

LEARNING MODULE

Data Flow Diagram (DFD)

Understanding DFD through the 5W1H Approach

Course: Information Systems Analysis and Design
(Analisis Desain Sistem Informasi – ADSI)

Topics: Definition • Symbols • Levels • Rules • Case Study • Exercises

Name: _____ Student ID: _____

Table of Contents

1. Module Overview	4
1.1 Description	4
1.2 Learning Objectives	4
1.3 How to Use This Module	4
1.4 Prerequisites	4
2. Introduction to Data Flow Diagrams	5
2.1 Historical Background	5
2.2 Definition	5
2.3 Logical DFD vs Physical DFD	5
3. Understanding DFD through 5W1H	7
3.1 WHAT — What Is a DFD?	7
3.2 WHY — Why Do We Use a DFD?	8
3.3 WHO — Who Is Involved with a DFD?	8
3.4 WHEN — When Is a DFD Created and Used?	9
3.5 WHERE — Where Is a DFD Used?	9
3.6 HOW — How Is a DFD Created?	9
4. DFD Components and Symbols	11
4.1 Process	11
4.2 Data Flow	11
4.3 Data Store	11
4.4 External Entity	12
5. DFD Levels and Decomposition	13
5.1 Context Diagram	13
5.2 Level 0 DFD	13
5.3 Level 1 DFD	14
5.4 Level-n DFD and Primitive DFD	15
6. Requirements for Creating a Good DFD	17
7. DFD Rules	18
8. Case Study: Campus Library Borrowing System	23
8.1 Case Description	23
8.2 Identifying the Components (5W1H in Practice)	23
8.3 Context Diagram	23
8.4 Level 0 DFD	24
8.5 Level 1 DFD for Process 3.0 (Process Return)	25
8.6 Data Flow Descriptions	25
9. Common Mistakes and Comparison with Other Diagrams	26
9.1 Common Mistakes Made by Students	26
9.2 DFD Compared with Other Modeling Tools	26

10. Summary	27
11. Exercises	28
11.1 Multiple Choice	28
11.2 Essay Questions.....	29
11.3 Practical Assignment	29
12. Answer Key and Glossary	30
12.1 Answer Key (Multiple Choice).....	30
12.2 Guidance for Essay Questions	30
12.3 Glossary	30
References	31

1. Module Overview

1.1 Description

This module discusses the **Data Flow Diagram (DFD)**, one of the most widely used modeling tools in structured systems analysis and design. It is developed from the lecture slides "Data Flow Diagram – Analisis Desain Sistem Informasi" and expands every topic in the slides: the definition and history of DFD, its symbols, the levels of decomposition (context diagram, Level 0, Level 1, and Level n), the requirements for drawing a good DFD, and the twelve rules that every valid DFD must follow.

To make the material easier to understand and remember, the module is organized around the **5W1H** framework: *What* is a DFD, *Why* is it used, *Who* makes and reads it, *When* is it created, *Where* is it used, and *How* is it drawn. The module closes with a complete case study, a list of common mistakes, a summary, exercises with an answer key, and a glossary.

1.2 Learning Objectives

After completing this module, students are expected to be able to:

1. Explain the definition, history, and purpose of a Data Flow Diagram.
2. Answer the 5W1H questions (What, Why, Who, When, Where, How) related to DFD.
3. Identify and draw the four DFD components using Gane & Sarson and DeMarco & Yourdon notations.
4. Distinguish between the context diagram, Level 0, Level 1, and Level n (primitive) DFDs.
5. Apply the requirements and the twelve rules of DFD to check whether a diagram is valid.
6. Decompose a system into a balanced set of DFDs using a real-world case study.

1.3 How to Use This Module

- Read each chapter in order, because later chapters (levels, rules, case study) build on earlier ones (symbols and definitions).
- Pay attention to the **Key Point** boxes; they summarize ideas that often appear in exams.
- Try to redraw the figures by hand or with a tool such as draw.io, Lucidchart, or Microsoft Visio.
- Complete the exercises in Chapter 11 before checking the answer key.

1.4 Prerequisites

Students should already understand basic concepts of information systems (input, process, output, storage), the System Development Life Cycle (SDLC), and requirement gathering techniques such as interviews and observation.

2. Introduction to Data Flow Diagrams

2.1 Historical Background

The use of DFD as a modeling tool was popularized in 1979 by two groups of authors: **Tom DeMarco and Edward Yourdon**, and **Chris Gane and Trish Sarson**. Both groups used DFDs as the central diagram of the **Structured Systems Analysis** methodology. Before this era, systems were often documented with long narrative text or program flowcharts, which were hard for business users to read and tended to mix *what* a system does with *how* the computer does it.

Structured analysis introduced the idea of viewing a system as a network of processes connected by flows of data. The two groups produced slightly different symbol sets, which is why today we still recognize two main notations: the Gane & Sarson notation (rounded rectangles for processes) and the DeMarco & Yourdon notation (circles, often called "bubbles"). Both notations describe exactly the same concepts.

2.2 Definition

A **Data Flow Diagram (DFD)** is a graphical representation that illustrates the flow of data within an information system — whether an existing system or a new system — from a **logical perspective**, without considering the physical environment in which the data resides. In other words, a DFD shows *where data comes from*, *what processes transform it*, *where it is stored*, and *where it goes*, but it does not show which computer, which database product, or which programming language is used.

A DFD is also a system design tool that **focuses on data flow** and is easy for programmers to understand and communicate. Because it uses only four simple symbols, it can be discussed with non-technical users as well, making it a bridge between business people and technical people.

Key Point

Logical view: a DFD describes *what* the system does with data, not *how* it is physically implemented.

Four symbols only: process, data flow, data store, and external entity.

Hierarchical: a DFD is drawn in levels, from one big process (context diagram) down to detailed primitive processes.

2.3 Logical DFD vs Physical DFD

Although the slides emphasize the logical perspective, analysts actually distinguish two kinds of DFD. A **logical DFD** focuses on business activities and the data they need. A **physical DFD** shows how the system will be implemented — including hardware, software, files, and people. Analysts usually draw the logical model first to understand the business, then derive the physical model during design.

Table 1. *Comparison of logical and physical DFD*

Aspect	Logical DFD	Physical DFD
Focus	What the business does	How the system is implemented
Process names	Business activities (e.g., "Validate Order")	Programs, people, or devices (e.g., "Order Entry Clerk", "validate.php")
Data stores	Collections of data regardless of medium	Actual files or database tables (e.g., "tbl_order in MySQL")
Stability	Stable; changes only when the business changes	Changes when technology changes
Main audience	Users and analysts	Designers and programmers

3. Understanding DFD through 5W1H

The 5W1H framework is a set of six basic questions used by journalists, researchers, and analysts to understand any topic completely. Applying the framework to DFD gives a full picture of the tool — not just its symbols, but also its purpose, users, timing, context, and method.

