

UNIT 3

Analysis

(13 Hours) — System Analysis and Design (CACS203)

1. System Requirements

1.1 Introduction

Analysis is the phase in which the team learns what the system must do. **Requirements determination** is the process of gathering and studying information about the current system and the needs for the new one. Requirements are either **functional** (what the system does — e.g., “generate an invoice”) or **non-functional** (how well it does it — performance, security, usability, reliability, availability).

1.2 Performing Requirements Determination

The analyst gathers **information about the current system** (people, data, procedures, problems), the **organization** (structure, goals, policies) and the **desired system**. Good analysts stay **impartial**, question **assumptions**, pay attention to **details**, **reframe** problems in new ways and stay **organized**. The result is the **System Requirements Specification**, which should be complete, correct, consistent, unambiguous, feasible and traceable.

1.3 Traditional Methods for Determining Requirements

Method	Description	Strengths	Weaknesses
Interviews	Face-to-face question-and-answer with users (structured or unstructured)	Rich, flexible, builds rapport	Time-consuming, depends on interviewer skill
Questionnaires	Written questions to many respondents	Cheap, reaches many people	Low response, no follow-up
Direct observation	Watch users perform their work	Shows real practice	Observer effect, time-consuming
Document analysis	Study forms, reports, manuals, policies	Reveals formal rules and data	May be outdated, not the real practice
Sampling / joint reviews	Select a sample of records or users	Saves effort, statistically sound	Sample may be biased

Example 3.1 — Interview vs questionnaire

An analyst must gather requirements for a hospital appointment system from 5 doctors and 800 patients. Suggest methods.

Solution

Use **interviews** for the 5 doctors (detailed, complex needs) and a **questionnaire** for the 800 patients (large group, simple questions). Supplement with **observation** of the reception desk and **document analysis** of existing appointment forms.

1.4 Contemporary Methods

- **Joint Application Design (JAD)** — a structured workshop of users, managers and analysts led by a facilitator, with a scribe and a fixed agenda; it reaches agreement quickly and reduces later rework.
- **Group Support Systems (GSS)** — electronic meeting software for anonymous, parallel idea generation.
- **CASE tools** — capture requirements and diagrams in a shared repository.
- **Prototyping** — build a quick working model of screens or functions so that users can react to it; repeated until the requirements are accepted. Suitable when requirements are unclear or the interface is critical.

1.5 Radical Methods

Business Process Reengineering (BPR) means a radical redesign of business processes to achieve dramatic improvements in cost, quality, speed and service. Instead of automating the current process, the team asks why it exists and how it would be done if started afresh. Techniques include **process identification**, **disruptive technologies** (e.g., online forms replacing paper approval chains), and eliminating steps that add no value. BPR is risky and needs strong management support.

1.6 Requirements Management Tools and Agile Requirements

Requirements management tools (e.g., Jira, IBM DOORS, Azure DevOps) store requirements, assign identifiers, record status and owners, trace each requirement to design, code and tests (**traceability**), and control changes.

With **agile methodologies**, requirements are captured continuously as **user stories** in a **product backlog**, in the form “As a <role>, I want <feature> so that <benefit>.” The backlog is

prioritized by the product owner, refined every sprint, and the customer is available continuously for clarification. Acceptance criteria define when a story is done.

Example 3.2 — User story

Write a user story for a library member who wants to renew books online.

Solution

As a library member, **I want** to renew my borrowed books online **so that** I do not have to visit the library.

Acceptance criteria: the book must not be reserved by others; the new due date is shown; a confirmation e-mail is sent.

2. System Process Requirements

2.1 Introduction and Process Modeling

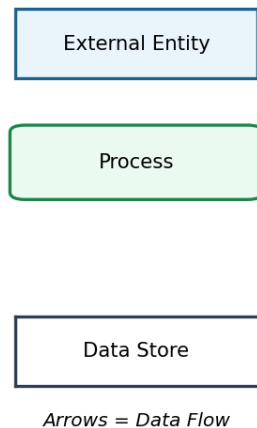
Process modeling graphically represents the processes that capture, manipulate, store and distribute data between a system and its environment and within the system. The main tool is the **Data Flow Diagram (DFD)**, which shows the flow of data through the system, independent of physical implementation. DFDs help analysts understand the existing system, communicate with users and identify improvements.

