
UNIT-II

Requirement Gathering and Analysis

Requirements Engineering

- **Requirement:** A function, constraint or other property that the system must provide to fill the needs of the system's intended user(s).
 - **Engineering:** implies that systematic and repeatable techniques should be used
 - **Requirement Engineering** means that requirements for a product are defined, managed and tested systematically
-

Requirements Engineering

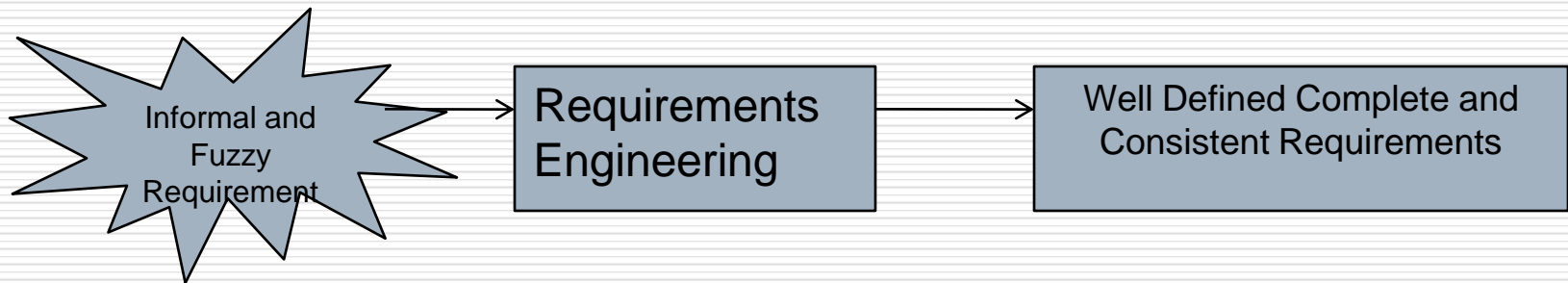
The basic goal of requirements phase in the SDLC is to produce the Requirements Specification Document(RSD) or Software Requirement Specification (SRS) which describes the complete external behavior of the proposed software system. The process of developing this document is Requirements Engineering.

Requirements Engineering *can be defined as*

- "The systematic process of documenting requirements through an interactive co-operative process of analyzing the problem, documenting the resulting observations in a variety of representation format and checking the accuracy of the understanding gained."*
 - "The systematic use of proven principles, techniques, tools and languages for the cost effective analysis, documentation and ongoing evolution of user needs and the specification of the external behavior of the system in order to satisfy these needs."*
-

Requirements Engineering

Input to this activity of requirements engineering are requirements which are informal and fuzzy and output is clear, well defined and consistent requirements written in formal notation RSD or SRS as shown -



Requirements Engineering (RE)

- ❑ It is essential that the software engineering team understand the requirements of a problem before the team tries to solve the problem.
 - ❑ RE is software engineering actions that start with communication activity and continues into the modeling activity.
 - ❑ RE establishes a solid base for design and construction. Without it, resulting software has a high probability of not meeting customer needs.
-

Requirements Engineering (RE)

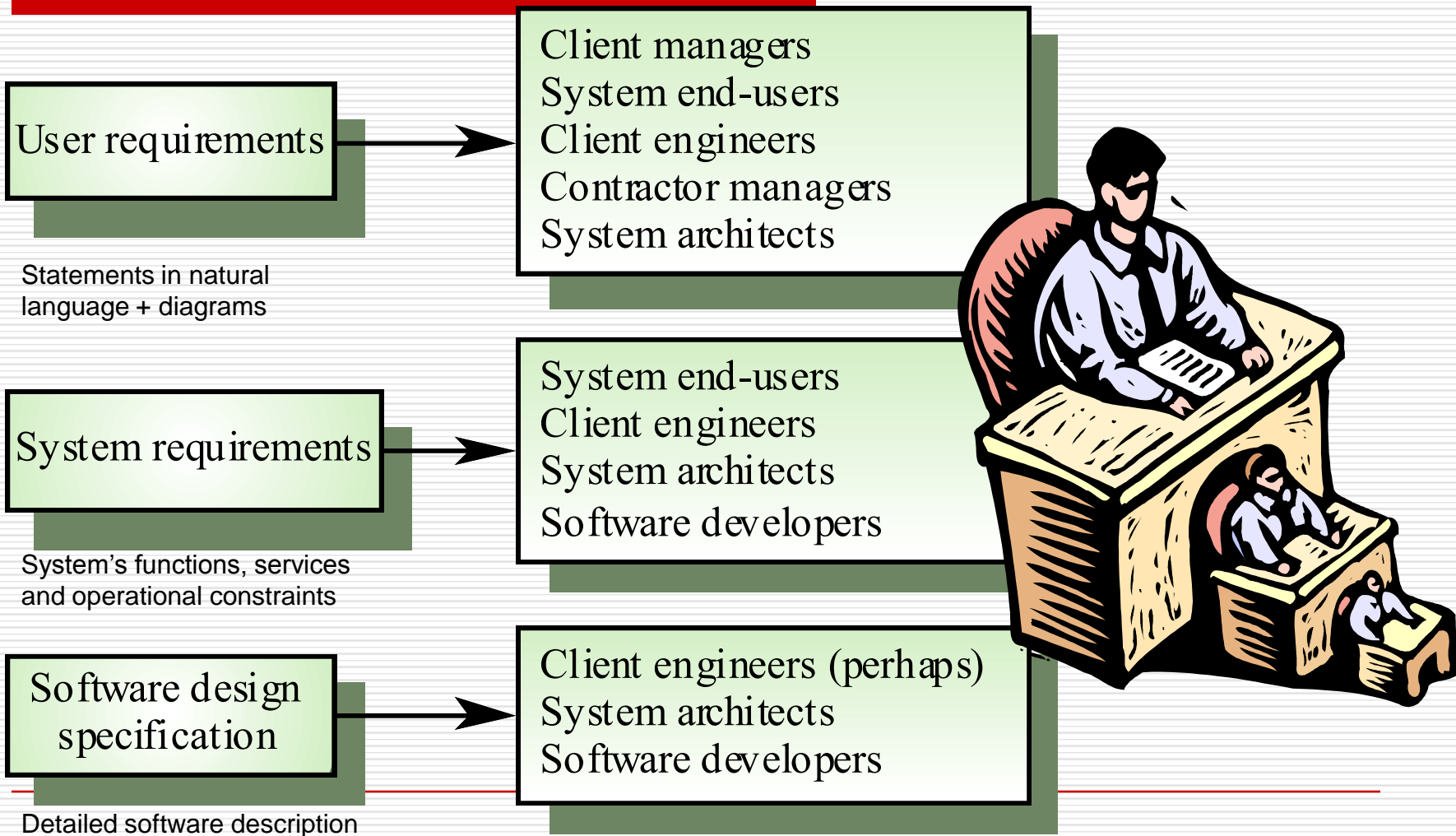
The requirements engineering is the most crucial and most important activity in the software development life cycle because the errors introduced at this stage are the most expensive errors requiring lot of rework if detected late in the software life cycle.

*The **SRS** document produced as an output of this activity is used by successive stages of life cycle i.e. for generating design document for coding, for generating test cases in software testing and also plays a lead role in acceptance testing.*

Characteristics of a Good Requirement

- ☐ Clear and Unambiguous
 - standard structure
 - has only one possible interpretation
 - Not more than one requirement in one sentence
- ☐ Correct
 - A requirement contributes to a real need
- ☐ Understandable
 - A reader can easily understand the meaning of the requirement
- ☐ Verifiable
 - A requirement can be tested
- ☐ Complete
 - Contains all required information
- ☐ Consistent
 - Does not conflict with other requirements
- ☐ Traceable
 - Has unique identity, cannot be broken into parts

Requirements readers



Types of Requirements

☐ **Functional requirements**

- Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.

☐ **Non-functional requirements**

- constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
-

Functional Requirements

As the name implies, functional requirements focus on the functionality of the software components that build the system.

High level FR are expressed in terms of:-

- Inputs
- Outputs
- Processing input data in some way

In simple words functional requirements are the services which the end users expect the final product to provide.

Documenting Functional Requirements

□ Steps:-

- Identify the set of services supported by the system
 - Specify each of the services or functionality by identifying the input data, output data and processing done on input in order to get the output.
 - Identify all the sub requirements for a high level requirement corresponding to the different user interactions
-

Non-functional requirements (NFRs)

- ❑ **NFRs are the constraints imposed on the system and deal with issues like maintainability, security, performance, reliability, probability, scalability, response time and storage requirements.**
 - ❑ Non-functional requirements are also called as
 - ❑ Quality attributes
 - ❑ Constraints
 - ❑ Goals
 - ❑ Non-behavioral requirements
 - ❑ **Non-functional requirements** may be more critical than functional requirements. If these are not met, the system is useless
-

Non-functional classifications

□ **Product requirements**

- Requirements which specify that the **delivered product must behave in a particular way** e.g. execution speed, reliability, etc.