Table 2. 5W1H summary of Data Flow Diagrams

Question	Short Answer
What	A graphical model that shows how data flows through processes, stores, and external entities in a system.
Why	To understand, communicate, verify, and document system requirements in a way everyone can read.
Who	Created by systems analysts; read and validated by users, managers, designers, and programmers.
When	Mainly during the analysis phase of the SDLC, refined in the design phase, and updated during maintenance.
Where	In requirement documents, system proposals, CASE tools, software projects, audits, and business process improvement.
How	By identifying entities, flows, processes, and stores; drawing a context diagram; then decomposing it level by level while following the DFD rules.

3.1 WHAT — What Is a DFD?

A DFD is a **process model**: a picture of the movement of data between external entities, processes, and data stores within a system. It answers questions such as "Which data enters the system?", "What happens to that data?", "Where is it kept?", and "Which reports or outputs leave the system?".

What a DFD **shows**:

- The sources and destinations of data (external entities).
- The processes that transform input data into output data.
- The places where data is kept for later use (data stores).
- The paths that data follows (data flows) and the name of each piece of data moving along a path.

What a DFD **does not show**:

- Timing or sequence of events (a DFD is not a flowchart; it has no start/stop or loop control).
- Decision logic in detail (this is described later using structured English, decision tables, or decision trees).
- Physical technology such as servers, databases, or programming languages (in a logical DFD).

- The structure of stored data and relationships between tables (this is the job of an Entity Relationship Diagram).

3.2 WHY — Why Do We Use a DFD?

Organizations use DFDs because a picture of the data is much easier to discuss than pages of text. The main reasons are:

1. **Understanding the current system.** By drawing the "as-is" DFD, analysts discover how data really moves in the organization, including duplicated work, bottlenecks, and missing data.
2. **Communication.** DFDs use only four symbols, so users, managers, and programmers can share the same picture of the system and agree on requirements.
3. **Defining system boundaries.** The context diagram clearly shows what is inside the system and what is outside, which prevents "scope creep" in a project.
4. **Managing complexity through decomposition.** Large systems can be broken into smaller levels, so each diagram stays simple and readable.
5. **Finding errors early.** Rules such as "no process with only inputs" help detect missing outputs, unnecessary data, or illogical flows before any code is written.
6. **Foundation for design.** Processes in a primitive DFD become program modules, data stores become database tables, and data flows become screens, forms, reports, or APIs.
7. **Documentation.** DFDs remain useful during maintenance, auditing, and training of new staff.

3.3 WHO — Who Is Involved with a DFD?

Table 3. Stakeholders of a DFD

Role	Involvement with DFD
Systems analyst	Main creator. Gathers requirements, draws the DFDs, checks balancing and rules, and maintains the documentation.
End users	Provide information about how the business works and validate that the DFD represents their activities correctly.
Managers / system owners	Review the context diagram and Level 0 to confirm scope, objectives, and reports they need.
System designers	Transform the logical DFD into a physical design: database, interfaces, and program structure.
Programmers	Use primitive DFDs and process specifications to write the program modules.
Testers and auditors	Use data flows to design test cases and to trace how data is processed and controlled.

It is also important to understand the "who" *inside* the diagram: people or organizations that interact with the system are drawn as **external entities** (e.g., Customer, Supplier, Bank). In a

logical DFD, the internal staff who perform a process are usually not drawn as entities, because their work is represented by the process itself.

3.4 WHEN — When Is a DFD Created and Used?

DFDs are most closely associated with the **analysis phase** of the System Development Life Cycle, but they appear in several phases:

Table 4. Use of DFD in each SDLC phase

SDLC Phase	Use of DFD
Planning	A rough context diagram helps define project scope and feasibility.
Analysis	Analysts draw "as-is" DFDs of the current system and "to-be" DFDs of the proposed system. This is the main phase for DFD.
Design	Logical DFDs are refined into physical DFDs; processes are mapped to modules and stores to tables.
Implementation & Testing	DFDs guide program development and help create test scenarios for each data flow.
Maintenance	DFDs are updated whenever the system changes, so the documentation stays accurate.

A DFD is especially suitable when the system is **data-intensive** (e.g., transaction processing, inventory, payroll, academic administration) and when the project follows a structured or waterfall methodology. For systems designed with an object-oriented approach, UML diagrams are often preferred, although DFDs are still used to explain data movement at a high level.

3.5 WHERE — Where Is a DFD Used?

- **In documentation:** System Requirement Specifications (SRS), project proposals, theses, and final-year projects in information systems.
- **In industry sectors:** banking (transaction processing), retail and restaurants (ordering and inventory), health care (patient registration), education (academic information systems), and government (public services).
- **In tools:** CASE (Computer-Aided Software Engineering) tools and diagramming software such as Microsoft Visio, draw.io/diagrams.net, Lucidchart, Visual Paradigm, and Enterprise Architect.
- **In security and auditing:** threat modeling (e.g., the STRIDE method) uses DFDs to find where sensitive data crosses trust boundaries.
- **In business process improvement:** DFDs show redundant data handling, which can be eliminated to make processes efficient.

3.6 HOW — How Is a DFD Created?

Drawing a DFD is an iterative activity. The general steps are:

1. **Gather requirements.** Collect information through interviews, questionnaires, observation, and review of existing forms and reports.
2. **Identify external entities.** List every person, organization, or other system outside the boundary that sends data to or receives data from the system.
3. **Identify inputs and outputs.** For each entity, list the data it sends (e.g., order) and receives (e.g., receipt).
4. **Draw the context diagram.** Place a single process numbered 0 in the center representing the whole system and connect it to the entities with named data flows. No data stores appear here.
5. **Identify major processes and data stores.** Break the system into its main functions (usually 3–9 processes) and determine what data must be stored.
6. **Draw the Level 0 DFD.** Number the processes 1.0, 2.0, 3.0, and so on, add data stores (D1, D2, ...), and make sure all context-level flows still appear (balancing).
7. **Decompose further.** Explode complex processes into Level 1 (e.g., 4.1, 4.2), Level 2 (e.g., 4.3.1), and so on, until each process is simple enough to be described in a short process specification. These are primitive DFDs.
8. **Check the rules and balancing.** Verify every diagram against the twelve DFD rules in Chapter 7 and confirm that inputs and outputs of a parent process match those of its child diagram.
9. **Review with users.** Walk through the diagrams with stakeholders, correct misunderstandings, and repeat until everyone agrees.

Key Point: Balancing

Balancing means that the data flows entering and leaving a process on a parent diagram must be exactly the same as the data flows entering and leaving its child diagram. For example, if Process 4.0 receives "Daily Goods Sold Amounts" and "Daily Inventory Depletion Amounts" and produces "Management Reports" in Level 0, then its Level 1 diagram must also have exactly those inputs and that output.

4. DFD Components and Symbols

Every DFD is built from only four components. Figure 1 (taken from the lecture slides) shows each component, the information a CASE tool usually records about it, and its symbol in both Gane & Sarson and DeMarco & Yourdon notations.