2.2 Data Flow Diagramming Mechanics

A DFD uses four building blocks:

Symbol	Meaning	Naming rule
Data flow (arrow)	Data in motion between components	Noun phrase describing the data, e.g., “Order details”
Process (rounded rectangle or circle)	Work that transforms input data into output	Verb phrase with an ID, e.g., “1.0 Validate order”
Data store (open rectangle)	Data at rest	Noun, e.g., “D1 Customer”
External entity / source-sink (square)	Person or system outside the boundary	Noun, e.g., “Customer”

DFD Symbols (Gane-Sarson)



Context Diagram: Online Library System

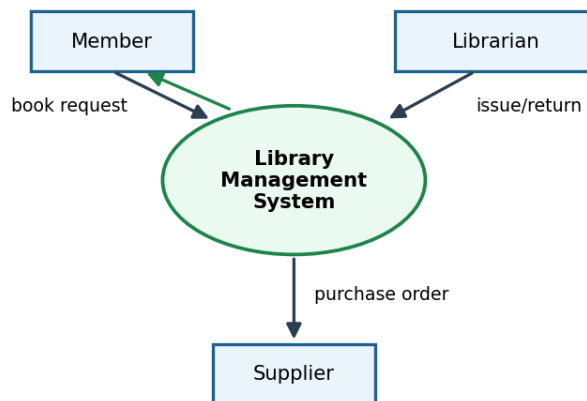


Figure 3.1 — DFD symbols and a context diagram for a library system

2.3 Rules of DFD Construction

- Every process must have **at least one input and one output**; a process with no input is a “miracle”, one with no output is a “black hole”.
- Data cannot flow directly between two external entities, two data stores, or between an entity and a data store — a **process** must be involved.
- Each data flow is named; a process is named with a verb; a data store is named with a noun.
- Processes are numbered; data flows leaving a process are different from those entering it.
- The diagram is **balanced**: inputs and outputs of a parent process match those of its child diagram.
- The **context diagram** shows the system as a single process with external entities; **Level-0** diagram breaks it into major processes; further levels decompose each process (**levelling**).

2.4 Using DFDs in the Analysis Process

Analysts usually draw (1) a **context diagram**, (2) a **Level-0 (system) diagram**, and (3) detailed child diagrams (Level-1, 2 ...) until each process is a **functional primitive** that can be described in a few lines of logic. Both **current logical/physical** and **new logical/physical** DFDs may be produced. Good practice: limit a diagram to about 7 ± 2 processes, use meaningful names, avoid crossing lines and check for completeness with the **data dictionary**.

Example 3.3 — Level-0 DFD for order processing

Draw the processes for a simple order system with entity *Customer*, store *Orders*, and store *Inventory*.

Solution (described)

1.0 Receive Order: input *Order details* from Customer; output *Valid order* to store D1 Orders.

2.0 Check Stock: input *Valid order* from D1; reads D2 Inventory (*Stock status*); output *Stock confirmation*.

3.0 Generate Invoice: input *Stock confirmation*; output *Invoice* to Customer; updates D1 (*Order status*).

The context diagram shows just one process “Order Processing System” with Customer sending *Order details* and receiving *Invoice*.

2.5 Modeling Logic with Decision Tables

A **decision table** presents complex conditional logic in a compact tabular form. It has four quadrants: **condition stubs**, **condition entries** (Y/N/–), **action stubs** and **action entries** (X). Each column is a **rule**. For n binary conditions there are 2^n rules. Tables should be checked for **completeness**, **redundancy** and **contradiction**, and simplified by merging rules that lead to identical actions using “–” (don't care).

Example 3.4 — Discount policy

A store gives a 10% discount if the customer is a member. If the purchase is over Rs. 5,000, a free delivery is also given, regardless of membership. Build the decision table.

Solution

Conditions: C1 Member? C2 Purchase over Rs. 5,000? — $2^2 = 4$ rules.

Rule →	R1	R2	R3	R4
C1: Member?	Y	Y	N	N
C2: Purchase > 5,000?	Y	N	Y	N
A1: Give 10% discount	X	X	–	–
A2: Free delivery	X	–	X	–
A3: No benefit	–	–	–	X

Alternatives to decision tables are **structured English**, **decision trees** and **pseudocode**.

3. System Data Requirements

3.1 Introduction and Conceptual Data Modeling

Conceptual data modeling describes the data used by the organization — the entities, their attributes and relationships — independent of any database technology. Its main output is an **Entity-Relationship Diagram (ERD)** together with a **data dictionary**. It is developed in the analysis phase and later mapped to relations in the design phase.

3.2 Gathering Information for Conceptual Data Modeling

Data requirements are gathered by asking users about **the things about which the business keeps data**, using **top-down** (starting from the business and asking for its entities) and **bottom-up** (studying existing forms, reports and files) approaches. Typical questions: What are the subjects, objects and events the business is concerned with? What is unique about each? How many are there? How are they related? What rules limit the data?