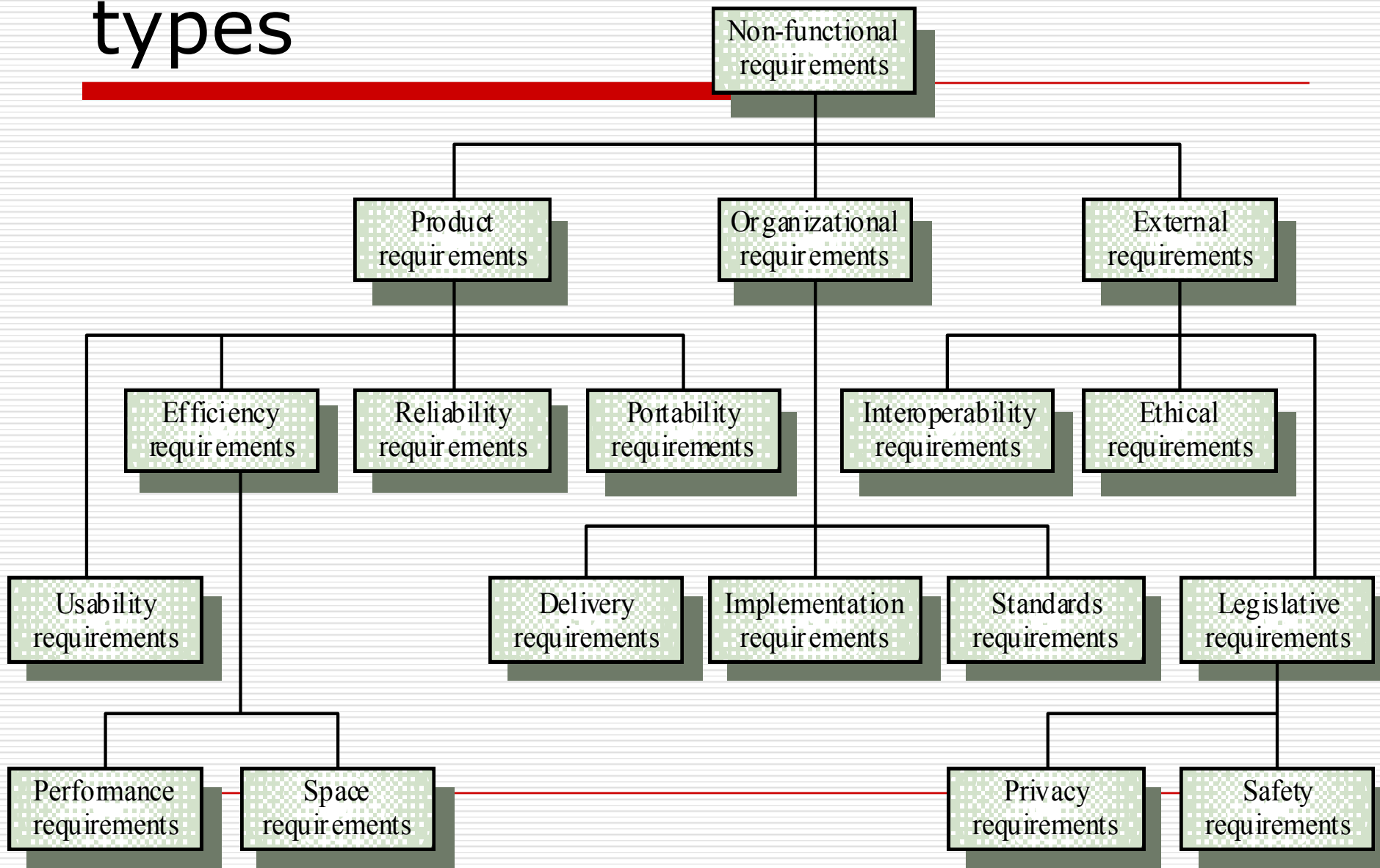
□ **Organisational requirements**

- Requirements which are a **consequence of organisational policies and procedures** e.g. process standards used, implementation requirements, etc.

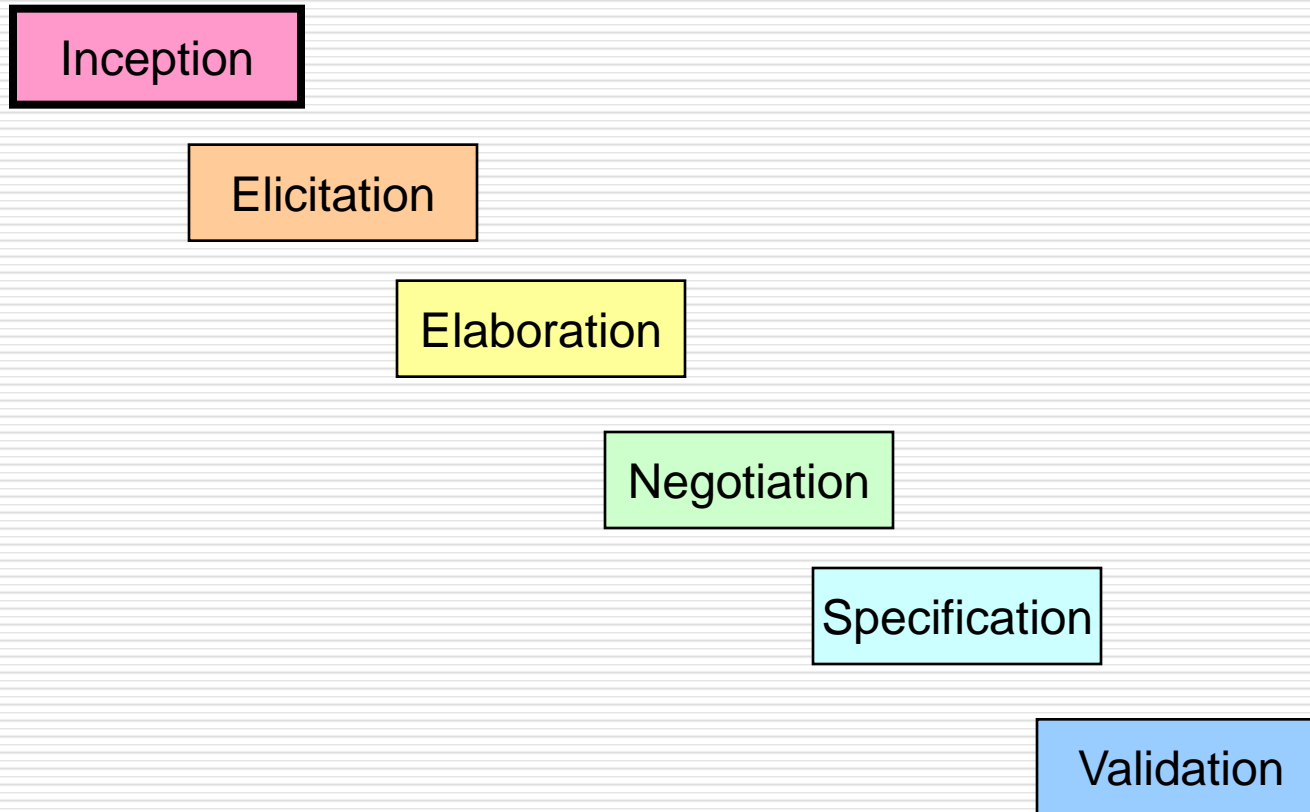
□ **External requirements**

- Requirements which arise from factors which are **external to the system and its development process** e.g. interoperability requirements, legislative requirements, etc.
-

Non-functional requirement types



Requirements Engineering Process



Requirements Engineering Process

- ❑ **Inception** —Establish a basic understanding of the problem and the nature of the solution.
 - ❑ **Elicitation** —Draw out the requirements from stakeholders.
 - ❑ **Elaboration (Highly structured)**—Create an analysis model that represents information, functional, and behavioral aspects of the requirements.
 - ❑ **Negotiation**—Agree on a deliverable system that is realistic for developers and customers.
 - ❑ **Specification**—Describe the requirements formally or informally.
 - ❑ **Validation** —Review the requirement specification for errors, ambiguities, omissions, and conflicts.
 - ❑ **Requirements management** — Manage changing requirements.
-

