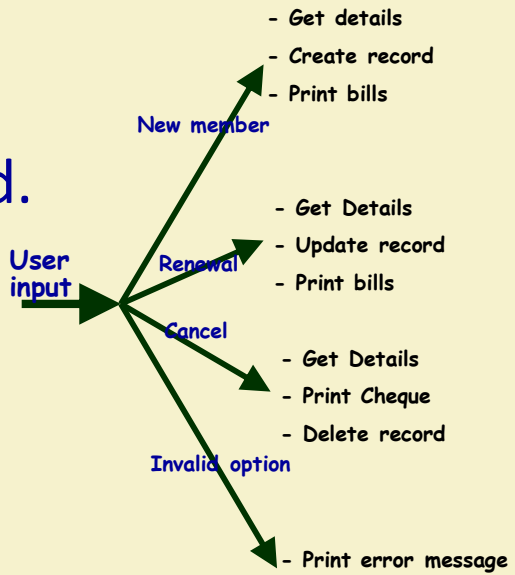


Representation of complex processing logic

- Decision trees
- Decision tables

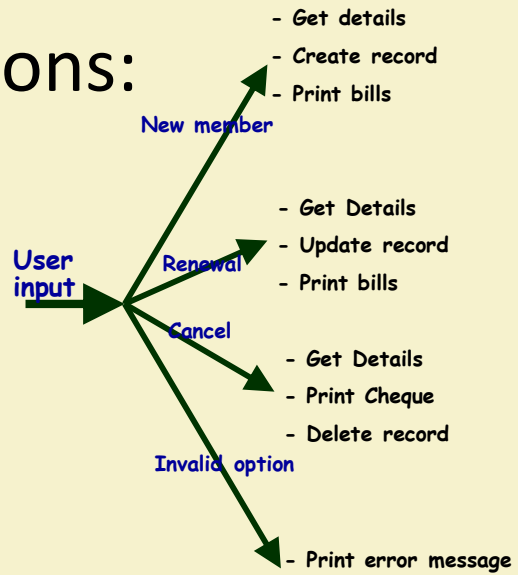
Decision Trees

- Decision trees:
  - Edges of a decision tree represent conditions
  - Leaf nodes represent actions to be performed.
- A decision tree gives a graphic view of:
  - Logic involved in decision making
  - Corresponding actions taken.



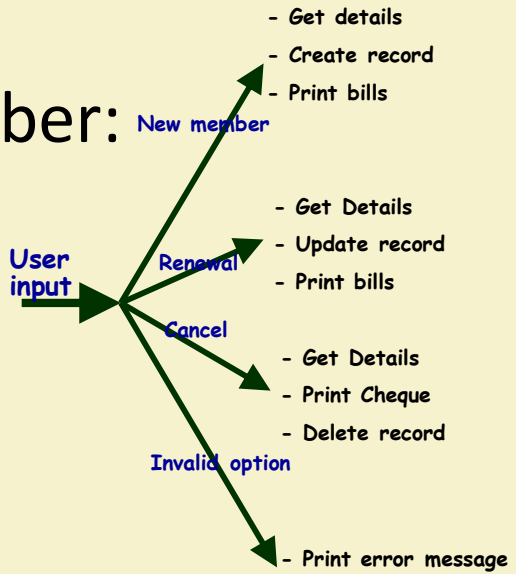
Example: LMS

- A Library Membership automation Software (LMS) should support the following three options:
  - New member,
  - Renewal,
  - Cancel membership.



Example: LMS

- When the new member option is selected,
  - The software asks details about the member:
    - name,
    - address,
    - phone number, etc.



Example(cont.)

- If proper information is entered,
  - A membership record for the member is created
  - A bill is printed for the annual membership charge plus the security deposit payable.

