Software Requirements Modeling and Specifications

SWT 41032

Topic 01

Organization of this Lecture

- Introduction
- Requirements analysis
  - Gathering, Analysis & Specification
  - SRS Documents
  - Decision Trees & Tables

Requirements Phase

- Projects often fail because design begins too early in the lifecycle:
  - without understanding the **problem**, it is impossible to develop a good **solution**.
- It is important to learn:
  - requirements analysis and specification techniques thoroughly.

Requirements Phase : Aims

- To elucidate(to identify) and understand exactly what the customer needs (the requirements):
  - this may not be what they ask for!
- To eliminate errors from the requirements.
- To ensure the requirements are complete.
- To document the results in a Software Requirements Specification:
  - the SRS document.

Software Runaways

- Problems arise with requirements when…
- There are too many-huge projects fail far more often than small ones.
- They are unstable-the customer cannot decide what problem they really want solved.
- They are ambiguous-it not possible to determine what the requirements mean.
- They are incomplete-there is insufficient information to allow a system to be built.

Requirements Phase

- Pressman (2000) divides the requirements phase into five areas of effort:
  - problem recognition;
  - evaluation and synthesis;
  - modeling;
  - specification;
  - review.
- Throughout evaluation and synthesis, the emphasis is on WHAT, not HOW!

Requirements Phase

- The requirements phase involves:
  - Requirements **Gathering & Analysis**;
  - Requirements **Specification**.
- **Explicit Requirements** – these are clearly stated by the customer in documents and/or discussions.
- **Implicit Requirements** – these are so 'obvious' that the customer does not even mention!
- **Exciting Requirements** – these go beyond the expectations of the customer.

Requirements Gathering

- Requirements **gathering** involves:
  - observation of existing systems and procedures;
  - interaction with the customer and/or end-users.
- Systems Analysts / Architects often specialize in a single domain:
  - e.g. medical or military applications.
- Interacting with customers requires:
  - good communication skills;
  - years of experience!

Requirements Analysis

- Requirements **analysis** involves:
  - obtaining a clear, in-depth understanding of the problem that is to be solved;
  - detecting and eliminating errors from the customer perception of the requirements.
- Problems are resolved through repeated interaction with the customer:
  - ambiguity;(clear less)
  - inconsistency;
  - incompleteness.

Ambiguous & Inconsistent requirements

- **Ambiguity:** arises when requirements are difficult to define or quantify:
  - often due to the use of vague or subjective terms – e.g. some, most, good, reliable, fast;
  - how would you validate a requirement such as “the system should be user friendly”?
- **Inconsistency** arises when one requirement seems to contradict another:
  - when temp. > 100 °C turn on water shower;
  - when temp. > 100 °C turn on cooling system.

Incomplete requirements

- **Incompleteness** arises when a requirement is accidentally left out:
  - sometimes analysts make mistakes;
  - sometimes an implicit requirement is so ‘obvious’ that the customer does not bother to mention it!
- A simple example:
  - the analyst did not record what should happen when the temperature falls back below 100 °C;
  - Presumably (suppose) the water shower or cooling system should be turned off?

Requirements Specification

- Requirements **specification** involves:
  - organizing the requirements into a Software Requirements Specification (SRS) document.
- The SRS document focuses on **what** the system must do (i.e. the problem)...
  - but avoids saying **how** (i.e. the solution).
- The SRS document is useful in various ways:
  - a statement of customer needs;
  - a specification for design, implementation, testing;
  - a reference document for maintenance.

SRS Document

- The SRS is a **legal contract** between the developer and the customer:
  - it can be used to settle disagreements;
  - the final product must be accepted by the customer if it matches the SRS document.
- The SRS is written using customer terminology:
  - complete, concise, consistent & unambiguous;
  - easy to read and understand;
  - easy to modify and maintain;
  - traceable.
- Most importantly, the SRS must be **verifiable** during acceptance testing!

SRS Document : organization

1. Introduction:
   - background; system environment.
2. **Functional Requirements**:
   - a numbered list of functions, with inputs & outputs.
3. **Non-Functional Requirements**:
   - system characteristics;
   - external interfaces (e.g. **user interface**).
4. **Constraints**.
5. Verification (Acceptance) Criteria.