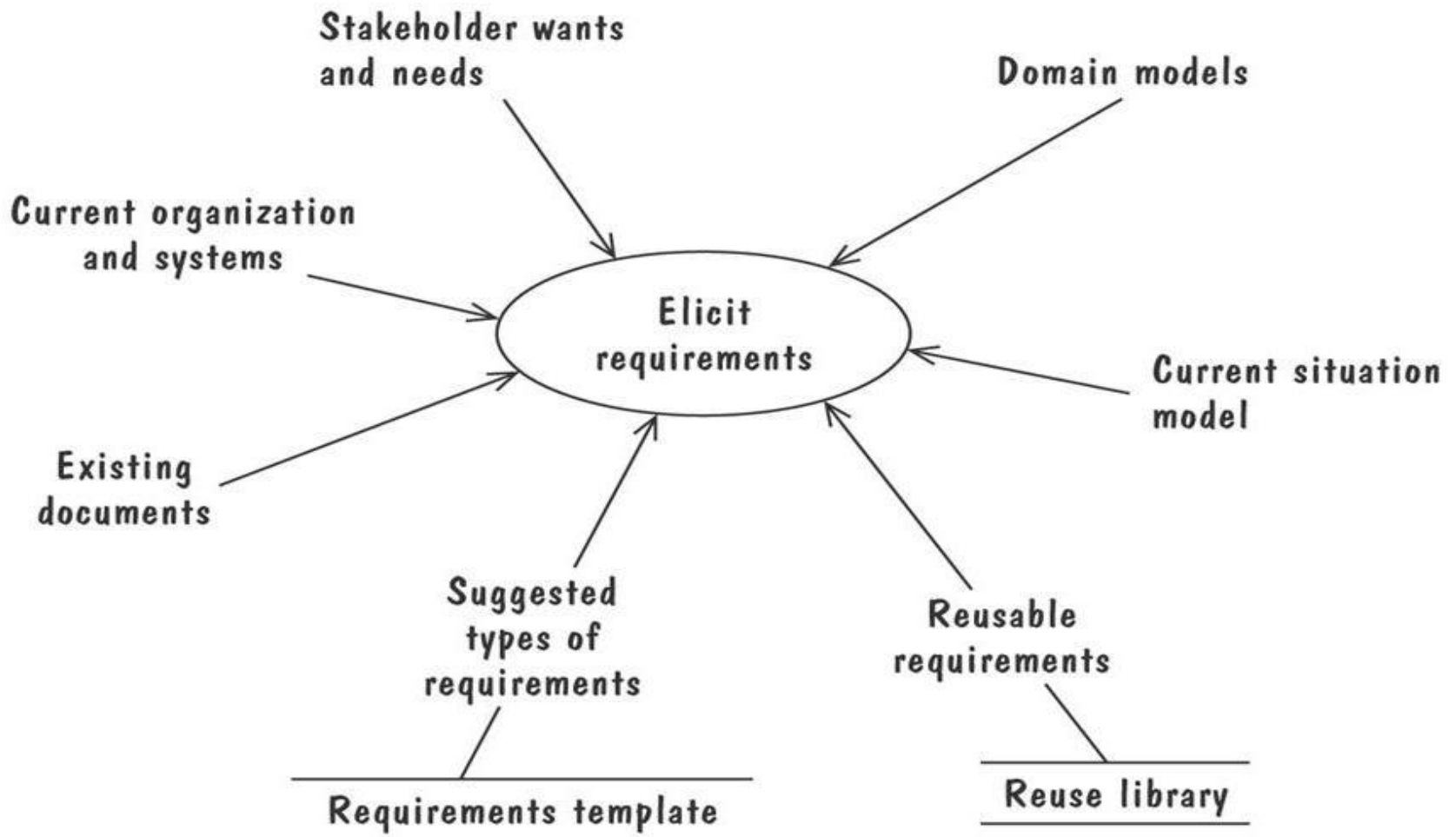
Inception

- Inception— Ask “context-free” questions that establish ...
 - Basic understanding of the problem
 - The people who want a solution
 - The nature of the solution that is desired, and
 - The effectiveness of preliminary communication and collaboration between the customer and the developer
-

Elicitation

- **Elicitation** - elicit requirements from customers, users and others.
 - Find out from customers, users and others what the product objectives are
 - what is to be done
 - how the product fits into business needs, and
 - how the product is used on a day to day basis
 - Also called as *Requirement Analysis*.
-

The Volere requirements process model suggests some additional sources for requirements.



Why Requirement elicitation is difficult?

- Problems of scope:
 - The boundary of the system is ill-defined.
 - Customers/users specify unnecessary technical detail that may confuse rather than clarify objectives.
- Problems of understanding:
 - Customers are not completely sure of what is needed.
 - Customers have a poor understanding of the capabilities and limitations of the computing environment.
 - Customers don't have a full understanding of their problem domain.
 - Customers have trouble communicating needs to the system engineer.
 - Customers omit detail that is believed to be obvious.
 - Customers specify requirements that conflict with other requirements.
 - Customers specify requirements that are ambiguous or not able to test.
- Problems of volatility:
 - Requirement change over time.

Tasks of Requirements Elicitation

- **Fact Finding** – studies the organization in which the final system will be installed and defines the scope of the proposed system
 - **Collecting Requirements** – captures information from different users viewpoint
 - **Evaluation** – focuses on identifying inconsistencies in the requirements collected
 - **Prioritization** – prioritizes the requirements depending upon cost, user needs, ease of development etc.
 - **Consolidation** – consolidates the requirements from the information gained in a way that can be analyzed further and ensures that they are consistent with the goals of the organization.
-

Requirement Eliciting Techniques

Techniques for eliciting requirement:

1. Interviews
 2. Brainstorming
 3. Task Analysis
 4. Form Analysis
 5. User Scenarios and use case based requirement elicitation
 6. Delphi Technique
 7. Domain Analysis
 8. Joint Application Design (JAD)
 9. Facilitated Application Specific Technique (FAST)
 10. Prototyping
-

Facilitated Application Specification Technique (FAST)

- ❑ FAST was developed specifically for collecting requirements and is similar to brainstorming and JAD.
 - ❑ Attended by *developers & customers/end users*.
 - ❑ The meeting is controlled by *facilitator* and an informal agenda is published.
 - ❑ Before starting of session, list of objects, functions and constraints is made and is presented in the session for discussion.
-

Facilitated Application Specification Technique (FAST)

- ❑ Participants agree not to debate. After discussion some of the entries from the list is eliminated and new entries are also added to the list. This process is continued till a consensus is reached.

 - ❑ Following activities take place additionally:
 - Identification of external data objects.
 - Detailed description of software functionality.
 - Understanding the behavior of software.
 - Establishing the interface characteristics.
 - Describing the design constraints.
-

Elaboration

- ❑ Focuses on developing a refined technical model of software functions, features, and constraints using the information obtained during inception and elicitation
 - ❑ Create an analysis model that identifies data, function and behavioral requirements.
 - ❑ It is driven by the creation and refinement of user scenarios that describe how the end-user will interact with the system.
 - ❑ Each event parsed into extracted.
 - ❑ End result defines informational, functional and behavioral domain of the problem
-

Negotiation

- **Negotiation** - agree on a deliverable system that is realistic for developers and customers
 - Requirements are categorized and organized into subsets
 - Relations among requirements identified
 - Requirements reviewed for correctness
 - Requirements prioritized based on customer needs
 - Negotiation about requirements, project cost and project timeline.
 - There should be no winner and no loser in effective negotiation.
-

Specification OR Documentation

- ❑ Specification – Different things to different people.
 - ❑ It can be –
 - Written Document
 - A set of graphical models,
 - A formal mathematical models
 - Collection of usage scenario.
 - A prototype
 - Combination of above.
 - ❑ The Formality and format of a specification varies with the size and the complexity of the software to be built.
 - ❑ For large systems, written document, language descriptions, and graphical models may be the best approach.
 - ❑ For small systems or products, usage scenarios
-

Validation

- **Requirements Validation** - formal technical review mechanism that looks for
 - Errors in content or interpretation
 - Areas where clarification may be required
 - Missing information
 - Inconsistencies (a major problem when large products or systems are engineered)
 - Conflicting or unrealistic (unachievable) requirements.
-

Software Requirement Specification (SRS) or Requirement Specification Document

- ❑ SRS document is generated as output of requirement analysis.
 - ❑ A SRS is a document which contains the complete description of software without talking much about implementation details. It is a set of precisely stated properties and constraints which final product must satisfy.
 - ❑ Clearly and accurately describes each of the essential requirements (functions, performance, design constraints, and quality attributes) of the system / software and its external interfaces
 - ❑ Defines the scope and boundaries of the system / software
-

Software Requirement Specification (SRS) or Requirement Specification Document

- ❑ Each requirement must be described in such a way that it is feasible and objectively verifiable by a prescribed method (e.g., by inspection, demonstration, analysis, or test)
- ❑ Basis for contractual agreements between contractors or suppliers and customers
- ❑ Specifications are intended to a diverse audience
 - Customers and users for validation, contract, ...
 - Systems (requirements) analysts
 - Developers, programmers to implement the system
 - Testers to check that the requirements have been met
 - Project Managers to measure and control the project

Components of SRS Document

- ☐ Functional Requirements
 - ☐ Non - Functional Requirements
 - ☐ Static Requirements
 - ☐ Execution Constraints eg. Response time, throughput time
 - ☐ Standards to be followed
 - ☐ Security requirements
 - ☐ Company Policies
 - ☐ Interface with other external agents ie. persons, software or hardware
-

GOALS OF SRS DOCUMENT

1. **Feedback to customer** : It is the customer's assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems.
 2. **Problem Decomposition**: SRS helps to break down the problem into its component parts in an orderly fashion.
 3. **Input to Design Specification** : SRS contains sufficient detail in the functional system requirements so that a design solution can be devised.
 4. **Product Validation Check**: SRS serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.
-

Characteristics of an SRS

1. Correct
 2. Complete
 3. Unambiguous
 4. Consistent
 5. Verifiable
 6. Traceable
 7. Modifiable
 8. Ranked for importance and/or stability
 9. Design Independent
 10. Testability
 11. Understandable by Customer
 12. Right level of abstraction
-