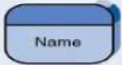
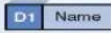
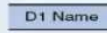
	Data Flow Diagram Element	Typical Computer-Aided Software Engineering Fields	Gane and Sarson Symbol	DeMarco and Yourdon Symbol
Process	Every <i>process</i> has A number A name (verb phrase) A description One or more output data flows Usually one or more input data flows	Label (name) Type (process) Description (what is it) Process number Process description (Structured English) Notes		
Data flow	Every <i>data flow</i> has A name (a noun) A description One or more connections to a process	Label (name) Type (flow) Description Alias (another name) Composition (description of data elements) Notes		
Data store	Every <i>data store</i> has A number A name (a noun) A description One or more input data flows Usually one or more output data flows	Label (name) Type (store) Description Alias (another name) Composition (description of data elements) Notes		
External entity	Every <i>external entity</i> has A name (a noun) A description	Label (name) Type (entity) Description Alias (another name) Entity description Notes		

Figure 1. DFD elements, CASE fields, and symbols in two notations (source: lecture slides)

4.1 Process

A process represents work performed on data: it transforms incoming data flows into outgoing data flows. Every process has a **number**, a **name** written as a *verb phrase* (e.g., "Calculate Fine", "Validate Order"), a description, one or more output data flows, and usually one or more input data flows. In Gane & Sarson notation it is a rounded rectangle with the number at the top; in DeMarco & Yourdon notation it is a circle.

4.2 Data Flow

A data flow is data in motion, drawn as an arrow. It has a **name written as a noun** (e.g., "Customer Order", "Receipt"), a description, and one or more connections to a process. The arrow head shows the direction of movement. Data flows may carry an alias and a composition (the list of data elements it contains, e.g., Customer Order = order number + date + customer name + items).

4.3 Data Store

A data store is data at rest — a place where data is held for later use by one or more processes. It has a **number** (D1, D2, ...), a **name written as a noun** (e.g., "Inventory File"), a description, and usually both input and output data flows. In Gane & Sarson notation it is an open-ended rectangle with the identifier in a small box on the left; in DeMarco & Yourdon notation it is two parallel lines.

4.4 External Entity

An external entity (also called a *source* or *sink*) is a person, organization, department, or another system outside the system boundary that supplies or receives data. It has a **name as a noun** (e.g., "Customer", "Kitchen", "Bank") and a description. It is drawn as a rectangle (often with a shadow). The system has no control over what an entity does internally.

Table 5. Summary of DFD components

Component	Naming	Gane & Sarson	DeMarco & Yourdon	Example
Process	Verb phrase + number	Rounded rectangle	Circle (bubble)	1.0 Receive Customer Order
Data flow	Noun	Arrow	Arrow	Customer Order
Data store	Noun + D-number	Open-ended rectangle	Two parallel lines	D1 Inventory File
External entity	Noun	Rectangle	Rectangle	Customer, Kitchen

5. DFD Levels and Decomposition

A real system is too complex to show in one diagram. DFDs therefore use **functional decomposition**: a process is "exploded" into a more detailed diagram at the next level. The lecture slides use the classic Hoosier Burger food ordering system (Valacich & George) to illustrate each level.

5.1 Context Diagram

The context diagram is the highest-level view. It shows the **system boundary**, the **external entities** that interact with the system, and the **major information flows** between the entities and the system. It contains **only one process symbol** (numbered 0) and **no data stores**.

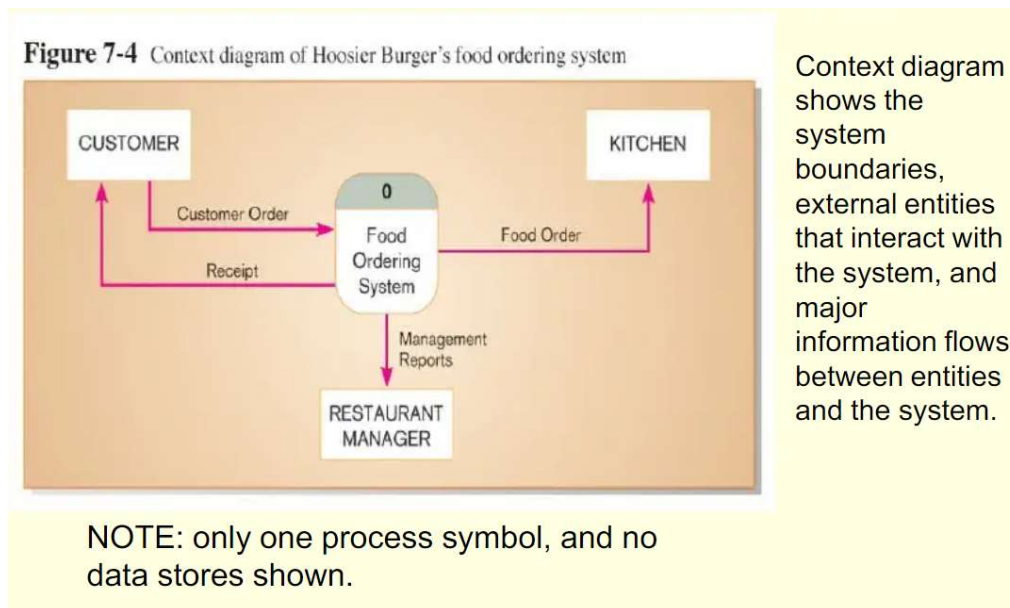
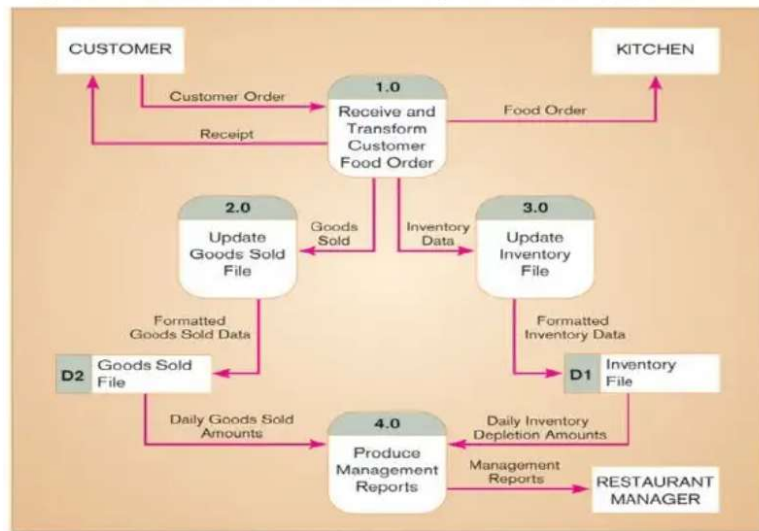


Figure 2. Context diagram of Hoosier Burger's food ordering system (source: lecture slides)

In Figure 2, the Customer sends a Customer Order and receives a Receipt; the system sends a Food Order to the Kitchen and Management Reports to the Restaurant Manager. Everything inside the process "0 Food Ordering System" is still hidden.