3.3 Introduction to E-R Modeling

Element	Definition	Example
Entity	A person, place, object, event or concept about which data is kept	STUDENT, COURSE
Entity type / instance	Category vs. a specific occurrence	STUDENT vs. “Sita Sharma”
Attribute	A property of an entity (simple, composite, multivalued, derived, key)	Name, Phone, Age (derived from DOB)
Identifier (key)	Attribute(s) uniquely identifying an instance	StudentID
Relationship	Association between entity types	Student ENROLLS IN Course
Degree	Number of entity types in a relationship: unary, binary, ternary	Binary: Student–Course
Cardinality	Number of instances allowed: one-to-one, one-to-many, many-to-many	M:N Student–Course
Associative entity	A relationship that has its own attributes, so it becomes an entity	ENROLLMENT (grade, date)

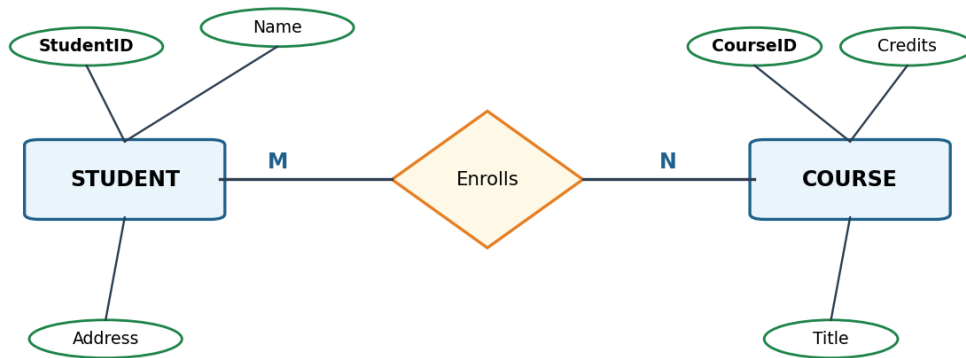


Figure 3.2 — E-R diagram of Student and Course with a many-to-many relationship

3.4 Conceptual Data Modeling and the E-R Model

Relationship **participation** may be **mandatory** (minimum cardinality one) or **optional** (minimum zero). Crow's-foot notation shows it near each entity: a bar means one, a crow's foot means many, a circle means zero. A **weak entity** depends on another (owner) entity for its identity, e.g., DEPENDENT of EMPLOYEE.

Example 3.5 — Draw an ERD (text form)

A college has departments. Each department has many teachers; each teacher belongs to one department and teaches many courses; a course may be taught by several teachers.

Solution

Entities: DEPARTMENT(DeptID, Name), TEACHER(TeacherID, Name), COURSE(CourseID, Title).

DEPARTMENT 1 : N TEACHER (“employs”) — mandatory for teacher.

TEACHER M : N COURSE (“teaches”) — becomes an associative entity

TEACHING(TeacherID, CourseID, Semester).

3.5 Representing Super-types and Sub-types

A **supertype** is a generic entity type that has relationships with one or more subtypes; a **subtype** is a subgrouping of the supertype with its own attributes. Subtypes **inherit** the attributes of the supertype. Example: supertype EMPLOYEE (EmpID, Name, Address) with subtypes HOURLY_EMPLOYEE (HourlyRate), SALARIED_EMPLOYEE (AnnualSalary) and CONSULTANT (ContractNo).

- **Total specialization** — every supertype instance must belong to a subtype; **partial** — some may not.
- **Disjoint rule** — an instance can belong to only one subtype; **overlap rule** — to more than one.
- **Subtype discriminator** — an attribute of the supertype that determines the subtype.

3.6 Business Rules

Business rules are statements that define or constrain some aspect of the business (policies, procedures, constraints). They are expressed in the data model through entities, relationships, cardinalities and constraints. Good rules are declarative, precise, atomic, consistent, expressible and business oriented. Examples: “A student can enroll in a maximum of 6 courses per semester”; “An order must have at least one line item”; “A book can be borrowed for at most 14 days.” Rules are classified as **structural assertions**, **derivations** and **action assertions**.

3.7 Packaged Conceptual Data Models — Database Patterns

Rather than model everything from scratch, analysts may reuse **packaged data models** — generic models for common business areas (e.g., customers and orders, accounting, human resources, inventory) supplied by vendors, consultants or ERP packages. Also called **database patterns** or **universal data models**, they save time, capture industry best practice and reduce errors. They must be tailored to the organization to avoid unneeded complexity. Popular patterns: **Party**, **Order/Invoice**, **Product**, **Account**, and **Contract**.

Key Takeaways

- ✓ Requirements are gathered using interviews, questionnaires, observation, document analysis, JAD, prototyping, BPR and agile user stories.
- ✓ DFDs model processes using processes, data flows, data stores and external entities; diagrams are levelled and balanced.
- ✓ Decision tables capture complex logic; n conditions give 2^n rules.
- ✓ The E-R model represents entities, attributes, relationships, cardinality and supertype/subtype hierarchies.
- ✓ Business rules and packaged data models (database patterns) help build accurate data models faster.

— End of Unit 3 —