Functional Requirements:

- Functional requirements describe a software system as a set of functions {f()}.
- Each function f() transforms a domain of inputs to a corresponding range of outputs:

Input Data ——— f() ———→ Output Data

- Each function is described in terms of:
  - the **input** from and **output** to the user;
  - the **processing** required.

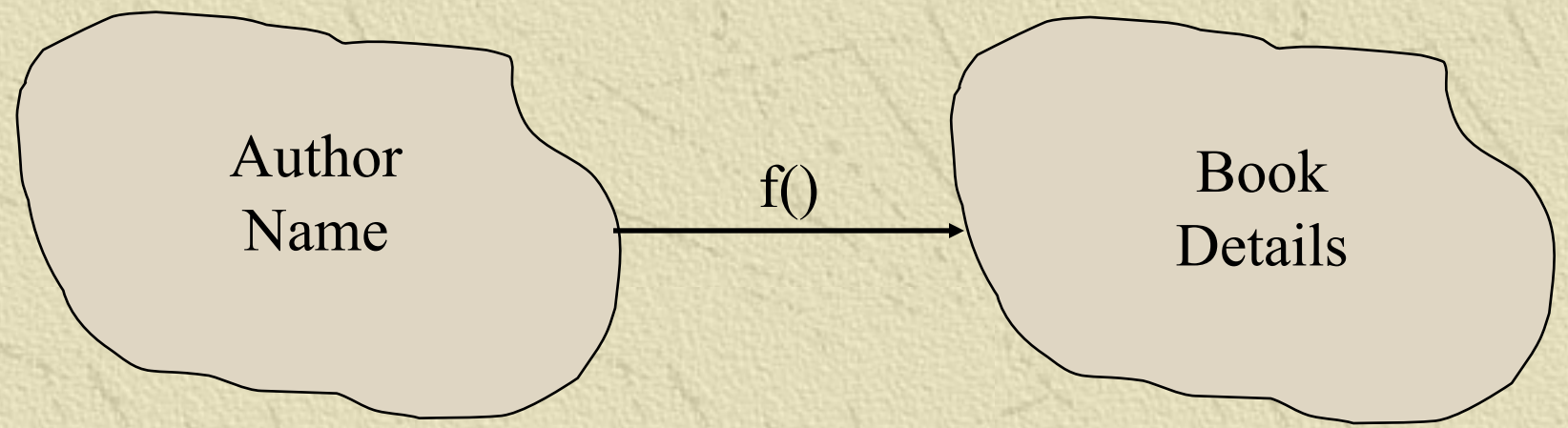
Functional Requirements: A black box specification

- The functional requirements are a “black box” specification of the system:
  - only externally visible behaviour (i.e. input and output) is documented;
  - the internal details of the system are not known.

- In short, the functional requirements describe **what** the system must do, but **not how**!

Functional Requirements

- Example: F1 – Find Books
  - **Input:** the name of an author.
  - **Output:** details of any books by the author –
    - title, publisher, ISBN, etc.
  - **Processing:** search library catalogue for books by the specified author.



Non Functional Requirements

- These are characteristics of the system that cannot be expressed as functions...
  - security – e.g. internet systems;
  - reliability – e.g. safety-critical systems;
  - performance – e.g. real-time systems;
  - usability (human-computer interface);
  - maintainability and portability.
- Non-functional requirements are within the control of the developers – use metrics!

Constraints

- Most systems must satisfy constraints that are beyond the control of the developers…
- Management constraints:
  - e.g. budget, schedule, and resource limits.
- Standards compliance:
  - e.g. ANSI C/C++, and many others.
- Compatibility:
  - e.g. hardware, operating system, or DBMS.
- Interfaces with other systems.

SRS Documents: Things to avoid

- **Narrative Prose**: rarely complete, concise, consistent, or unambiguous; difficult to maintain.
- **Forward References:** mentioning aspects of a problem that are not defined until later.
- **Silence:** absence of information that is relevant to the problem (incompleteness).
- **Noise:** presence of information that is not relevant.
- **Over-Specification:** specifying **how** limits the solution space for the designers.
- **Wishful Thinking:** requirements for which realistic solutions will be difficult to design.

Example:
A Narrative Specification

- 1. When the system is used to **Find Books**, the user is asked for **one or more keyword**. The system displays **details of any books** with titles containing the keyword(s). Book details include the Author, Title, Publisher, Year of Publication, and ISBN.
- 2. When the system is used to **Renew Loans**, the user is asked to provide a **membership number**. After membership validation the system lists details of any **books currently on loan** to the member. The user may choose to renew one or more book loans, after which they are informed of the **new due date** for each book loan that is successfully renewed.