5.2 Level 0 DFD

The Level 0 DFD (also called the *overview diagram*) shows the system's **major processes, data flows, and data stores** at a high level of abstraction. It is the explosion of process 0 from the context diagram. Processes are labeled 1.0, 2.0, 3.0, and so on; each can later be decomposed into lower-level DFDs.

Figure 7-5 Level-0 DFD of Hoosier Burger's food ordering system

Level-0 DFD shows the system's major processes, data flows, and data stores at a high level of abstraction.

Processes are labeled 1.0, 2.0, etc. These will be decomposed into more primitive (lower-level) DFDs.

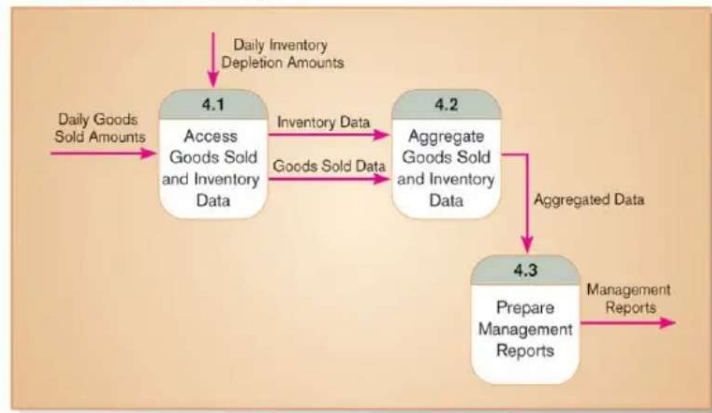
Figure 3. Level-0 DFD of Hoosier Burger's food ordering system (source: lecture slides)

Notice that the Level 0 diagram is **balanced** with the context diagram: Customer Order, Receipt, Food Order, and Management Reports still appear with the same entities. New elements — the data stores D1 Inventory File and D2 Goods Sold File, and the internal flows between processes — are added only inside the boundary.

5.3 Level 1 DFD

A Level 1 DFD shows the **sub-processes of one process** in the Level 0 DFD. Figure 4 decomposes Process 4.0 "Produce Management Reports" into 4.1 Access Goods Sold and Inventory Data, 4.2 Aggregate Goods Sold and Inventory Data, and 4.3 Prepare Management Reports. The processes are labeled 4.1, 4.2, etc., and can be decomposed further if necessary.

Figure 7-8 Level-1 diagram showing the decomposition of Process 4.0 from the level-0 diagram for Hoosier Burger's food ordering system



Level-1 DFD shows the sub-processes of one of the processes in the Level-0 DFD.

This is a Level-1 DFD for Process 4.0.

Processes are labeled 4.1, 4.2, etc. These can be further decomposed in more primitive (lower-level) DFDs if necessary.

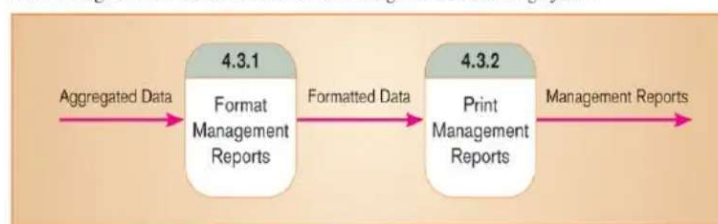
Figure 4. Level-1 DFD showing the decomposition of Process 4.0 (source: lecture slides)

The inputs (Daily Goods Sold Amounts, Daily Inventory Depletion Amounts) and output (Management Reports) are exactly the same as those of Process 4.0 in Level 0 — the diagrams are balanced.

5.4 Level-n DFD and Primitive DFD

A Level-n DFD shows the sub-processes of one of the processes in the Level n-1 DFD. Figure 5 is a Level-2 DFD for Process 4.3, containing 4.3.1 Format Management Reports and 4.3.2 Print Management Reports. If this is the lowest level of the hierarchy, it is called a **primitive DFD**.

Figure 7-9 Level-2 diagram showing the decomposition of Process 4.3 from the level-1 diagram for Process 4.0 for Hoosier Burger's food ordering system



Level-n DFD shows the sub-processes of one of the processes in the Level n-1 DFD.

This is a Level-2 DFD for Process 4.3.

Processes are labeled 4.3.1, 4.3.2, etc. If this is the lowest level of the hierarchy, it is called a *primitive DFD*.

Figure 5. Level-2 DFD showing the decomposition of Process 4.3 (source: lecture slides)

When should decomposition stop? A process is usually considered primitive when it performs a single, well-defined function, can be described in about one page of structured English, has a small number of inputs and outputs, and would not become clearer if broken down further.

Table 6. *Comparison of DFD levels*

Level	Content	Process numbering	Data stores?
Context diagram	Whole system as one process, all external entities	0	No
Level 0	Major processes of the system	1.0, 2.0, 3.0 ...	Yes
Level 1	Sub-processes of one Level 0 process	4.1, 4.2, 4.3 ...	Yes (if used)
Level 2 ... n	Sub-processes of one Level n-1 process	4.3.1, 4.3.2 ...	Yes (if used)
Primitive DFD	Lowest level; processes are not decomposed further	Deepest number	Yes (if used)

6. Requirements for Creating a Good DFD

A properly designed DFD should be **visually appealing, easy to read, and easy for users to understand**. The lecture slides list five requirements, each explained below with practical guidance.

1) Use appropriate names for each DFD component

Processes use strong verb + object phrases ("Calculate Fine", not "Fine" or "Processing"). Data flows, data stores, and entities use clear nouns ("Loan Receipt", "Member File", "Librarian"). Avoid vague names such as "Data", "Information", or "Process 1". Two different flows should never share the same name unless they carry exactly the same data.

2) Number the process components appropriately

Numbers show the position of a process in the hierarchy: 0 for the context diagram, 1.0–n.0 for Level 0, 3.1–3.n for the children of process 3.0, and 3.2.1 for the children of process 3.2. Numbers identify a process; they do not indicate the order of execution. Data stores are numbered D1, D2, and so on, and keep the same number on every level.

3) Arrange the DFD clearly and in a visually pleasing manner

Place entities that supply data on the left or top and entities that receive outputs on the right or bottom. Minimize crossing lines; if necessary, a data store or entity may be drawn twice (marked with an extra line or asterisk) to avoid long crossing arrows. Keep symbol sizes consistent and label every arrow.

4) Avoid creating overly complex DFD diagrams

A common guideline is to have about **5 to 9 processes per diagram** ("seven plus or minus two"). If a diagram has more, group related processes and move the detail to a lower level. Each diagram should fit on one page.

5) Ensure that the DFD accurately represents the system

The diagram must reflect the real (or planned) system: every input used, every output produced, every store that really exists. Validate it with users, check balancing between levels, and make sure no required data flow is missing.