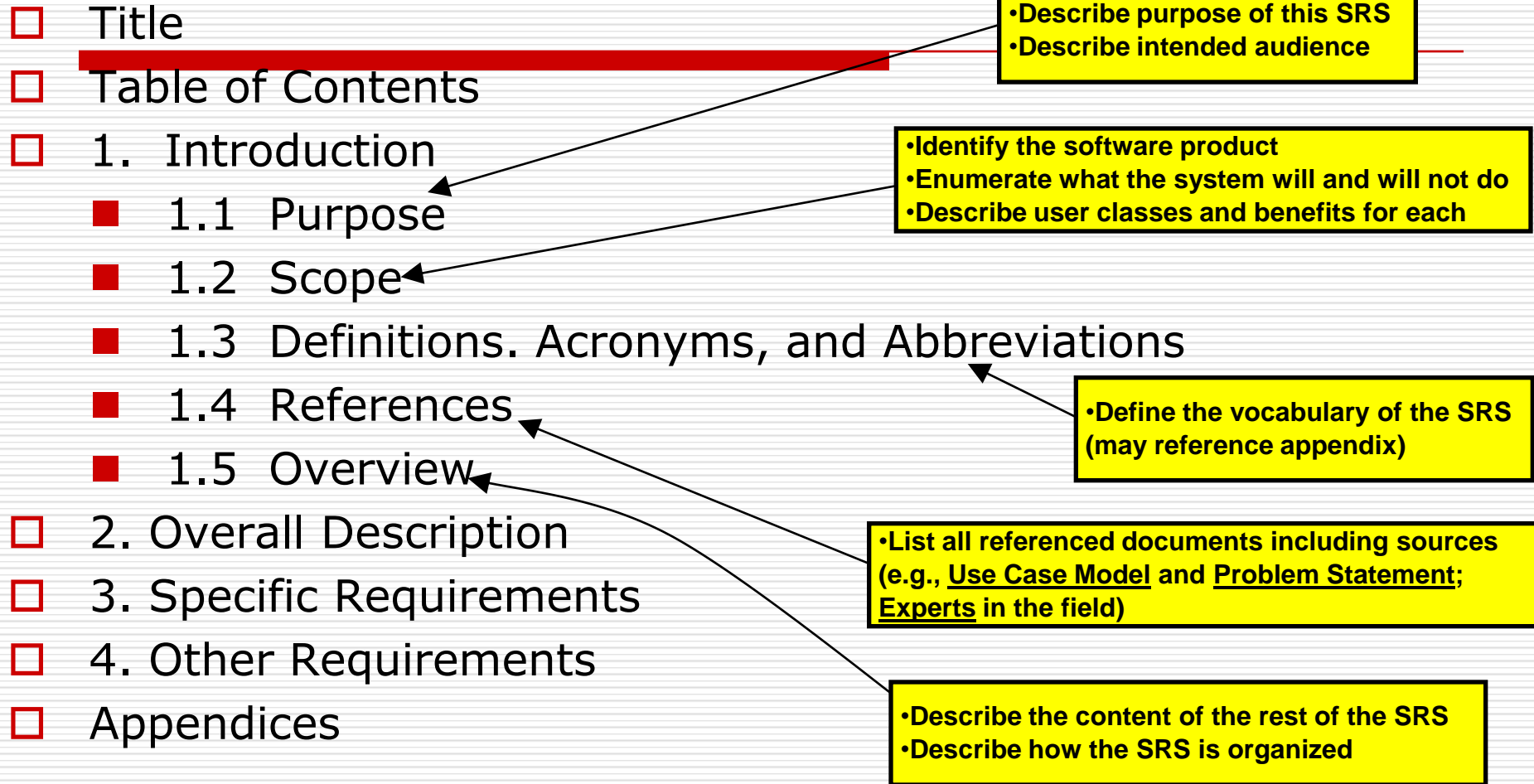
IEEE 830-1998 Standard – Structure of the SRS

□ Section 5 of IEEE 830

□ Contents of SRS

1. Introduction
 2. General description of the software product
 3. Specific requirements (detailed)
 4. Additional information such as appendixes and index, if necessary
-

IEEE 830-1998 Standard – Section 1 of SRS



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. Other Requirements
- Appendices

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


•Summarize the major functional capabilities
•Include the Use Case Diagram and supporting narrative (identify actors and use cases)
•Include Data Flow Diagram if appropriate

•Describe and justify technical skills and capabilities of each user class

States assumptions about availability of certain resources that, if not satisfied, will alter system requirements and/or effect the design.

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

IEEE 830-1998 Standard – Section 3 of SRS

- ☐ ...
- ☐ 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. Other Requirements
- ☐ Appendices

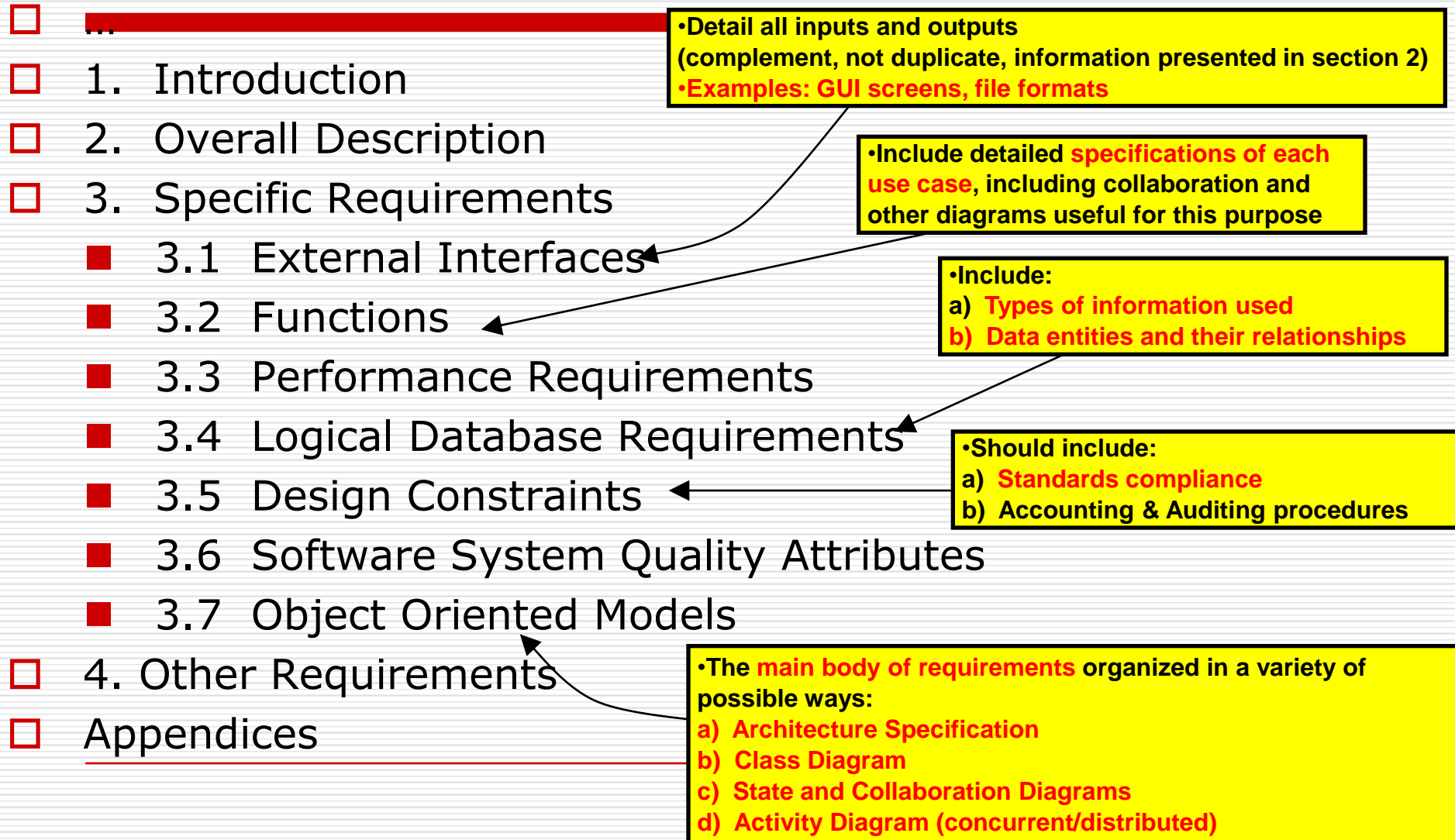
Specify software requirements in sufficient detail to enable designers to design a system to satisfy those requirements and testers to verify requirements

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

Requirements should include, at a minimum, 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 or in support of an output

- (a) Requirements should have characteristics of high quality requirements
- (b) Requirements should be cross-referenced to their source.
- (c) Requirements should be uniquely identifiable
- (d) Requirements should be organized to maximize readability

IEEE 830-1998 Standard – Section 3 of SRS



IEEE 830-1998 Standard – Section 4 of SRS

- Title
- Table of Contents
- 1. Introduction
- 2. Overall Description
- 3. Specific Requirements
- 4. Other Requirements
 - Apportioning of requirements
 - Database
 - Schedule & budgets
- Appendices

give the details of requirements which are given to the third party for development.

describes the details of the database which will be developed as a part of the project.

describes the detailed project plan and schedule and budget estimate.