Example: Functional Requirements

- **F1: Find Books**
- **Input:** keyword(s).
- **Output:** details of any books found.
- **Processing:** search for books with titles containing the keyword(s).
- **Definition:** book details include the Author, Title, Publisher, Year of Publication, ISBN.

Example: Functional Requirements

- **F2: View Loans**
- **Input:** a membership number.
- **Output:** current book loan details.
- **Processing:** validate the membership number and retrieve book loan details.
- **Error:** invalid membership number.

Example: Functional Requirements

- **F3: Renew Loans**
- **Input:** to be determined!
- **Output:** new due date(s).
- **Processing:** update the due date for each book that is renewed (if any).
- **Error:** one or more books has been recalled.

Try this Question?

- Identify the functional requirements for software to manage the scheme described below. Specify these requirements in terms of input data, output data, and processing (i.e. specify what, not how).

- A supermarket has devised a scheme to encourage customer loyalty. To register for the scheme, a customer must supply their name, address, and telephone number. Each registered customer is assigned a unique Customer Identification Number (CIN). A customer can present this CIN to the checkout staff so that the value of a purchase is recorded. At the end of every year the supermarket intends to award surprise gifts to the three customers who spent the most money in the supermarket; it also intends to award a $10 shopping voucher to every customer who spent more than a threshold amount (e.g. $1000) during the year. The total against each CIN is reset on the first day of every year, when the supermarket is closed for the public holiday.

Representation of Complex Processing Logic

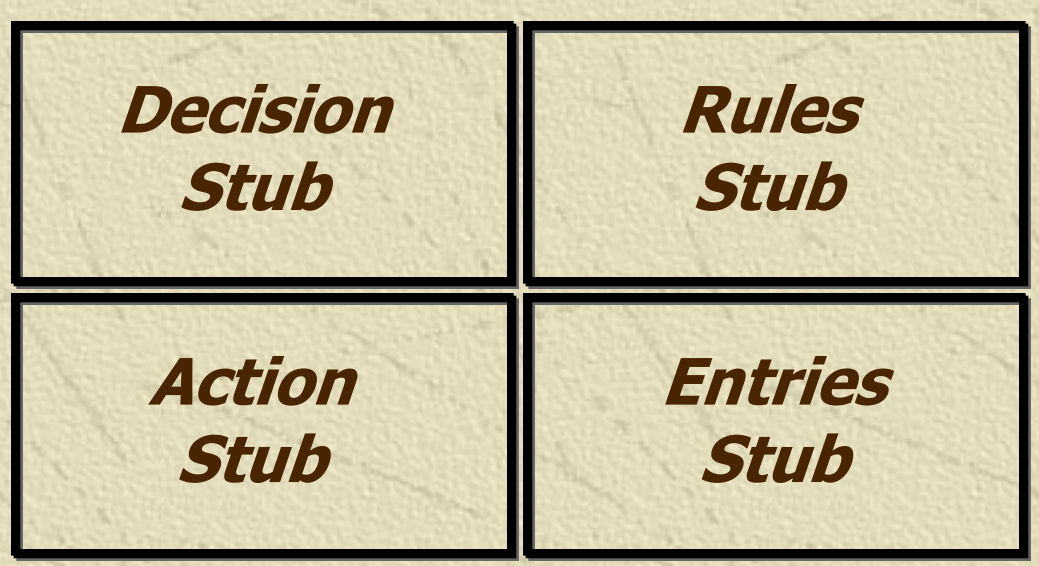
- Decision tables
- Decision trees

Decision Tables

- A decision table shows processing logic in the form of conditions and actions:
  - upper rows specify the **conditions** to be tested – these can be true (Y), false (N), or irrelevant (–);
  - lower rows specify the **actions** to be taken when corresponding conditions are true or false.
- A column of a decision table is a rule:
  - if the given set of conditions is satisfied, then execute the corresponding action.

Decision Table Layout

- Standard format used for presenting decision tables.