Checklist Before Submitting a DFD

- ☐ Every process has a number and a verb-phrase name.
- ☐ Every data flow is labeled with a noun and has one direction.
- ☐ Every data store is numbered and has at least one input and one output across the set of diagrams.
- ☐ The context diagram has one process and no data stores.
- ☐ Each child diagram is balanced with its parent process.
- ☐ None of the twelve rules in Chapter 7 is violated.

7. DFD Rules

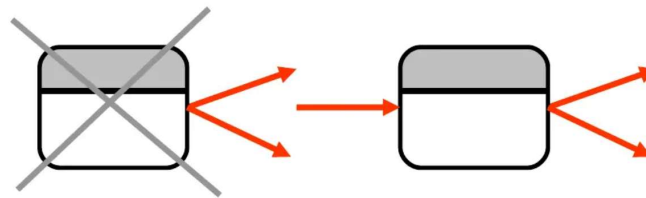
The following twelve rules, presented in the lecture slides, determine whether a DFD is syntactically correct. In the figures, a crossed diagram (X) is **incorrect**, while the diagram on the right is the **correct** version. Black rectangles represent external entities, rounded rectangles represent processes, and open rectangles represent data stores.

Table 7. Summary of the twelve DFD rules

No.	Rule	Category
1	No process can have only outputs.	Process
2	No process can have only inputs.	Process
3	Data cannot move directly from one data store to another.	Data store / Entity
4	Data cannot move directly from an external entity to a data store.	Data store / Entity
5	Data cannot move directly from a data store to an external entity (sink).	Data store / Entity
6	Data cannot move directly from an external source to an external sink.	Data store / Entity
7	A data flow has only one direction of flow between symbols.	Data flow
8	A fork in a data flow means that exactly the same data goes from a common location to two or more different processes.	Data flow
9	A join in a data flow means that exactly the same data comes from two or more different processes or data stores to a common location.	Data flow
10	No recursive data flow on a process.	Data flow
11	A data flow to a data store means update (insert, delete, or change).	Data store access
12	A data flow from a data store means retrieve or use.	Data store access

Rule 1: No process can have only outputs.

A process that produces outputs without any input is called a **miracle**: data appears from nowhere. Every process must receive the data it needs to create its outputs.



No process can have only outputs

Figure 6. Rule 1 — No process can have only outputs

Rule 2: No process can have only inputs.

A process that receives data but produces nothing is called a **black hole**. If a process only receives data, it is probably really an external entity (sink) or a data store, or an output flow is missing.

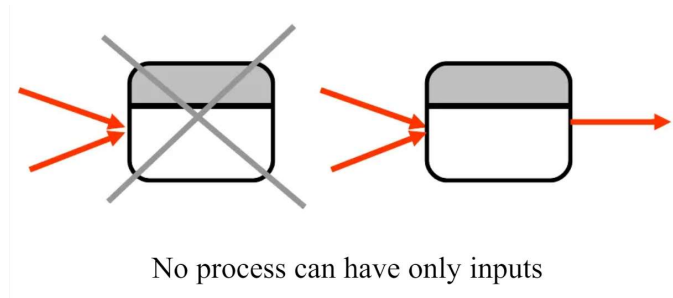
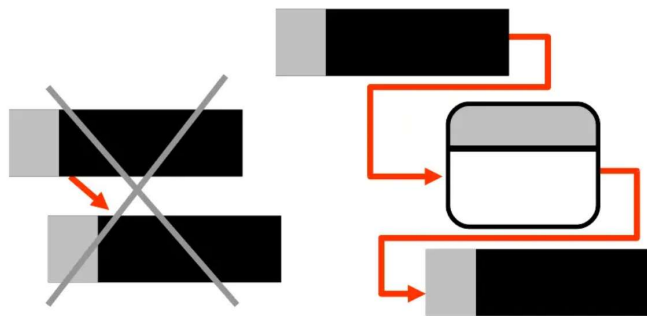


Figure 7. Rule 2 — No process can have only inputs

Rule 3: Data cannot move directly from one data store to another.

Data stores are passive; they cannot move data by themselves. Data must pass through a process that reads it from one store and writes it to another (for example, "Archive Old Loans").

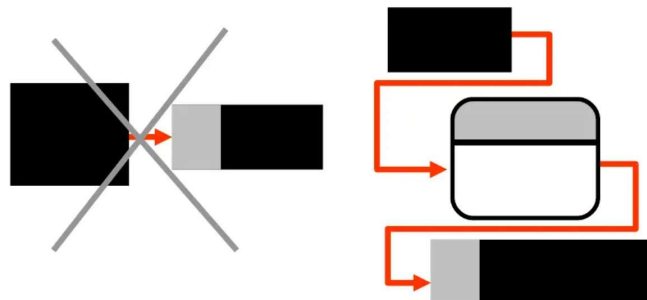


Data cannot move directly from one data store to another

Figure 8. Rule 3 — Data cannot move directly from one data store to another

Rule 4: Data cannot move directly from an external entity to a data store.

An outside party cannot write into the system's storage without the system doing something. A process must receive the data from the entity and then store it.

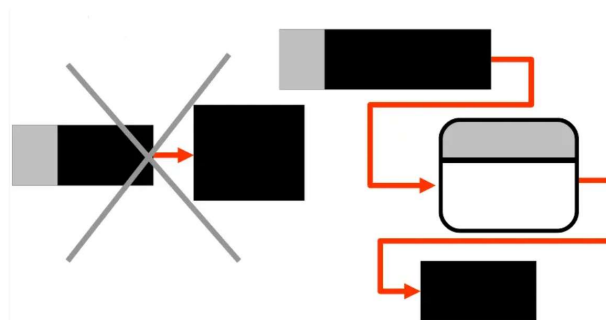


Data cannot move directly from an external entity to a data store

Figure 9. Rule 4 — Data cannot move directly from an external entity to a data store

Rule 5: Data cannot move directly from a data store to an external entity (sink).

Likewise, an entity cannot read stored data directly. A process must retrieve the data and deliver it, for example as a report or notification.

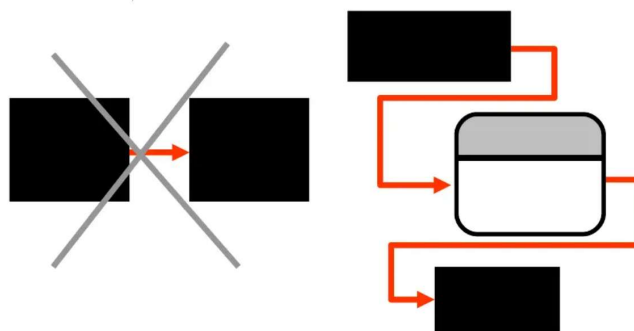


Data cannot move directly from a data store to an external entity (sink)

Figure 10. Rule 5 — Data cannot move directly from a data store to an external entity (sink)

Rule 6: Data cannot move directly from an external source to an external sink.

Data exchanged between two entities happens outside the system and therefore is not part of the DFD. If the system is involved, a process must be between them.