Requirements vs. Design

Requirements	Design
Describe what will be delivered	Describe how it will be done
Primary goal of analysis: UNDERSTANDING	Primary goal of design: OPTIMIZATION
There is more than one solution	There is only one (final) solution
Customer interested	Customer not interested (Most of the time) except for external

Quality Function Deployment (QFD)

A technique of translating customer needs into technical system requirements:

Identifies three types of requirements

- ❑ **Normal requirements:** reflect stated customer goals and objectives
 - ❑ **Expected requirements:** implicit to the product or system; their absence will cause significant customer dissatisfaction
 - ❑ **Exciting requirements:** featured going beyond customer expectations, causing customer euphoria (;-)
-
- ❑ concentrate on maximizing customer satisfaction

-
- ❑ **Function deployment**: determines the “value” (as perceived by the customer) of each function required of the system
 - ❑ **Information deployment**: identifies data objects and events, ties them to functions
 - ❑ **Task deployment**: examines the behavior of the system
 - ❑ **Value analysis**: determines the relative priority of requirements

Analysis Modelling

Goals of Analysis Modelling

1. Provides the first technical representation of a system
 2. Is easy to understand and maintain
 3. Deals with the problem of size by partitioning the system
 4. Uses graphics whenever possible
 5. Differentiates between **essential** information versus **implementation** information
 6. Helps in the tracking and evaluation of interfaces
 7. Provides tools other than narrative text to describe software logic and policy
-

Analysis Modeling Approaches

Structured analysis

- Considers data and the processes that transform the data as separate entities
- Top-down decomposition approach
- Data is modelled in terms of only attributes and relationships (but no operations)
- Processes are modelled to show the 1) input data, 2) the transformation that occurs on that data, and 3) the resulting output data



Analysis Modeling Approaches

Object-oriented analysis

- Focuses on the definition of classes and the manner in which they collaborate with one another to fulfil customer requirements
-

Elements of the Analysis Model

Object-oriented Analysis

Scenario-based modeling

Use case text
Use case diagrams
Activity diagrams
Swim lane diagrams

Class-based modeling

Class diagrams
Analysis packages
CRC models
Collaboration diagrams

Structured Analysis

Flow-oriented modeling

Data structure diagrams
Data flow diagrams
Control-flow diagrams
Processing narratives

Behavioral modeling

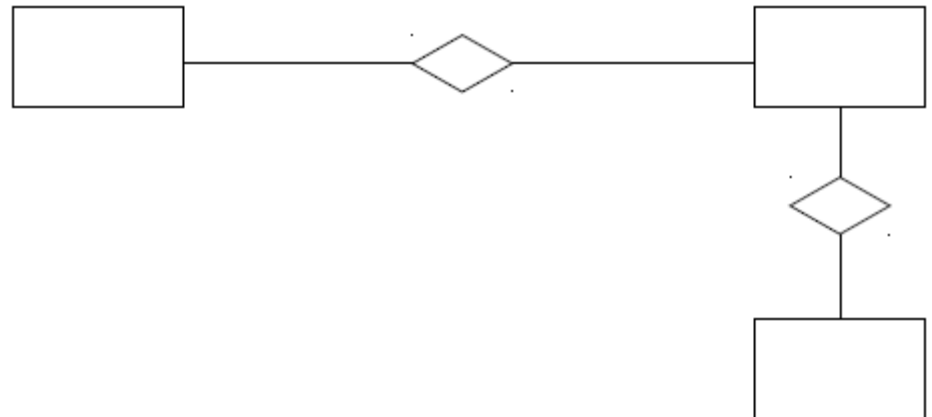
State diagrams
Sequence diagrams

Flow-oriented Modelling

Data Modelling

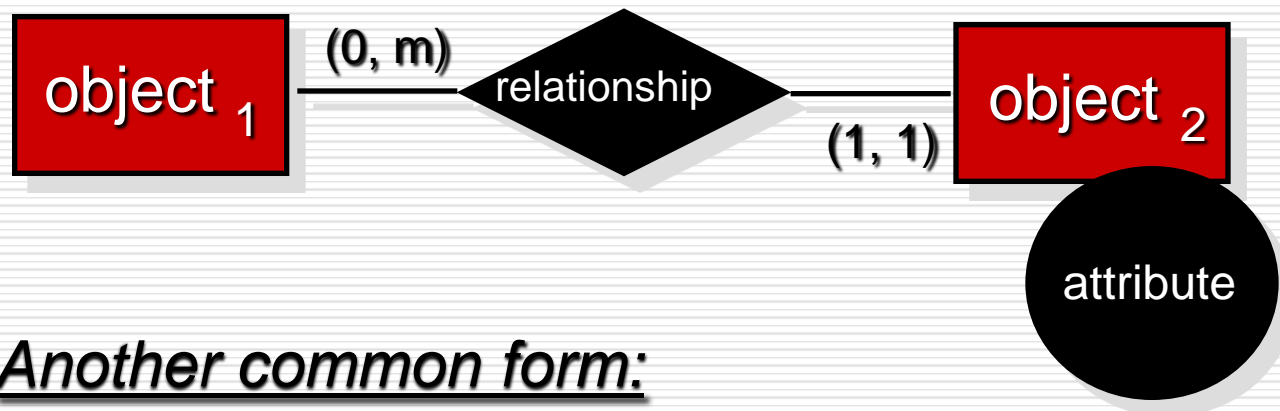
➤ Identify the following items

- Data objects (Entities)
- Data attributes
- Relationships
- Cardinality (number of occurrences) Data is modelled in terms of only attributes and relationships (but no operations)

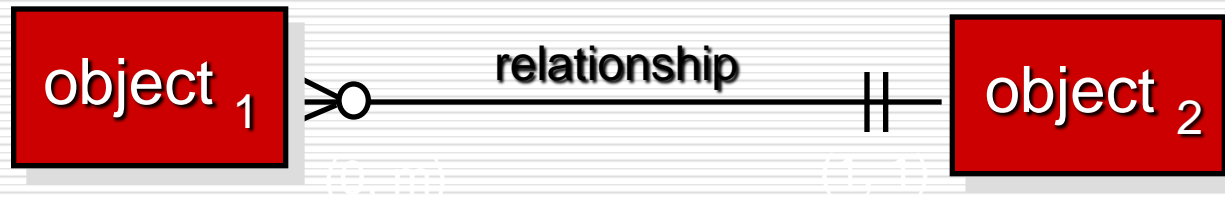


ERD Notation

One common form:

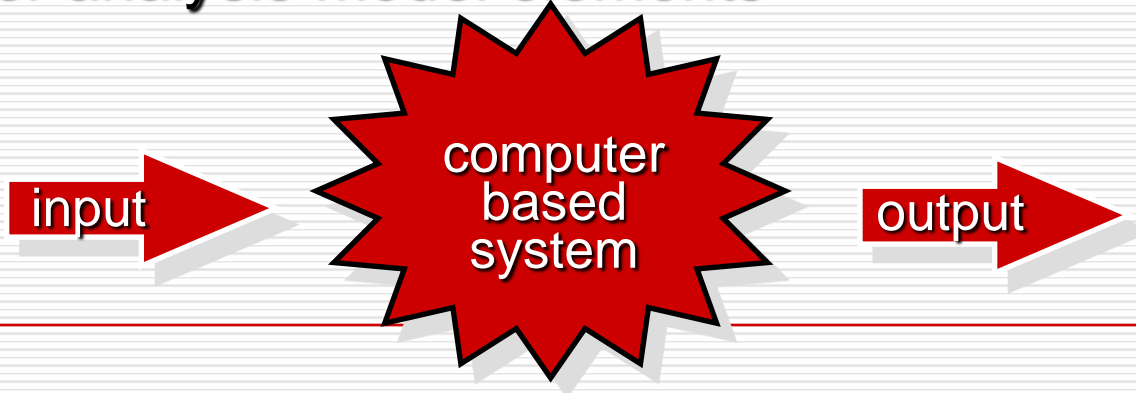


Another common form:



Data Flow Diagram (DFD)

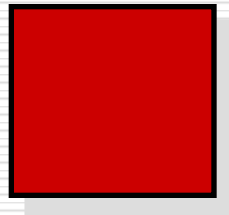
- A **data flow diagram (DFD)** is a graphical representation of the "flow" of data through an information system.
- A **data flow diagram (DFD)** is the diagrammatic form that is used
- Considered by many to be an 'old school' approach, flow-oriented modeling continues to provide a view of the system that is unique—it should be used to supplement other analysis model elements



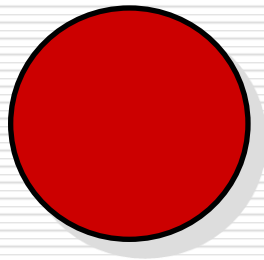
Data Flow Diagram (DFD)

- The system is represented in terms of the input data to the system, various processing carried out on these data, and the output data generated by the system.
 - Each function is considered as a processing station (or process) that consumes some input data and produces some output data.
 - A DFD model uses a very limited number of primitive symbols to represent the functions performed by a system and the data flow among these functions.
-

