Software Engineering

Requirements Analysis and Specification

Requirements Analysis and Specification

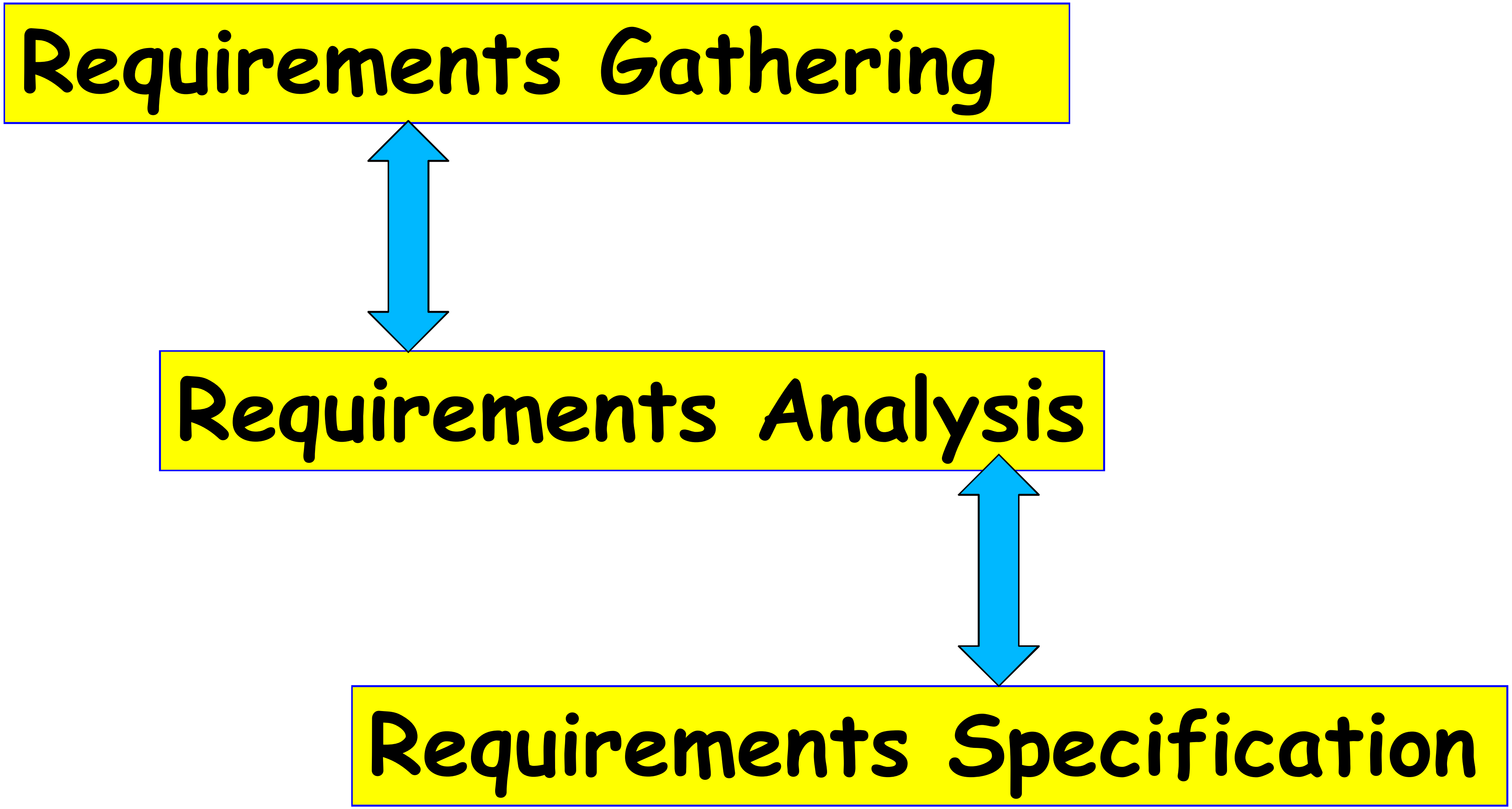
Many projects have failed:

- Because they started to implement the software.
- Without ever determining whether they are building what the customer really wanted.

What are Requirements?

- A Requirement is:
  - **A capability or condition required from the system.**
- What is involved in requirements analysis and specification?
  - Determine what is expected by the client from the system.
  - Document them in the form that is clear to the client as well as to the development team members.