Developing Decision Tables

- Process requires the determination of the number of conditions (inputs) that affect the decision.
- The set of possible actions (outputs) must likewise be determined
- The number of rules is computed
- Each rule must specify one or more actions

Number of Rules

- Each condition generally has two possible alternatives (outcomes): ***Yes*** or ***No***
  - In more advanced tables, multiple outcomes for each condition are permitted
- The total number of rules is equal to

$$2^{\text{no. of conditions}}$$

- Thus, if there are four conditions, there will be sixteen possible rules

Building the Table

- For each rule, select the appropriate action and indicate with an ‘X’
- Identify rules that produce the same actions and attempt to combine those rules; for example:
  - *Condition 1 Y Y*
    Condition 2 Y N
    Action 1 X X

    Condition 1 Y
    Condition 2 -
    Action 1 X

Cleaning Things Up

- Check the table for any impossible situations, contradictions, and redundancies and eliminate such rules

- Rewrite the decision table with the most reduced set of rules; rearranging the rule order is permissible if it improves user understanding

Decision Tables

- A decision table shows processing logic in the form of conditions and corresponding actions:
  - which variables are to be tested
  - **Upper rows specify the conditions to be tested – these can be true (Y), false (N), or don't care (-);**
  - Lower rows specify the **actions** to be taken when corresponding conditions are true or false.
- A **column** of a decision table is a **rule**:
  - If a given condition is true (or false), then execute the corresponding action.

An Example

You must decide how to dress for the weather. The conditions are that it may or may not be cloudy and it may or may not be cold. If it is cloudy, take an umbrella. If it is cold, take a coat.

Is it Cloudy?	**Y**	**N**	**Y**	**N**
Is it Cold	**Y**	**Y**	**N**	**N**
Take a coat	**X**	**X**	**-**	
Take an umbrella	**X**	**-**	**X**	

Advantages

- Provide a clear tabular representation linking conditions with actions
- Ensure coverage of all possible cases
- Highlight inconsistencies, redundancies, and ambiguities
- Easy to follow
- Can incorporate in program specs

Disadvantages

- Can become large and unwieldy

Decision Trees

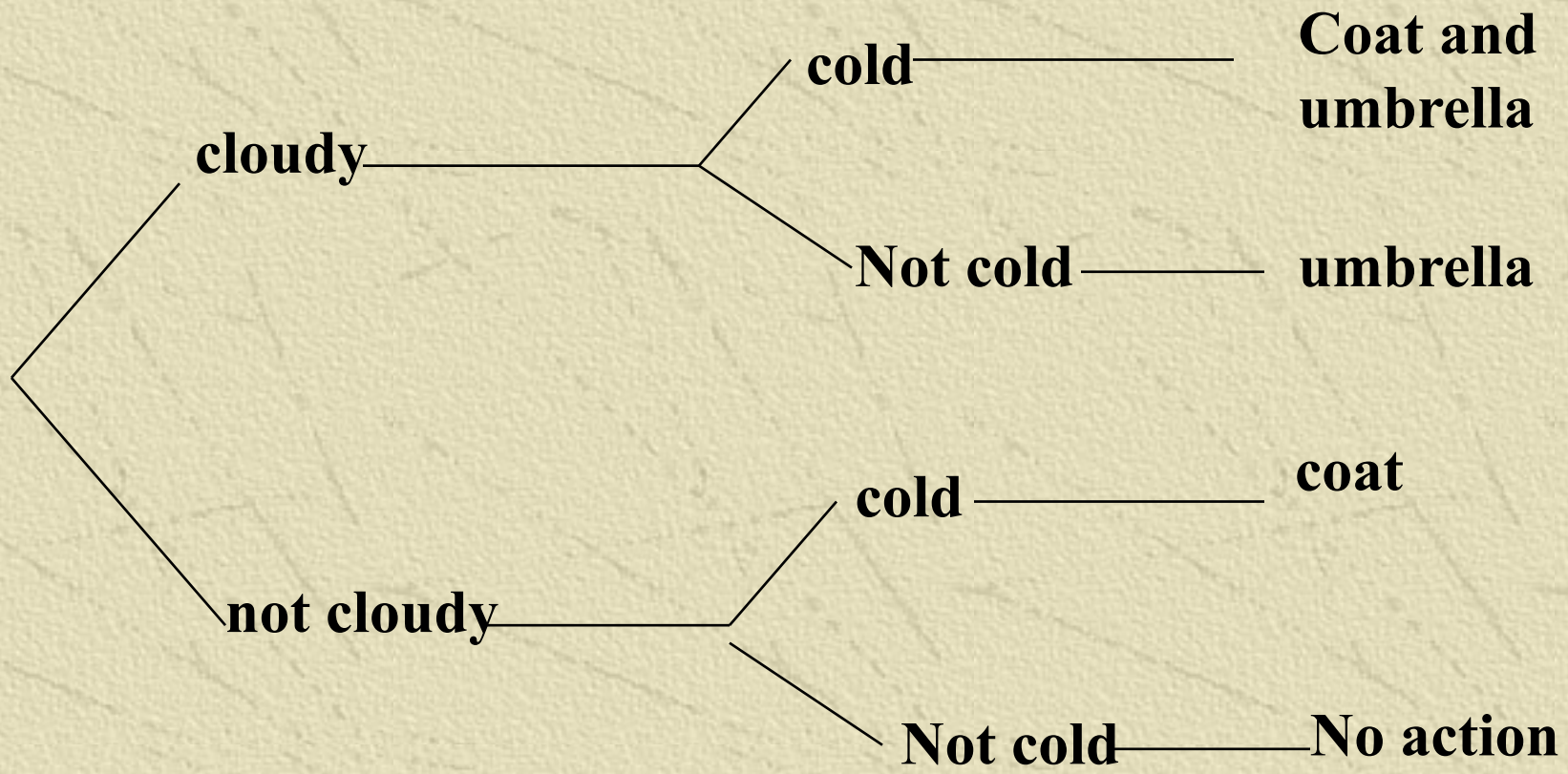
- A decision tree is a graphical view of:
  - the logic involved in decision making – i.e. the order in which decisions are made;
  - the corresponding actions taken.
- Edges (branches) of a decision tree represent **conditions** to be tested.
- Nodes (leaves) of a decision tree represent **actions** to be performed:
  - the choice of actions depends on whether or not the various conditions are satisfied.