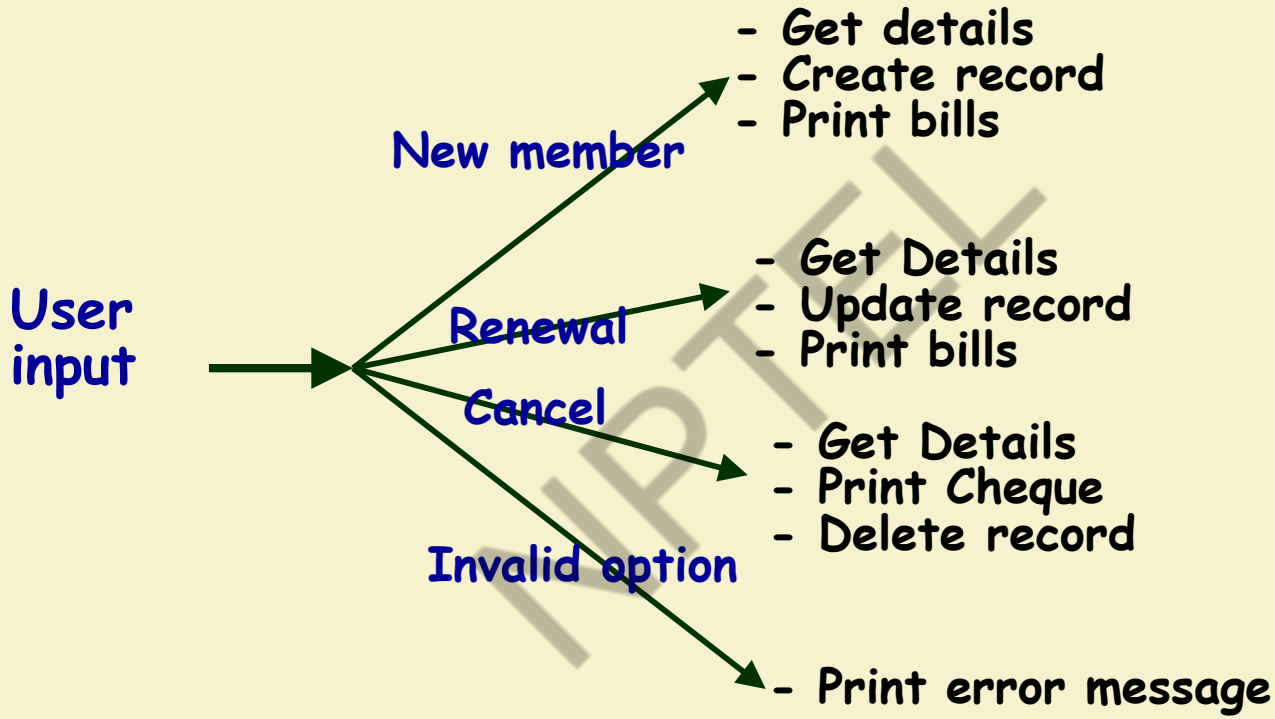
Example(cont.)

- If the renewal option is chosen,
  - LMS asks the member's name and his membership number
    - checks whether he is a valid member.
  - If the name represents a valid member,
    - the membership expiry date is updated and the annual membership bill is printed,
    - otherwise an error message is displayed.

Example(cont.)

- If the cancel membership option is selected and the name of a valid member is entered,
  - The membership is cancelled,
  - A cheque for the balance amount due to the member is printed
  - The membership record is deleted.

Decision Tree



Decision Table

- Decision tables specify:
  - Which variables are to be tested
  - What actions are to be taken if the conditions are true,
  - The order in which decision making is performed.

Decision Table

- A decision table shows in a tabular form:
  - Processing logic and corresponding actions
- Upper rows of the table specify:
  - The variables or conditions to be evaluated
- Lower rows specify:
  - The actions to be taken when the corresponding conditions are satisfied.

Decision Table

- In technical terminology,
  - A column of the table is called a rule.
  - A rule implies:
    - If a condition is true, then execute the corresponding action.

Conditions				
Valid selection	NO	YES	YES	YES
New member	--	YES	NO	NO
Renewal	--	NO	YES	NO
Cancellation		--	NO	NO YES
Actions				
Display error message				
Ask member's name etc.				
Build customer record	--	--		--
Generate bill	--			--
Ask membership details		--		
Update expiry date	--	--	--	
Print cheque		--	--	--
Delete record	--	--	--	

Example

Exercise

- **Develop decision table for the following:**
  - **If the flight is more than half-full and ticket cost is more than Rs. 3000, free meals are served unless it is a domestic flight. The meals are charged on all domestic flights.**

Thank You!!