Activities in Requirements Analysis and Specification

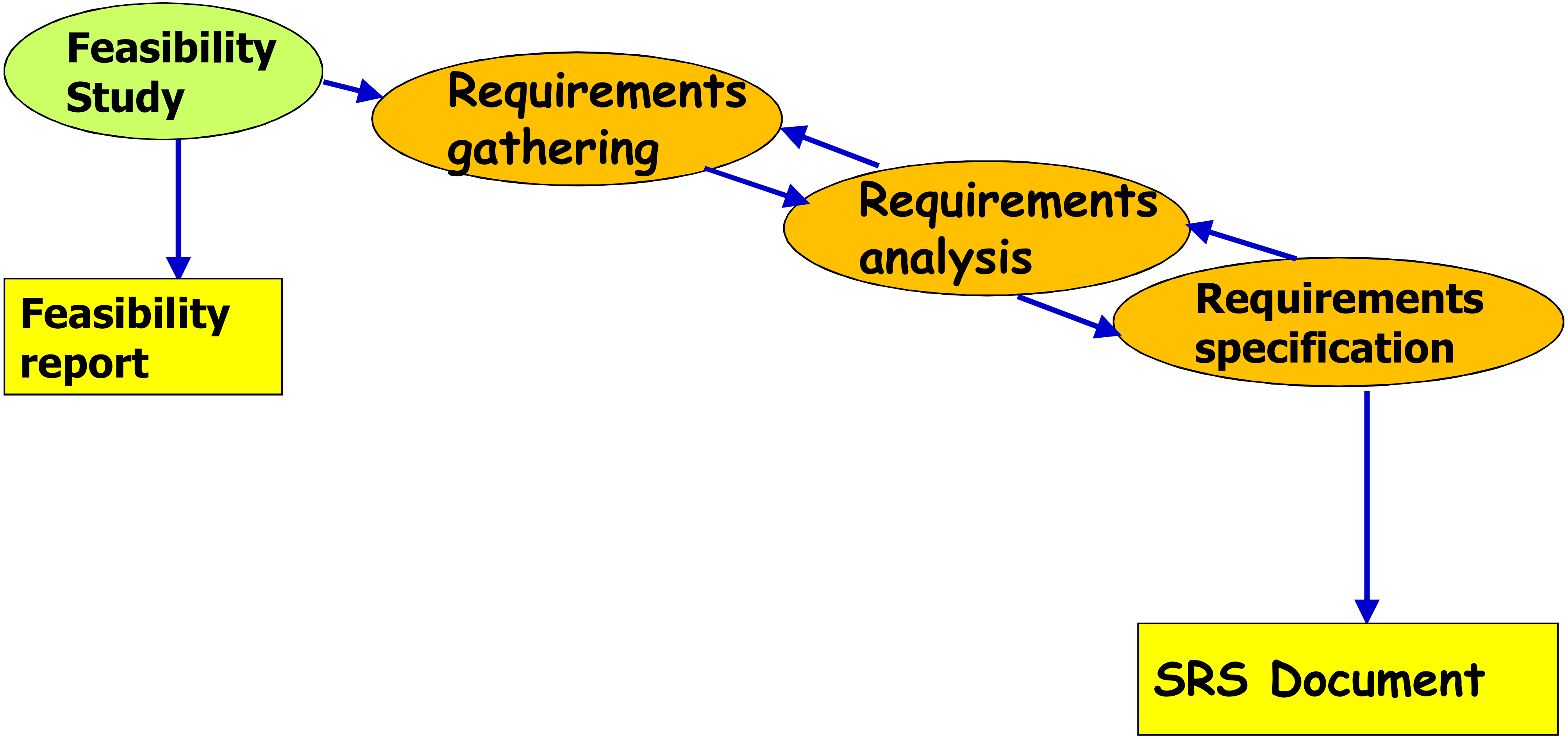


Requirements Gathering

Requirements Analysis

Requirements Specification

Requirements Engineering Process



Why requirements analysis?