Drawing Decision Trees

- First, identify all conditions and actions and the order and timing of these (if they are critical)
- Second, begin building the tree from left to right while making sure you are complete in listing all possible alternatives before moving over to the right

Decision Trees

Graphic representation of a decision table



Decision Tree Advantages

- Three advantages over a decision table
  - The order of checking conditions and executing actions is immediately noticeable
  - Second, conditions and actions of decision trees are found on some branches but not on others
  - Third, compared to decision tables, decision trees are more readily understood by others in the organization

Another Example: LMS

A Library Membership automation Software (LMS) should support the following three options:

- new membership,
- renewal membership,
- cancel membership.

Example: LMS

- **When the new member option is selected,**
  - **the software requests details about the member:**
    - **name,**
    - **address,**
    - **phone number, etc.**
- **If valid 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.**
  - **otherwise an error message is displayed.**

Example: LMS

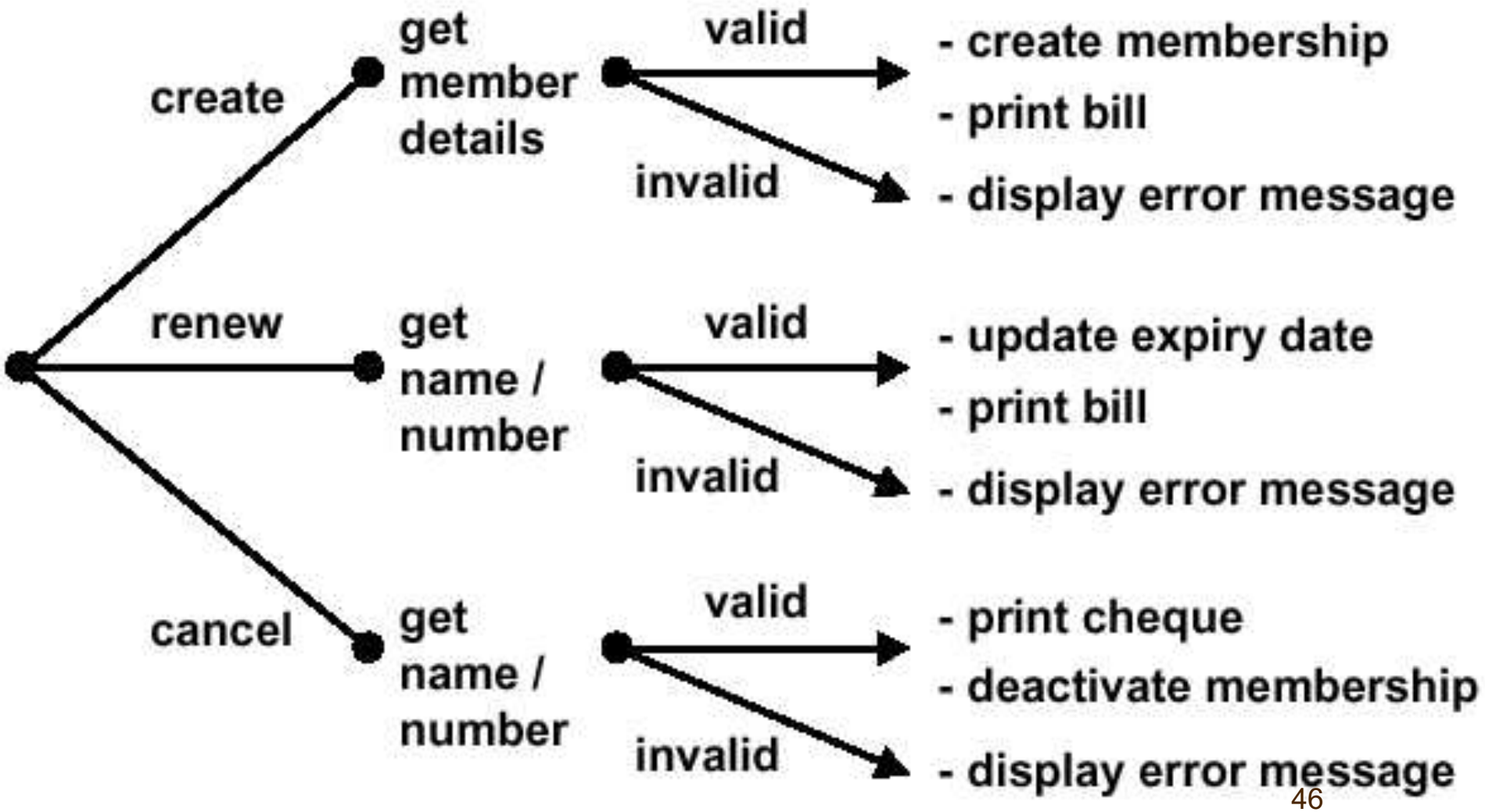
If the renewal option is chosen,

- **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: LMS

- If the cancel membership option is selected, the system:
  - requests the membership name or number, and checks whether this is of a valid member
- If the name or number is valid:
  - the membership is cancelled,
  - a cheque for the balance amount due to the member is printed
  - the membership record is deleted