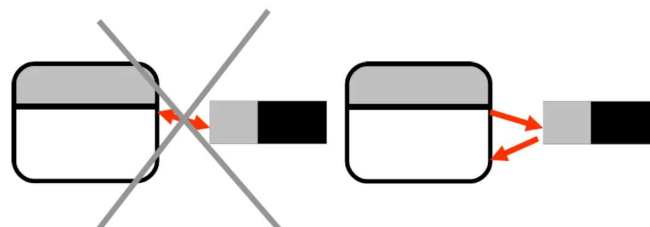


Data cannot move directly from an external source to an external sink

Figure 11. Rule 6 — Data cannot move directly from an external source to an external sink

Rule 7: A data flow has only one direction of flow between symbols.

Do not use double-headed arrows. If data moves in both directions (e.g., a process reads from and updates a store), draw two separate arrows, each with its own name, because they usually carry different data at different times.



A data flow has only one direction of flow between symbols

Figure 12. Rule 7 — A data flow has only one direction of flow between symbols

Rule 8: A fork in a data flow means that exactly the same data goes from a common location to two or more different processes.

A split arrow must carry the same data (A and A). If different data (A and B) goes to different processes, draw two separate data flows with different names.

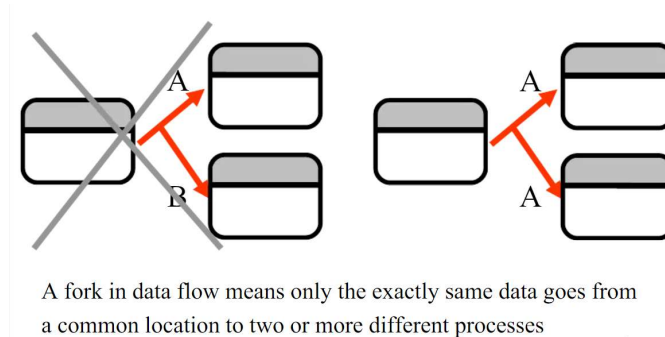


Figure 13. Rule 8 — A fork in a data flow means that exactly the same data goes from a common location to two or more different processes

Rule 9: A join in a data flow means that exactly the same data comes from two or more different processes or data stores to a common location.

Similarly, a joining arrow must carry identical data from each source. Different data must be drawn as separate flows.

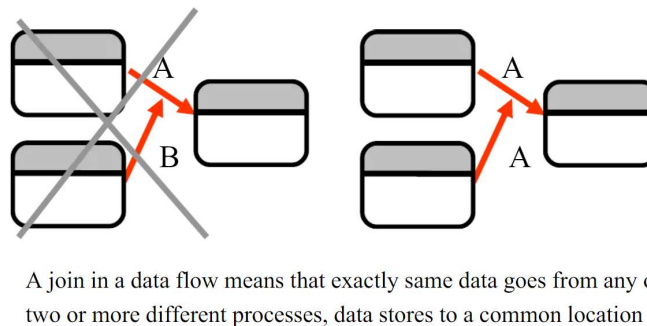


Figure 14. Rule 9 — A join in a data flow means that exactly the same data comes from two or more different processes or data stores to a common location

Rule 10: No recursive data flow on a process.

A data flow cannot leave a process and return directly to the same process. Repeated processing is an issue of logic, not data movement; such loops must involve other processes, as shown on the right of the figure.

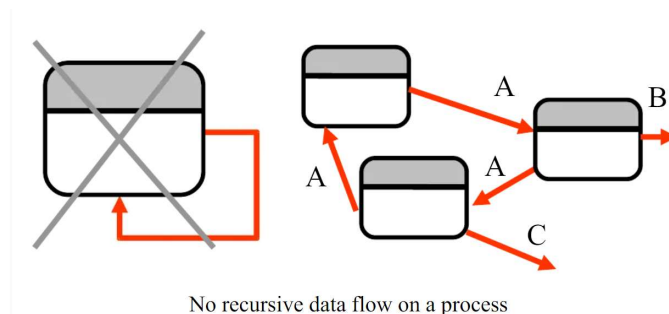


Figure 15. Rule 10 — No recursive data flow on a process

Rule 11: A data flow to a data store means update (insert, delete, or change).

An arrow pointing into a data store indicates that the process writes data — adding new records, changing existing ones, or deleting them.



A data flow to a data store means update (delete or change)

Figure 16. Rule 11 — A data flow to a data store means update (insert, delete, or change)

Rule 12: A data flow from a data store means retrieve or use.

An arrow pointing out of a data store indicates that the process reads data from it without changing it.



A data flow from a data store means retrieve or use

Figure 17. Rule 12 — A data flow from a data store means retrieve or use

Special Error Types to Remember

Black hole – a process with inputs but no outputs (violates Rule 2).

Miracle – a process with outputs but no inputs (violates Rule 1).

Grey hole – a process whose inputs are clearly insufficient to produce its outputs (e.g., "Calculate Salary" receives only "Employee Name").

8. Case Study: Campus Library Borrowing System

To apply all the concepts, this chapter develops a complete set of DFDs for a new case: a campus library borrowing system. The diagrams use the Gane & Sarson notation.

8.1 Case Description

A campus library wants to computerize its borrowing activities. Students and lecturers must **register** as members and receive a membership card. A member can **borrow** books by submitting a borrow request; the system checks that the member is valid and the book is available, records the loan, and gives a loan receipt. When the member **returns** a book, the system checks the loan record, calculates a fine if the book is late, sends a fine notice, and updates the loan and book status. The **librarian** enters new book data and can view book availability. Every month the **library head** receives a report of borrowing activity.

8.2 Identifying the Components (5W1H in Practice)

Table 8. Components of the library borrowing system

Component	Items identified
External entities (Who)	Member, Librarian, Library Head
Inputs (What comes in)	Registration Data, Borrow Request, Returned Book Data, New Book Data
Outputs (What goes out)	Membership Card, Loan Receipt, Fine Notice, Book Availability Info, Monthly Report
Major processes (How)	1.0 Register Member, 2.0 Process Borrowing, 3.0 Process Return, 4.0 Maintain Book Catalog, 5.0 Produce Reports
Data stores (Where data is kept)	D1 Member File, D2 Book File, D3 Loan File

8.3 Context Diagram

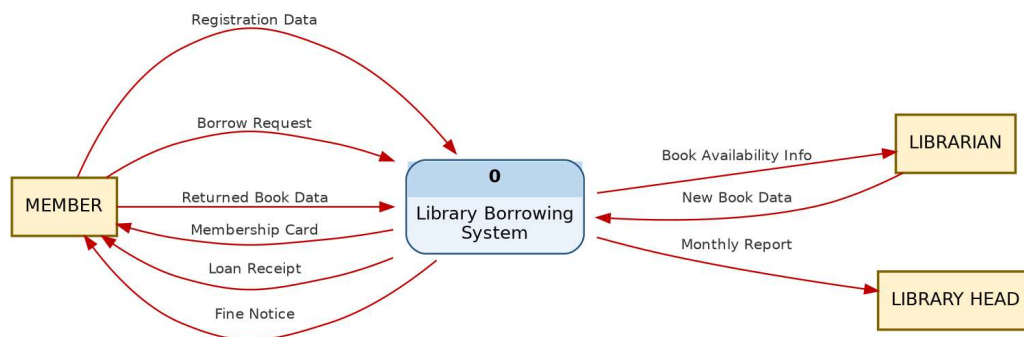


Figure 18. Context diagram of the Library Borrowing System

The context diagram contains a single process "0 Library Borrowing System", three external entities, and all inputs and outputs listed in Table 8. There are no data stores.