- Requirements: capabilities & conditions to which the system and the project must conform

- Goals of this phase: fully understand requirements
  - Gather requirements, constraints
  - Analyze collected data to remove inconsistencies, etc
  - Document requirements into Software Requirements Specification (SRS) document

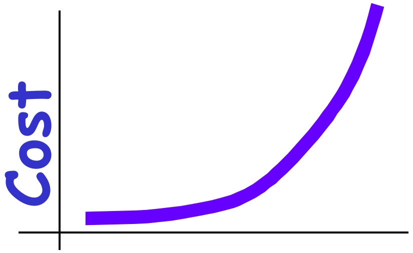
- Done by systems analysts

Need for SRS…

Good SRS reduces development cost

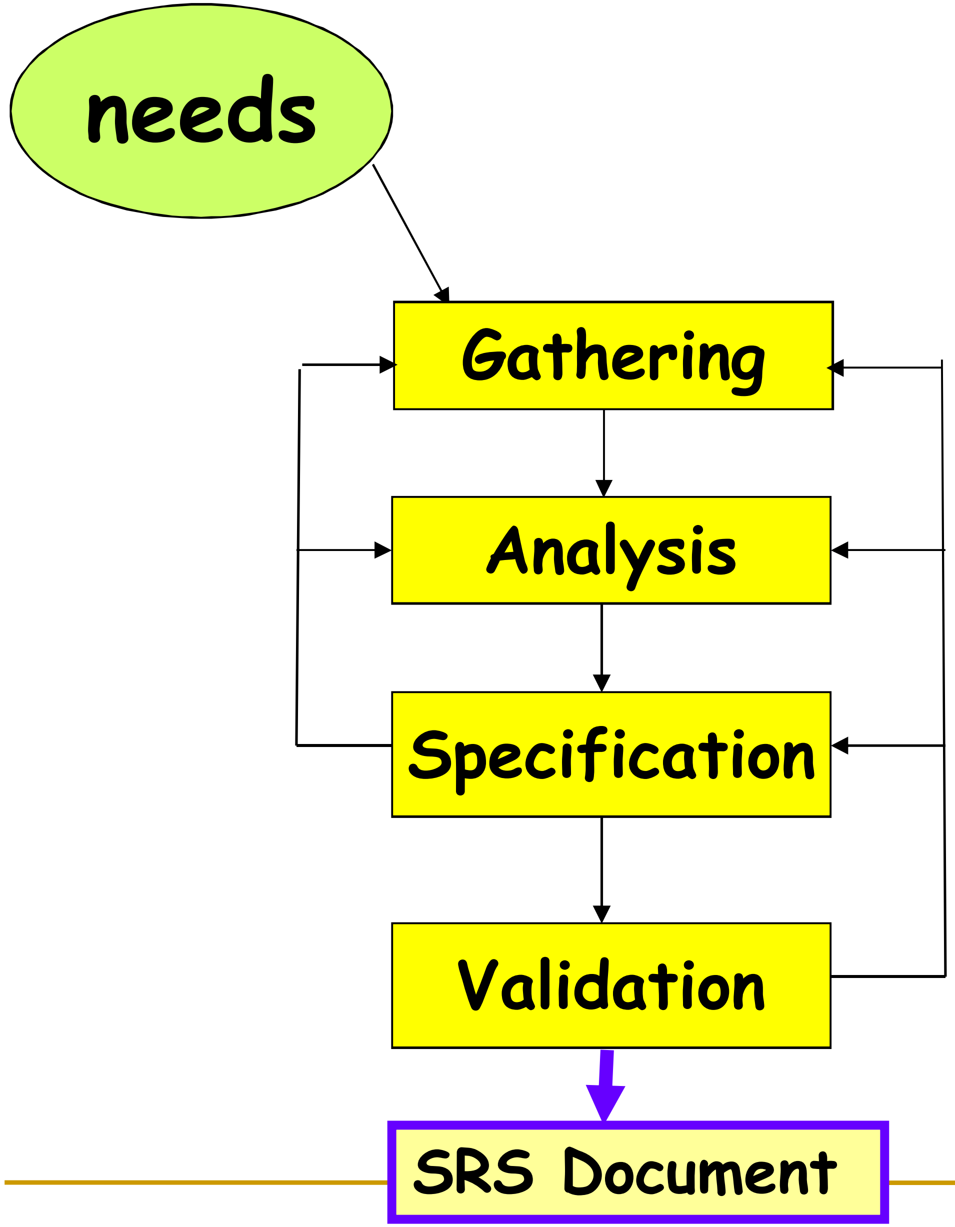
- Req. errors are expensive to fix later
- Req. changes cost a lot (typically 40%)
- Good SRS can minimize changes and errors
- Substantial savings --- effort spent during req. saves multiple times that effort

An Example:



- ❑ Cost of fixing errors in req. , design , coding , acceptance testing and operation are 2 , 5 , 15 , 50 , 150 person-months

Requirement process..