  - otherwise an error message is displayed.

Example Decision Tree



Example: Decision Table

Conditions	Rules			
Valid selection	NO	YES	YES	YES
New member	--	YES	NO	NO
Renewal	--	NO	YES	NO
Cancellation	--	NO	NO	YES
Actions				
Display error message	×	--	--	--
Get member's details	--	×	×	×
Create membership	--	×	--	--
Print bill	--	×	×	--
Get name & number	--	--	×	×
Update expiry date	--	--	×	--
Print cheque	--	--	--	×
Delete record	--	--	--	×

Comparison: Decision Trees & Decision Tables

- Both decision tables and decision trees
  - can represent complex program logic.
- Decision trees are easier to read and understand
  - when the number of conditions are small.
- Decision tables help to look at every possible combination of conditions.

Summary

- The requirements phase is an important part of the software lifecycle:
  - undetected errors from this phase affect all the phases of development.
- This phase involves three principal activities:
  - requirements gathering;
  - requirements analysis;
  - requirements specification.
- The aim is to give the customer what they need, not what they ask for!

Summary

- Main tasks of analysis and specification:
  - to organize the requirements systematically;
  - to eliminate errors and ensure completeness;
  - to specify the results in an SRS document.
- Main components of an SRS document:
  - functional requirements;
  - non-functional requirements;
  - constraints.

References:

- Essential Reading:
  - Pressman – Chapters 11 & 15;
  - Pfleeger – Chapter 4;
  - Sommerville – Chapters 4, 5 & 15.
- Bedtime Reading:
  - R. L. Glass: 1998. “Software Runaways: lessons learned from massive software project failures.” Prentice-Hall.