8.4 Level 0 DFD

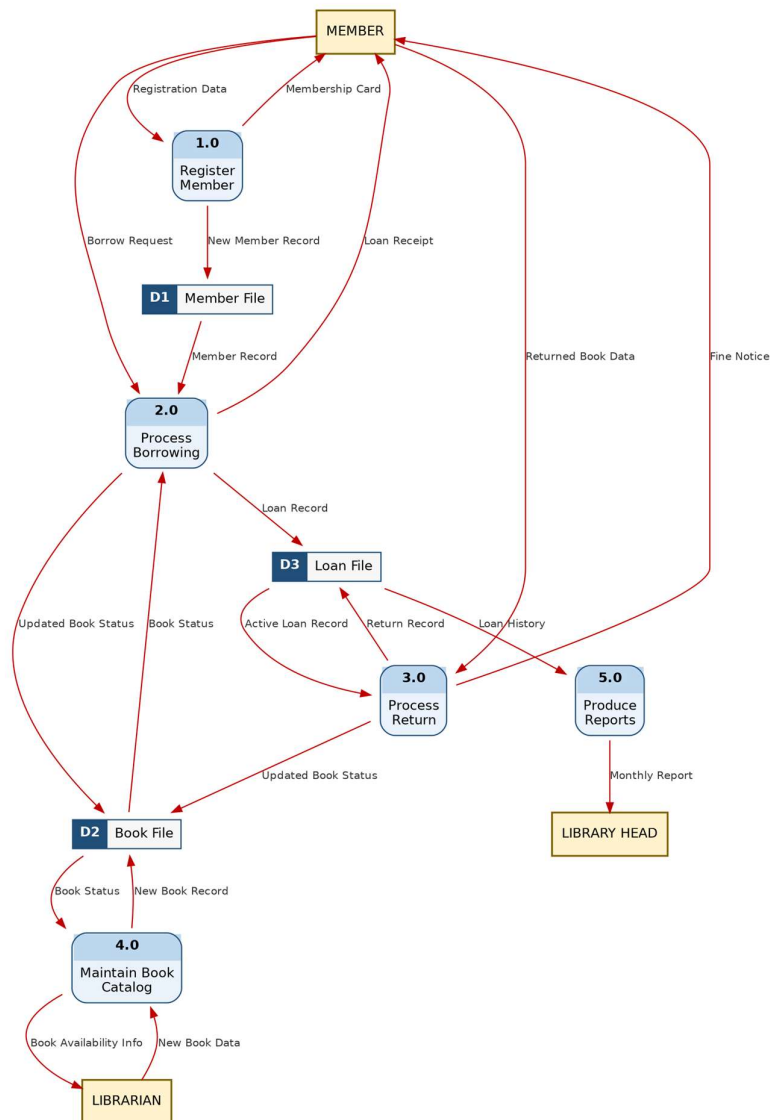


Figure 19. Level-0 DFD of the Library Borrowing System

The Level 0 DFD decomposes process 0 into five major processes. Check the balancing: every flow from the context diagram (Registration Data, Borrow Request, Returned Book Data, Membership Card, Loan Receipt, Fine Notice, New Book Data, Book Availability Info, and Monthly Report) appears again with the same entity. Also check the rules: no data store is connected directly to another store or to an entity, and every process has at least one input and one output.

8.5 Level 1 DFD for Process 3.0 (Process Return)

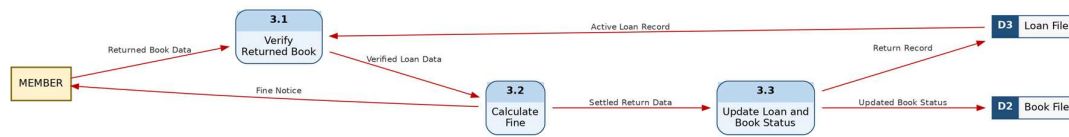


Figure 20. Level-1 DFD of Process 3.0 Process Return

Process 3.0 is decomposed into 3.1 Verify Returned Book, 3.2 Calculate Fine, and 3.3 Update Loan and Book Status. The external entity and data stores are drawn for readability. The diagram is balanced with Level 0: its inputs are Returned Book Data and Active Loan Record, and its outputs are Fine Notice, Return Record, and Updated Book Status — exactly the same as Process 3.0. The flows "Verified Loan Data" and "Settled Return Data" are new internal flows that exist only at this level.

8.6 Data Flow Descriptions

Table 9. Examples of data flow composition

Data flow	From → To	Composition (data elements)
Registration Data	Member → 1.0	name + student/staff ID + faculty + phone + email
Borrow Request	Member → 2.0	member ID + book code(s) + request date
Loan Record	2.0 → D3	loan ID + member ID + book code + loan date + due date
Returned Book Data	Member → 3.0	loan ID or member ID + book code + return date
Fine Notice	3.2 → Member	member ID + book code + days late + fine amount
Monthly Report	5.0 → Library Head	period + total loans + top books + total fines + overdue list

9. Common Mistakes and Comparison with Other Diagrams

9.1 Common Mistakes Made by Students

Table 10. Common DFD mistakes and how to fix them

Mistake	Why it is wrong	Correction
Drawing data stores in the context diagram	The context diagram shows only the boundary; stores are internal	Move stores to Level 0 or below
Treating a DFD like a flowchart (Start, End, IF decisions)	DFD shows data movement, not control flow	Remove control symbols; describe logic in process specifications
Unlabeled arrows	The reader cannot know what data moves	Name every data flow with a noun
Process named with a noun ("Report")	Processes represent actions	Use a verb phrase ("Produce Report")
Unbalanced levels	Child diagram adds or loses external flows	Match inputs/outputs of the parent process exactly
Entity connected directly to a data store	Violates Rules 4 and 5	Insert a process between them
Too many processes in one diagram	Diagram becomes unreadable	Group into 5–9 processes and decompose

9.2 DFD Compared with Other Modeling Tools

Table 11. DFD compared with other diagrams

Aspect	DFD	Flowchart	ERD	UML Activity / Use Case
Main focus	Movement of data	Sequence of steps and control logic	Structure of stored data and relationships	User interactions and workflow (object-oriented)
Shows timing/order?	No	Yes	No	Yes (activity diagram)
Shows decisions?	No	Yes	No	Yes
Typical user	Analyst and users	Programmer	Database designer	OO analyst/designer
Relation to DFD	—	Can detail a primitive process	Details the contents of DFD data stores	Alternative in OO projects

In practice these tools complement one another. A DFD shows *what data moves*, an ERD shows *how stored data is structured*, and process specifications (structured English, decision tables) or flowcharts show *how each primitive process works internally*.