- ✓ Specification itself may lead to more analysis
- ✓ Validation can show gaps that can lead to further analysis and specification .

Requirements gathering

- Analyst gathers requirements through
  - Observing similar existing systems
  - Discussion with customer / end-user / stake-holder
  - BDD: Behavior driven development (*)

- Easier if project is to automate some existing manual activity or a working system
  - Otherwise lot of imagination and creativity are required

- Requirements written down as user-stories
  - Description of how system will be used by stake-holders

Analysis of gathered requirements

- Goal: detect Anomaly, incompleteness, inconsistencies
  - Anomaly : Ambiguity in requirements
  - Inconsistency: some part of the requirements contradicts some other part
  - Incompleteness: some cases overlooked
- Resolved through further discussion
- Consider complexities that may arise while solving the problem

Requirement problem : Anomaly

Example-1 : When the temperature becomes high, the heater should be switched off

Example-2 : During the final attendance computation, if any student has sufficiently low count, then his/her parents will be called

Requirement problem : Inconsistency

- Example :
  - The furnace should be switched-off when the temp of the furnace raises above 500 C
  - When the temperature of the furnace raises above 500C, the water shower should be switched on and furnace should remain on

Requirement problem : Incompleteness

- Example :
  - The temp of the furnace raises above 400 C then an alarm bell must be sounded

– *no provision for resetting the alarm bell after the temp has been brought down in any of the requirements*

Document requirements into SRS

- SRS
  - Written using end-user terminology (reviewed by user)
  - black-box specification of system: only external behavior (input / output) documented

- Importance of SRS
  - Contract document with customer
  - Reference document for design & development team

- Challenge: make SRS understandable to different types of people

Properties of good SRS document

- Concise but not ambiguous or incomplete
- Specify what the system must do, not how to do it
- Well-structured, easy to change
- Traceable
  - **Requirements Traceability Matrix (RTM):** should be possible to trace which part of specification corresponds to which part of design, code, ... (& vice-versa)
- Verifiable
  - e.g. “system should be user friendly” is not verifiable

Contents of SRS document

- Functional requirements
- Non-functional requirements
- Goals of implementation

Functional requirements

- A system considered as a set of high-level functions or requirements

- Each high-level function or requirement
  - Users can do some useful piece of work through this
  - E.g. search for a book in library
  - May involve a series of interactions of system with user
  - Described by specifying input data (from user), processing required and output data
  - May consist of a set of sub-functions / sub-requirements

Example functional requirements

- Example: online library system
  - Requirement 1: search for a book
  - Requirement 2: renew borrowed book

- R1: search for a book
  - When user selects the “search” option
    - He/she is asked to enter the key words
  - System should output details of all books whose title or author name matches any of the key words entered
    - Details: Title, Author Name, Publisher name, Year of Publication, ISBN Number, Catalog Number, Location in the Library

Example functional requirements

- R2: renew borrowed book
  - When the “renew” option is selected
    - User asked to enter his membership id and password
  - After id and password validation
    - list of the books borrowed by him is displayed
  - User can renew any of the books by clicking in the corresponding renew box

R1: search for a book

- R1.1
  - Input: ‘search’ option
  - Output: user prompted to enter key words
- R1.2
  - Input: key words
  - Output: Details of all books whose title or author name matches any of the key words
  - Processing: search books list for the key words

R2: renew borrowed books

- R2.1
  - Input: ‘renew’ option selected
  - Output: user prompted to enter membership id, password
- R2.2
  - Input: membership id and password
  - Output:
    - List of books borrowed by user displayed. User prompted to indicate books to be renewed, or
    - User informed about bad password. Again prompt for id, passwd
  - Processing: password validation; search for books borrowed by user and display

R2: renew borrowed books (contd.)

- R2.3
  - Input: user choice for renewal of the books issued to him
  - Output: confirmation of the books renewed
  - Processing: renew the books selected by user.

Non-functional Requirements

Characteristics of the system which can not be expressed as functions:

- Performance
- Maintainability
- Portability
- Usability
- Security
- Safety, etc.