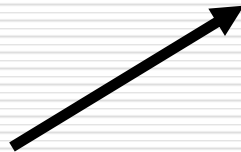
Flow Modeling Notation



external entity



process



data flow



data store

External Entity



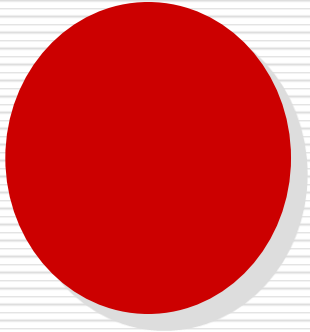
A producer or consumer of data

Examples: a person, a device, a sensor

Another example: computer-based system

Data must always originate somewhere and must always be sent to something

Process

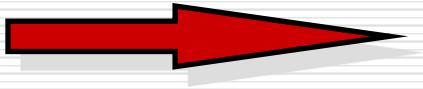


A data transformer (changes input to output)

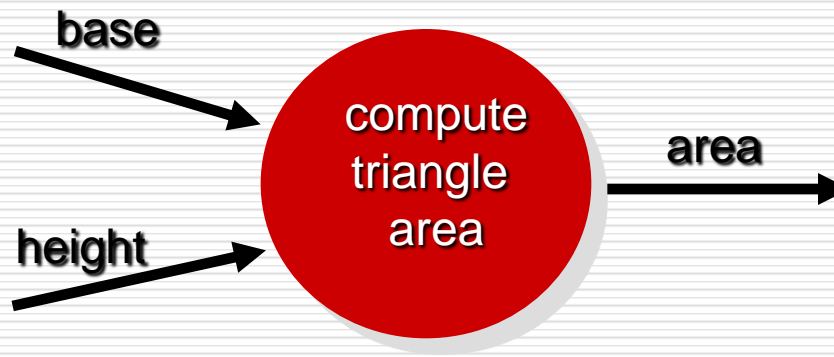
Examples: compute taxes, determine area, format report, display graph

Data must always be processed in some way to achieve system function

Data Flow

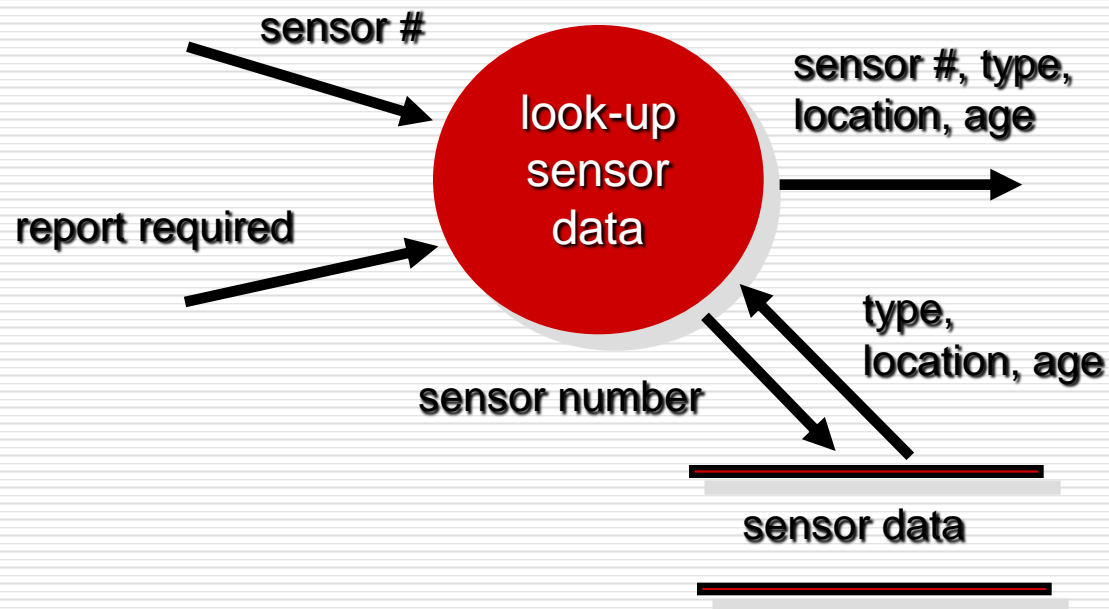


Data flows through a system, beginning as input and be transformed into output.



Data Stores

Data is often stored for later use.



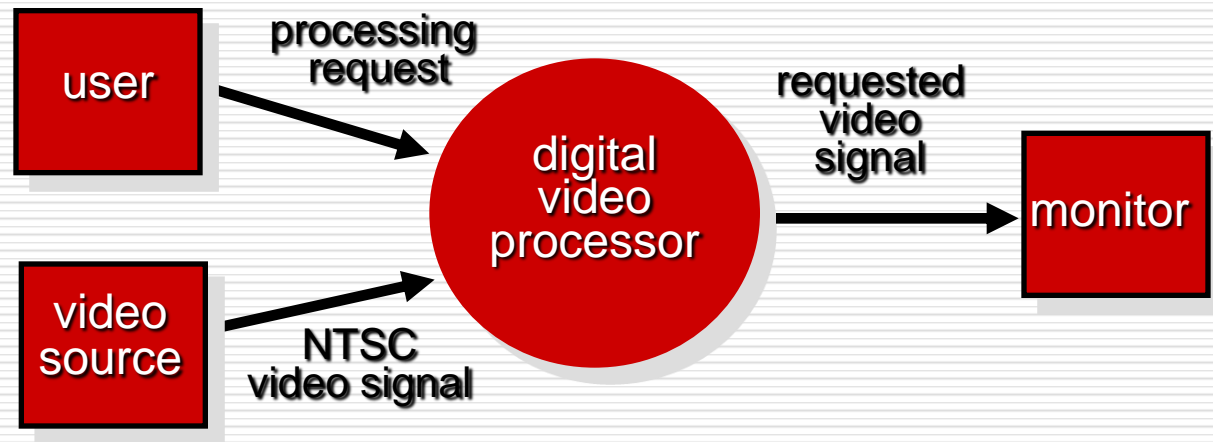
Data Flow Diagramming: Guidelines or Rules

- ❑ all icons must be labeled with meaningful names
 - ❑ the DFD evolves through a number of levels of detail
 - ❑ always begin with a context level diagram (also called level 0)
 - ❑ always show external entities at level 0
 - ❑ always label data flow arrows
 - ❑ do not represent procedural logic
-

Constructing a DFD

- ☐ review the data model to isolate data objects and use a grammatical parse to determine “operations”
 - ☐ determine external entities (producers and consumers of data)
 - ☐ create a level 0 DFD
-

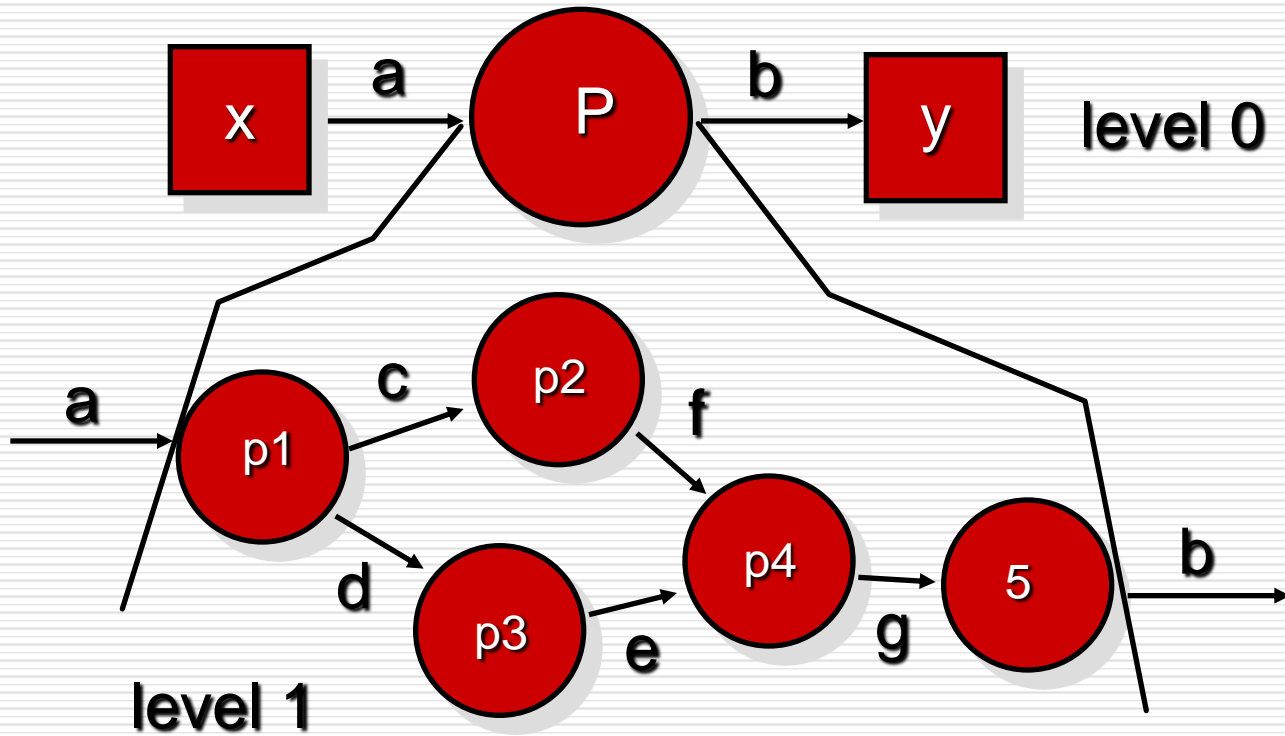
Level 0 DFD Example



Constructing a DFD

- ❑ write a narrative describing the transform
- ❑ parse to determine next level transforms
- ❑ “balance” the flow to maintain data flow continuity
- ❑ develop a level 1 DFD
- ❑ use a 1:5 (approx.) expansion ratio

The Data Flow Hierarchy



Flow Modeling Notes

- ❑ each bubble is refined until it does just one thing
 - ❑ the expansion ratio decreases as the number of levels increase
 - ❑ most systems require between 3 and 7 levels for an adequate flow model
 - ❑ a single data flow item (arrow) may be expanded as levels increase (data dictionary provides information)
-

Physical and Logical DFDs

□ Logical model

- Assumes implementation in perfect technology
- Does not tell how system is implemented

□ Physical model

- Describes assumptions about implementation technology
- Developed in last stages of analysis or in early design

Data Dictionary

- ❑ is the centralized collection of information about data.
- ❑ It stores meaning and origin of data, its relationship with other data, data format for usage etc.
- ❑ is often referenced as meta-data (data about data) repository.
- ❑ It is created along with DFD (Data Flow Diagram) model of software program and is expected to be updated whenever DFD is changed or updated.