10. Summary

- A **Data Flow Diagram** is a logical model of how data flows through an information system. It was popularized in 1979 by DeMarco & Yourdon and Gane & Sarson as part of Structured Systems Analysis.
- **5W1H**: DFD is a data-flow model (*What*) used to understand, communicate, and document requirements (*Why*); it is created by analysts and validated by users (*Who*); mainly during analysis and design (*When*); in requirement documents, CASE tools, and many industries (*Where*); by identifying entities, drawing a context diagram, and decomposing level by level (*How*).
- A DFD uses four components: **process** (verb phrase, numbered), **data flow** (noun, arrow), **data store** (noun, D-number), and **external entity** (noun, rectangle).
- DFD levels: the **context diagram** (one process, no stores), **Level 0** (major processes), **Level 1** (sub-processes of one Level 0 process), and **Level n** down to the **primitive DFD**. All levels must be **balanced**.
- A good DFD uses appropriate names, correct numbering, a clear layout, reasonable complexity, and an accurate representation of the system.
- Twelve rules govern processes (no miracles or black holes), data stores and entities (data must pass through a process), and data flows (one direction, same data on forks and joins, no recursion, into store = update, out of store = retrieve).

11. Exercises

11.1 Multiple Choice

Choose the best answer.

- 1. Who popularized the DFD as a modeling tool in 1979?**
 - A. Peter Chen and James Martin
 - B. DeMarco & Yourdon and Gane & Sarson
 - C. Booch, Rumbaugh and Jacobson
 - D. Codd and Date
- 2. A DFD describes a system from which perspective?**
 - A. Physical
 - B. Logical
 - C. Chronological
 - D. Hardware
- 3. Which component is named with a verb phrase?**
 - A. Data flow
 - B. Data store
 - C. Process
 - D. External entity
- 4. Which of the following is TRUE about a context diagram?**
 - A. It contains several data stores
 - B. It contains exactly one process and no data stores
 - C. Its processes are numbered 1.0, 2.0, 3.0
 - D. It shows only internal data flows
- 5. Processes in the child diagram of process 4.3 are numbered...**
 - A. 4.1, 4.2
 - B. 4.3.1, 4.3.2
 - C. 43.1, 43.2
 - D. 5.1, 5.2
- 6. A process that has inputs but no outputs is called...**
 - A. Miracle
 - B. Grey hole
 - C. Black hole
 - D. Primitive
- 7. A data flow directed INTO a data store means...**
 - A. Retrieve or use
 - B. Update (insert, delete, or change)
 - C. Delete the store
 - D. Print a report
- 8. Which connection is ALLOWED in a DFD?**
 - A. Data store → Data store
 - B. External entity → Data store
 - C. Process → Data store
 - D. External entity → External entity

9. "Balancing" in DFD means...

- A. Every diagram has the same number of processes
- B. Inputs and outputs of a parent process equal those of its child diagram
- C. Entities are placed symmetrically
- D. Every store has equal reads and writes

10. In which SDLC phase is the DFD mainly produced?

- A. Implementation
- B. Analysis
- C. Maintenance
- D. Deployment

11.2 Essay Questions

1. Explain the difference between a logical DFD and a physical DFD, and give one example of a process name for each.
2. Using the 5W1H framework, explain why DFD is still relevant for students of information systems today.
3. Look at Figure 3 (Level-0 Hoosier Burger). Identify all external entities, processes, data stores, and data flows, and prove that it is balanced with the context diagram in Figure 2.
4. A student draws an arrow from the entity "Customer" directly to the data store "D1 Customer File". Which rule is violated, and how should the diagram be corrected?
5. Explain the meaning of a fork and a join in a data flow, and when they are allowed.

11.3 Practical Assignment

Online Course Registration System. Students log in and submit a course registration form. The system checks the student's payment status from the Finance Department, checks course quotas, saves the registration, and sends a Study Plan Card (KRS) to the student. The academic advisor receives a list of registrations to approve and sends an approval decision. At the end of the registration period the Academic Office receives a registration summary report.

Tasks:

1. List the external entities, main inputs, and main outputs.
2. Draw the context diagram.
3. Draw a Level 0 DFD with 4–6 processes and the necessary data stores.
4. Choose one process and decompose it into a Level 1 DFD.
5. Check all diagrams against the twelve rules and write a short balancing check.

12. Answer Key and Glossary

12.1 Answer Key (Multiple Choice)

No.	1	2	3	4	5	6	7	8	9	10
Answer	B	B	C	B	B	C	B	C	B	B

12.2 Guidance for Essay Questions

- **Q1:** Logical = business activity ("Verify Payment"); physical = implementation ("Cashier checks payment in SIAKAD").
- **Q3:** Entities: Customer, Kitchen, Restaurant Manager. Processes: 1.0–4.0. Stores: D1 Inventory File, D2 Goods Sold File. Balanced because Customer Order, Receipt, Food Order, and Management Reports appear at both levels with the same entities.
- **Q4:** Rule 4 is violated; add a process such as "1.0 Register Customer" between the entity and the data store.
- **Q5:** A fork splits identical data to several destinations; a join combines identical data from several sources. Different data must be drawn as separate flows.

12.3 Glossary

Table 12. Glossary of DFD terms

Term	Meaning
Balancing	Consistency of inputs and outputs between a parent process and its child diagram.
Black hole	An invalid process that has inputs but no outputs.
CASE tool	Computer-Aided Software Engineering software that supports drawing and documenting system models.
Context diagram	The top-level DFD showing the whole system as one process with its external entities.
Data dictionary	A repository describing every data flow, data store, and data element in the DFDs.
Decomposition	Breaking a process into more detailed sub-processes on a lower-level diagram.
Grey hole	A process whose inputs are insufficient to produce its outputs.
Miracle	An invalid process that has outputs but no inputs.
Primitive DFD	The lowest-level DFD, whose processes are not decomposed further.
Sink / Source	An external entity that receives data (sink) or supplies data (source).
Structured English	A restricted form of English used to describe the logic of primitive processes.

References

- DeMarco, T. (1979). *Structured Analysis and System Specification*. Yourdon Press.
- Gane, C., & Sarson, T. (1979). *Structured Systems Analysis: Tools and Techniques*. Prentice Hall.
- Valacich, J. S., & George, J. F. *Modern Systems Analysis and Design*. Pearson (source of the Hoosier Burger examples).
- Kendall, K. E., & Kendall, J. E. *Systems Analysis and Design*. Pearson.
- Lecture slides: "Data Flow Diagram – Analisis Desain Sistem Informasi".