Non-functional Requirements

- Reliability issues :
- Performance issues :

  Example: How fast the system can produce results

  - so that it does not overload another system to which it supplies data, etc.
  - Needs to be measurable (verifiability)
  - Eg resp time should be xx 90% of the time

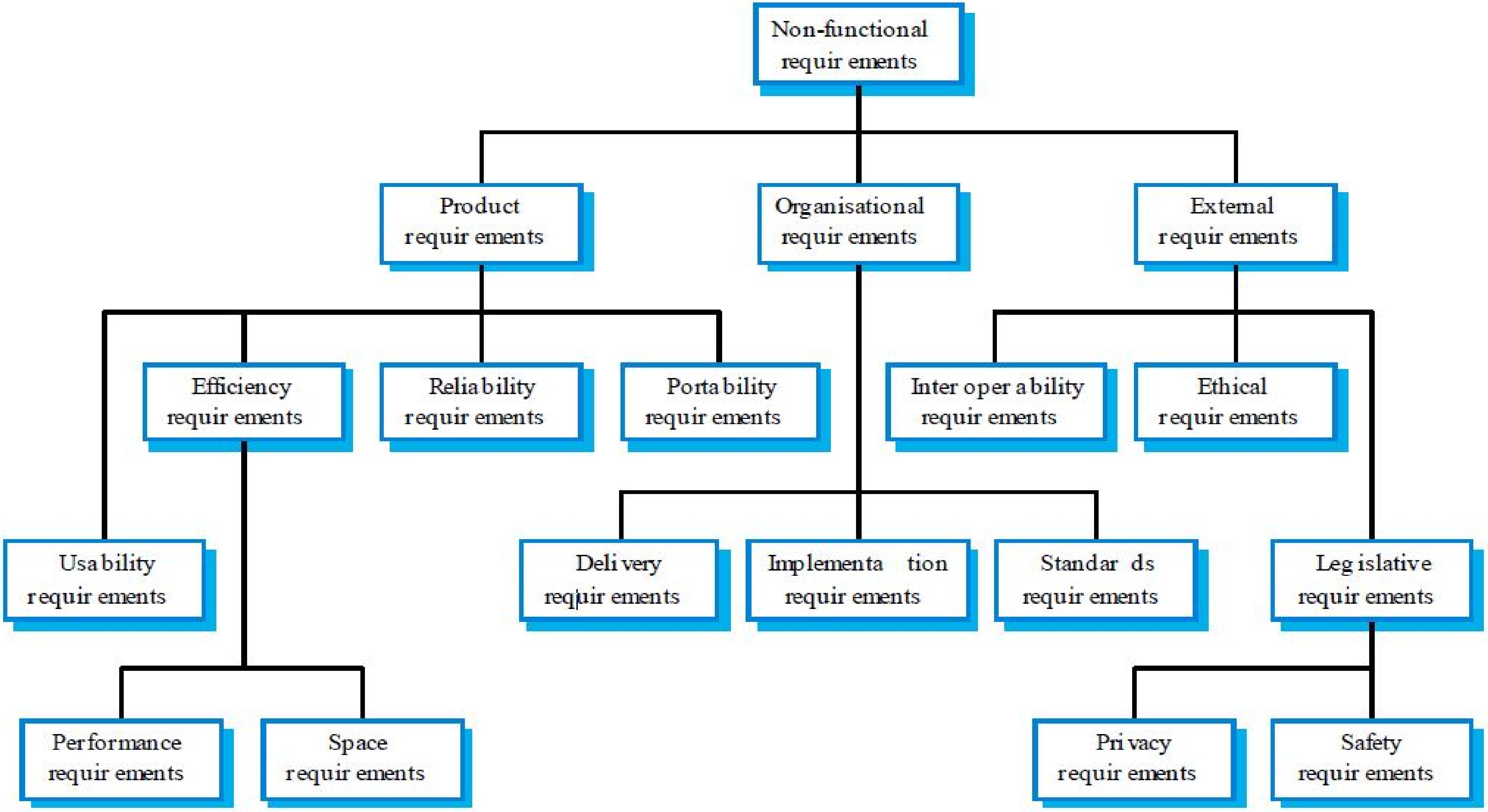
Design and Implementation Constraints

- Hardware to be used,
- Operating system or DBMS to be used
  - ✓ Eg. Oracle DBMS needs to be used as this would facilitate easy interfacing with other applications that are already operational in the organization
- Capabilities of I/O devices
- Standards compliance
- Data representations
  - ✓ by the interfaced system