Data Dictionary

- ❑ The data is referenced via data dictionary while designing and implementing software.
- ❑ Data dictionary removes any chances of ambiguity.
- ❑ It helps keeping work of programmers and designers synchronized while using same object reference everywhere in the program.
- ❑ Data dictionary provides a way of documentation for the complete database system in one place. Validation of DFD is carried out using data dictionary.

Data Dictionary Content

- Data dictionary should contain information about the following
 - Data Flow
 - Data Structure
 - Data Elements
 - Data Stores
 - Data Processing

Data Dictionary Content

- Data Flow is described by means of DFDs as studied earlier and represented in algebraic form as described.
- Data structure
 - Address = House No + (Street / Area) + City + State
 - Course ID = Course Number + Course Name + Course Level + Course Grades

Data Dictionary Content

□ Data Elements

- Data elements consist of Name and descriptions of Data and Control Items, Internal or External data stores etc. with the following details:
 - Primary Name
 - Secondary Name (Alias)
 - Use-case (How and where to use)
 - Content Description (Notation etc.)
 - Supplementary Information (preset values, constraints etc.)
-

Data Dictionary Content

☐ **Data Store**

- ☐ It stores the information from where the data enters into the system and exists out of the system. The Data Store may include

☐ **Files**

- ☐ Internal to software.
- ☐ External to software but on the same machine.
- ☐ External to software and system, located on different machine.

☐ **Tables**

-
- ☐ Naming convention
 - ☐ Indexing property

Data Dictionary Content

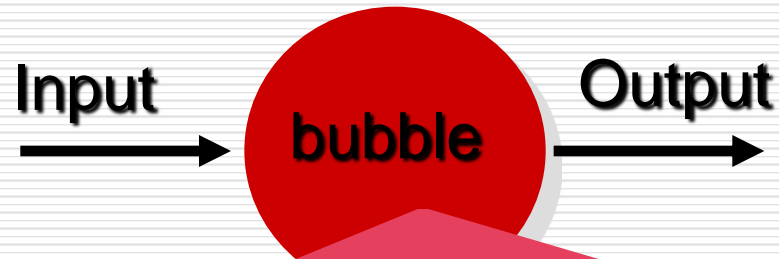
☐ Data Processing

☐ There are two types of Data Processing:

- **Logical:** As user sees it

- **Physical:** As software sees it

Process Specification (PSPEC)



PSPEC

- ❑ narrative
- ❑ pseudocode (PDL)
- ❑ equations
- ❑ tables
- ❑ diagrams and/or charts

Process Specification for a DFD

- specifies process “bubbles” in a DFD
 - NOT sources and sinks
 - must be consistent with DFD
 - specification compatible with diagram
 - must be complete with regard to the DFD
 - consists of
 - process decomposition (diagrams)
 - process description
-

Process Description

- ❑ (see also IEEE 830-1984)
 - ❑ name (and aliases)
 - ❑ informal description
 - ❑ inputs (see DFD & data dictionary)
 - ❑ outputs (see DFD & data dictionary)
 - ❑ processing (semi-formal)
 - ❑ related processes, sources and sinks
-

Process Specification Format

- ❑ Process specifications link the process to the DFD and the data dictionary.
- ❑ The following information should be entered:
 - The process number, which must match the process ID on the data flow diagram.
 - This allows an analyst to work or review any process and easily locate the data flow diagram containing the process.

Process Specification Format (Continued)

- The process name, the same as displays within the process symbol on the DFD.
- A brief description of what the process accomplishes.
- A list of input and output data flow, using the names found on the data flow diagram.
- Data names used in the formulae or logic should match the data dictionary, for consistency and good communication.

Process Specification Format (Continued)

- An indication of the type of process, whether it is batch, online, or manual.
- All online processes require screen designs.
- All manual processes should have well-defined procedures for employees performing the process tasks.
- If the process has prewritten code for it, include the name of the subprogram or function.

Process Specification Format (Continued)

- A description of the process logic.
- This should state policy and business rules, not computer language pseudocode.
- A reference to further information, such as a structured English description, a decision table, or tree depicting the logic.
- List any unresolved issues.
- These issues form the basis of the questions used for a follow-up interview.

Specifying Processes in DFDs

- Structured English

- Decision Trees

- Decision Tables

Structured English

- ❑ uses plain English words in structured programming paradigm. It is not the ultimate code but a kind of description **what is required to code and how to code it.**
 - ❑ Used when the process logic involves formulas or iteration, or when structured decisions are not complex
 - ❑ Based on structured logic and simple English statements such as add, multiply, and move
-

Writing Structured English

- ❑ Express all logic in terms of sequential structures, decision structures, case structures, or iterations
- ❑ Use and capitalize accepted keywords such as IF, THEN, ELSE, DO, and PERFORM
- ❑ Indent blocks of statements to show their hierarchy (nesting) clearly
- ❑ Underline words or phrases that have been defined in a data dictionary
- ❑ Clarify the logical statements

Examples of Structured English Types

Structured English Type	Example
Sequential Structure A block of instructions in which no branching occurs	Action #1 Action #2 Action #3
Decision Structure Only IF a condition is true, complete the following statements; otherwise, jump to the ELSE	IF Condition A is True THEN implement Action A ELSE implement Action B ENDIF
Case Structure A special type of decision structure in which the cases are mutually exclusive (if one occurs, the others cannot)	IF Case #1 implement Action #1 ELSE IF Case #2 Implement Action #2 ELSE IF Case #3 Implement Action #3 ELSE IF Case #4 Implement Action #4 ELSE print error ENDIF
Iteration Blocks of statements that are repeated until done	DO WHILE there are customers. Action #1 ENDDO

Advantages of Structured English

- ❑ Clarifying the logic and relationships found in human languages
- ❑ An effective communication tool, it can be taught to and understood by users in the organization

Decision Tables

- ❑ represents conditions and the respective actions to be taken to address them, in a structured tabular format.
- ❑ It is a powerful tool to debug and prevent errors.
- ❑ It helps group similar information into a single table and then by combining tables it delivers easy and convenient decision-making.

Decision Tables

- A decision table lists causes and effects in a matrix. Each column represents a unique combination.

		Combinations							
Causes	Values	1	2	3	4	5	6	7	8
Cause 1	Y, N	Y	Y	Y	Y	N	N	N	N
Cause 2	Y, N	Y	Y	N	N	Y	Y	N	N
Cause 3	Y, N	Y	N	Y	N	Y	N	Y	N
Effects									
Effect 1		X			X				X
Effect 2			X				X		X

Cause = condition

Effect = action = expected results

Decision Tables

- A customer requests a cash withdrawal. One of the business rules for the ATM is that the ATM machine pays out the amount if the customer has sufficient funds in their account or if the customer has the credit granted.

Conditions	R1	R2	R3
Withdrawal Amount \leq Balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

Developing Decision Tables

- ☐ Determine conditions that affect the decision
- ☐ Determine possible actions that can be taken
- ☐ Determine condition alternatives for each condition
- ☐ Calculate the maximum number of columns in the decision table

Developing Decision Tables

- ❑ Fill in the condition alternatives
- ❑ Complete table by inserting an X where rules suggest actions
- ❑ Combine rules where it is apparent
- ❑ Check for impossible situations
- ❑ Rearrange to make more understandable

Step 1 Analyze the requirement and create the first column

- ❑ Requirement: "Withdrawal is granted if requested amount is covered by the balance or if the customer is granted credit to cover the withdrawal amount".
- ❑ Express conditions and resulting actions in a list so that they are either TRUE or FALSE.
- ❑ In this case there are two conditions, "withdrawal amount \leq balance" and "credit granted". There is one result, the withdrawal is granted.

Conditions
Withdrawal Amount \leq Balance
Credit granted
Actions
Withdrawal granted

Step 2 add Colum

- ❑ Calculate how many columns are needed in the table. The number of columns depends on the number of conditions and the number of alternatives for each condition.
- ❑ If there are two conditions and each condition can be either true or false, you need 4 columns. If there are three conditions there will be 8 columns and so on.
- ❑ Mathematically, the number of columns is $2^{\text{conditions}}$.
- ❑ In this case $2^2 = 4$ columns.