External Interface Requirements

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

Non-functional requirement classification


Non-functional requir ements
Product requir ements
Organisational requir ements
External requir ements
Efficiency requir ements
Relia bility requir ements
Porta bility requir ements
Inter oper a bility requir ements
Ethical requir ements
Usa bility requir ements
Deli very reqpuir ements
Implementa tion requir ements
Standar ds requir ements
Leg islative requir ements
Performance requir ements
Space requir ements
Pri vacy requir ements
Safety requir ements

Goals of implementation

Features that are desirable in the system, but would not be checked for compliance

- Some general suggestions regarding development
  - These suggestions guide trade-off among design goals
- Portability, Reusability, Maintainability issues
- Scope for functionalities to be developed in future

These are the items which the developers might keep in their mind during development

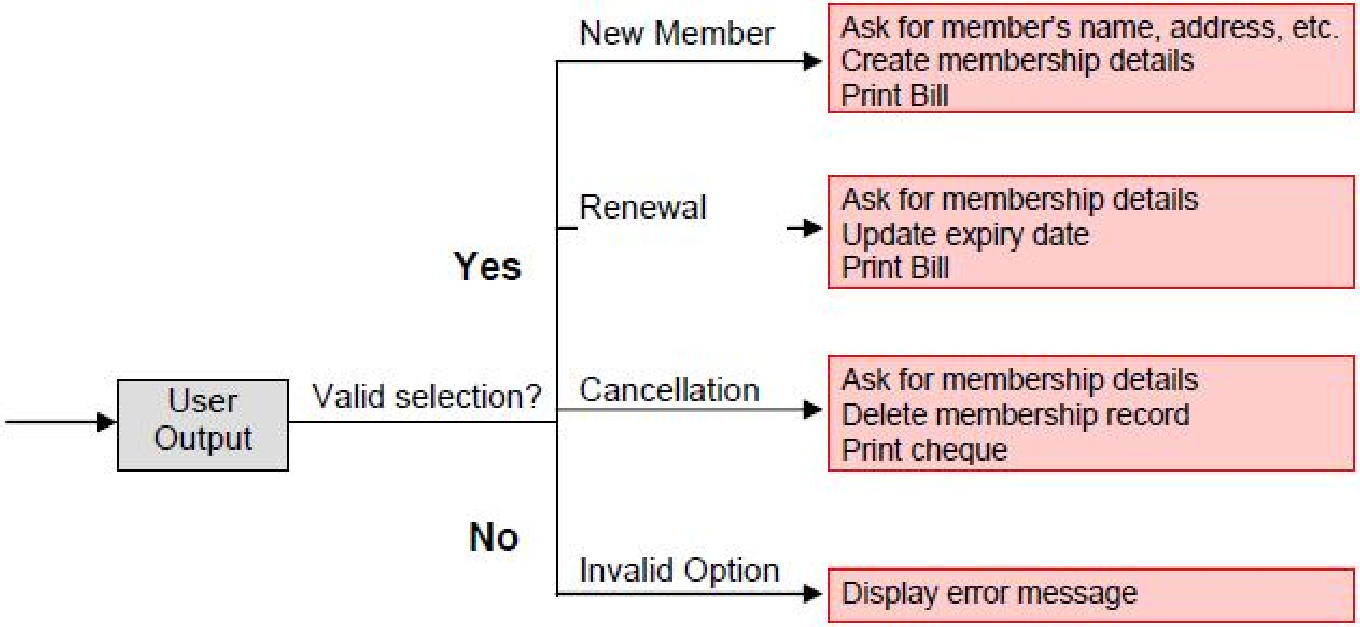
Representing complex processing logic

- **Decision trees** - gives a graphic view of the processing logic involved in decision making and the corresponding actions taken
    - Edges represent conditions
    - Nodes represent actions to be performed

Decision tree example

Consider Library Membership Automation Software where it should support the following three options:

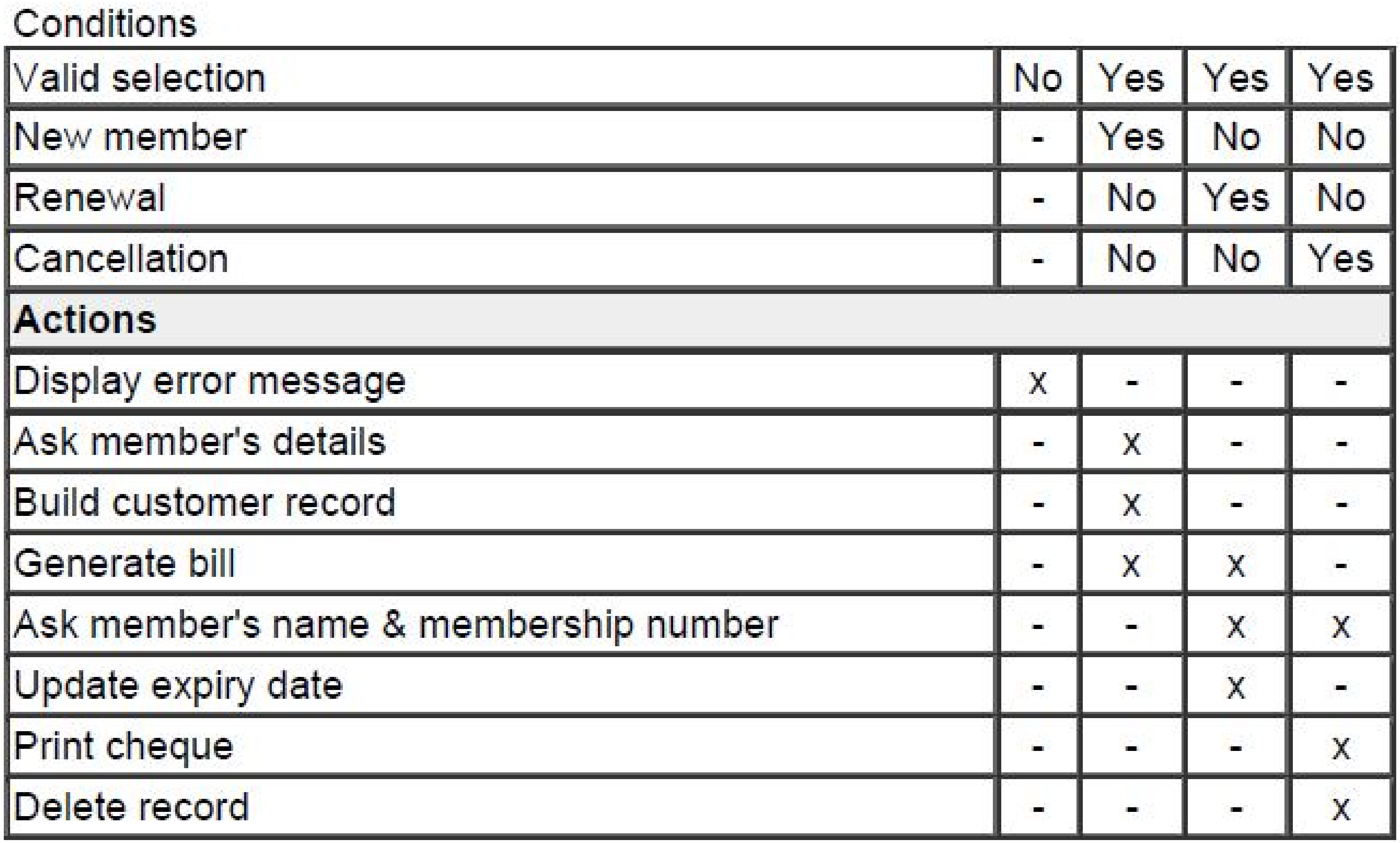
- **New member**
- **Renewal**
- **Cancel membership**



Representing complex processing logic

- **Decision table** - used to represent the complex processing logic in a tabular or a matrix form
  - The upper rows of the table specify the ***conditions*** to be evaluated
  - The lower rows of the table specify the ***actions*** to be taken when the corresponding conditions are satisfied.
  - A column in a table is called a ***rule***. A rule implies that if a condition combination is true, then the corresponding action is to be executed.

Decision Table example



Conditions				
Valid selection	No	Yes	Yes	Yes
New member	-	Yes	No	No
Renewal	-	No	Yes	No
Cancellation	-	No	No	Yes
Actions				
Display error message	x	-	-	-
Ask member's details	-	x	-	-
Build customer record	-	x	-	-
Generate bill	-	x	x	-
Ask member's name & membership number	-	-	x	x
Update expiry date	-	-	x	-
Print cheque	-	-	-	x
Delete record	-	-	-	x

Decision Table Vs Decision Tree

- 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.