Number of Conditions	Number of Columns
1	2
2	4
3	8
4	16
5	32

Step 2 Now is the time to fill in the T (TRUE) and F (FALSE) for the conditions

- ❑ How do you do that? The simplest is to say that it should look like this:
 - ❑ Row 1: TF
 - ❑ Row 2: TTFF
 - ❑ Row 3: TTTFFFFF
 - ❑ For each row, there is twice as many T and F as the previous line.
 - ❑ Repeat the pattern above from left to right for the entire row.
 - ❑ In other words, for a table with 8 columns, the first row will read TFTFTFTF, the second row will read TTFFTTFF and the third row will read TTTTFFFF.
-

Step 3 Reduce the table

- ❑ Mark insignificant values with "-".
- ❑ If the requested amount is less than or equal to the account balance it does not matter if credit is granted.
- ❑ In the next step, you can delete the columns that have become identical.

Conditions				
Withdrawal Amount \leq Balance	T	F	T	F
Credit granted	-	T	-	F
Actions				
Withdrawal granted				

Step 3 Check for invalid combinations

- ❑ Invalid combinations are those that cannot happen, for example, that someone is both an infant and senior. Mark them somehow, e.g. with "X". In this example, there are no invalid combinations.
 - ❑ Finish by removing duplicate columns.
 - ❑ In this case, the first and third column are equal, therefore one of them is removed.
-

Step 4 determine action

- ❑ Enter actions for each column in the table.
- ❑ You will be able to find this information in the requirement.
- ❑ Name the columns (the rules).
- ❑ They may be named R1/Rule 1, R2/Rule 2 and so on, but you can also give them more descriptive names

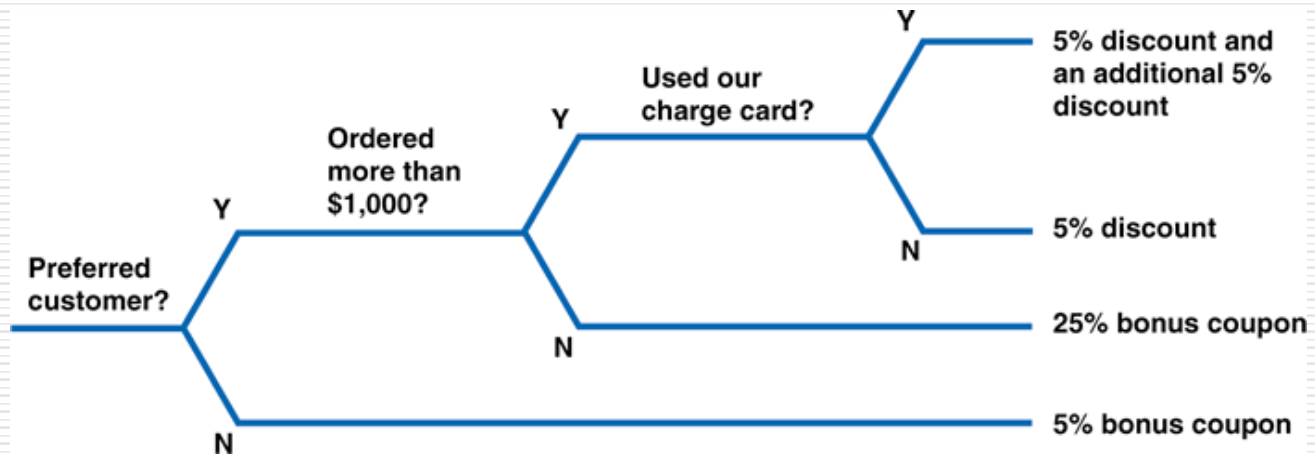
Conditions			
Withdrawal Amount \leq Balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

Step 5 Write test cases

- ❑ Write test cases based on the table. At least one test case per column gives full coverage of all business rules.
- ❑ Test case for R1: balance = 200, requested withdrawal = 200. Expected result: withdrawal granted.
- ❑ Test case for R2: balance = 100, requested withdrawal = 200, credit granted. Expected result: withdrawal granted.
- ❑ Test case for R3: balance = 100, requested withdrawal = 200, no credit. Expected Result: withdrawal denied.

Decision Trees

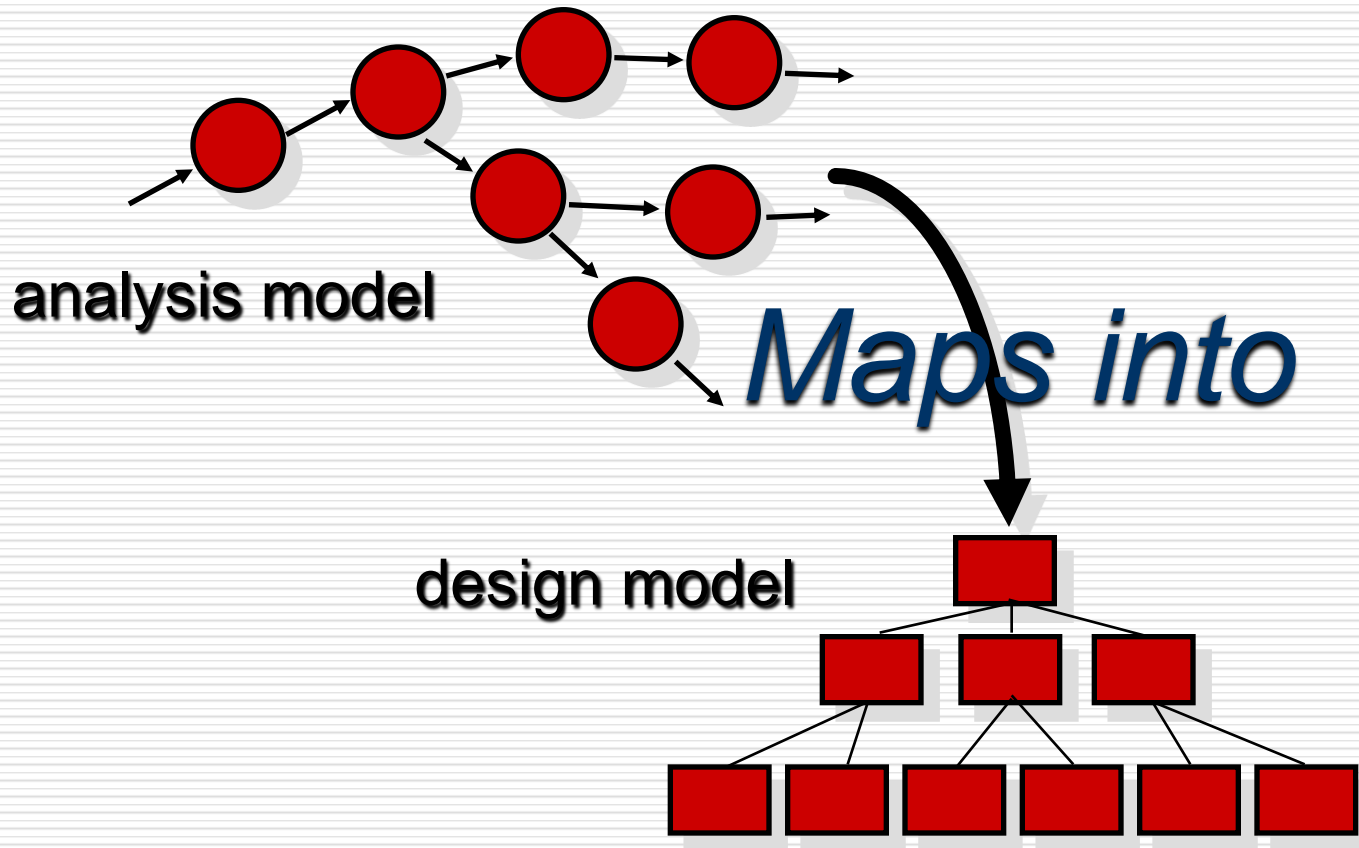
- Graphical representation of the conditions, actions, and rules found in a decision table
- Whether to use a decision table or tree often is a matter of personal preference



WHEN TO USE STRUCTURED ENGLISH, DECISION TABLES AND DECISION TREES

- **Use Structured English if there are many loops and actions are complex**
 - **Use Decision tables when there are a large number of conditions to check and logic is complex**
 - **Use Decision trees when sequencing of conditions is important and if there are not many conditions to be tested**
-

DFDs: A Look Ahead



-
- Produce a decision table to model the logic in this scenario:
 - A postal delivery company delivers parcels by air or rail transport. The price of delivery by air depends upon the weight of the parcel. There is a basic charge of 5 euros per kilo up to 50 kilos. Excess weight over 50 kilos is charged at 3 euros per kilo.
 - Delivery by rail is charged at 3 euros per kilo up to 50 kilos and then 2 euros per kilo. There is a special service guaranteeing same day delivery which carries an additional flat rate charge of 20 euros.
-

-
- Produce a structured English specification for this scenario:
 - A travel agent has account customers and individual customers. Account customers who have spent over 25,000 Rs. in the past year get a discount of 25%. Otherwise, they get 10% discount. Individual customers who have booked holidays previously get 5% discount. New customers get no discount. Account customers who have spent over 10,000 Rs. in any previous year will receive offers of free tickets on selected routes.
-