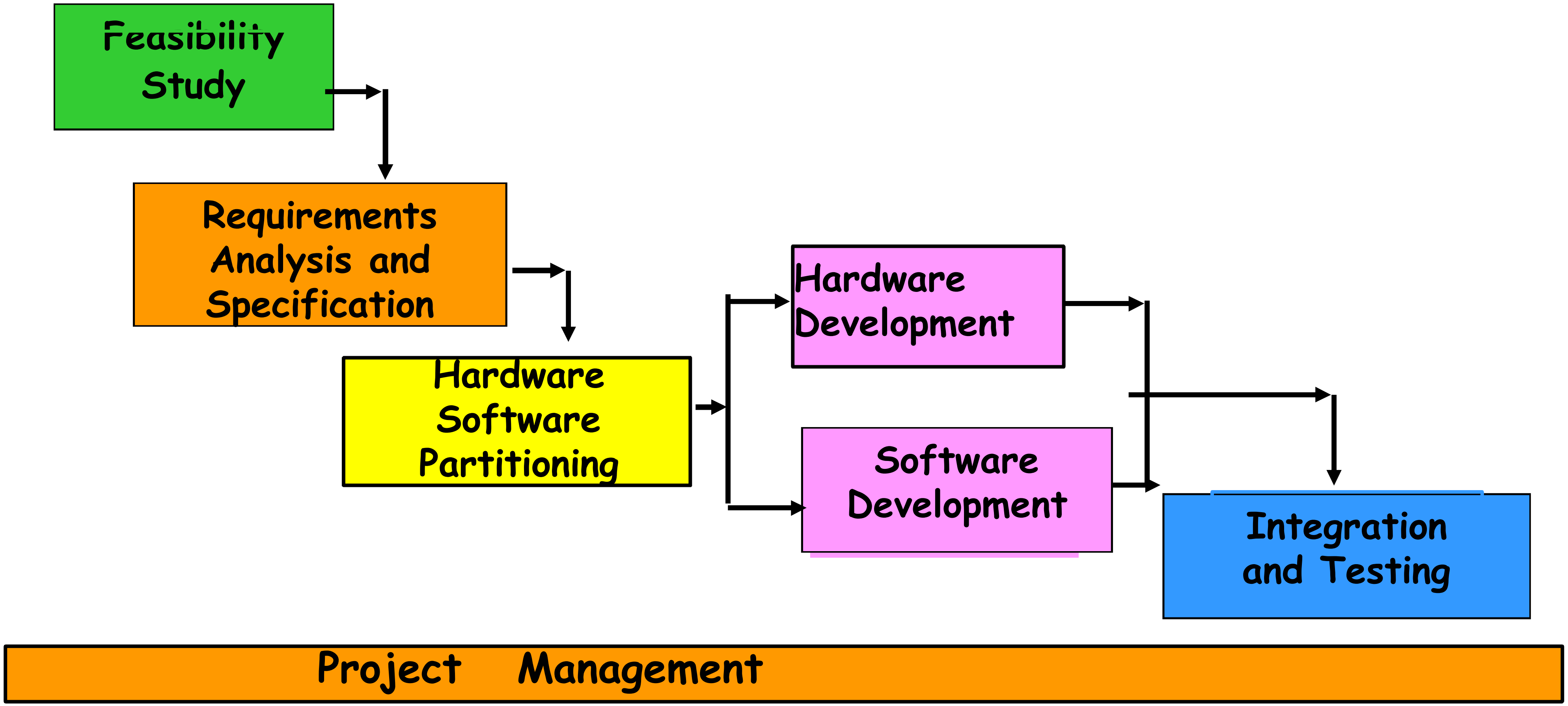
COMPUTER SYSTEMS ENGINEERING

Computer Systems Engineering

- Encompasses software engineering

- Many products require software as well as specific hardware to run the software
  - High-level problem: decide which tasks are to be solved by software, which by hardware
  - Often hardware and software developed together
  - Hardware simulator used to test software
  - Integration of hardware and software
  - Final system testing

Computer Systems Engineering



Feasibility Study
Requirements Analysis and Specification
Hardware Software Partitioning
Hardware Development
Software Development
Integration and Testing
Project Management