

1 Lecture 5: Data Flow Diagrams (DFD)

Course: Software Engineering and Systems Analysis

Topic: Data Flow Diagrams

Recommended duration: 90 minutes

1.1 Learning objectives

By the end of this lecture, students should be able to define a data flow diagram, distinguish external entities, processes, data stores, and data flows, create context and leveled DFDs, apply balancing rules, and identify common DFD errors.

1.2 1. What a DFD shows

A **Data Flow Diagram (DFD)** represents how data enters, moves through, is transformed by, and leaves a system. It focuses on logical data movement rather than program control flow or physical network topology.

DFDs are useful in requirements analysis because they make information boundaries and transformations visible. They help analysts ask what data is required, which process transforms it, where data is stored, and which external party receives the result.

1.3 2. Four basic elements

An **external entity** is a person, organization, or external system that sends data to or receives data from the system. A **process** transforms inputs into outputs. A **data store** holds data for later use. A **data flow** is a named movement of data between elements.

Use meaningful names. A process should normally contain a verb phrase, such as “Validate Payment” or “Create Invoice.” A data store should contain a noun phrase, such as “Customer Records.” A data flow should name the data, such as “Payment Details” or “Invoice.”

1.4 3. Context diagram

A context diagram is the highest-level DFD. It represents the entire system as one process and shows only its major external entities and data flows. It is useful for agreeing on scope.

Example for an online library system:

```
[Member] -- Search Request --> (Library System)
[Member] <-- Search Results -- (Library System)
[Member] -- Borrow Request --> (Library System)
(Library System) -- Loan Confirmation --> [Member]
(Library System) -- Payment Request --> [Payment Service]
[Payment Service] -- Payment Result --> (Library System)
```

The context diagram does not show internal data stores or internal subprocesses. Adding them would mix levels of detail.

1.5 4. Level-0 and lower-level diagrams

A level-0 DFD decomposes the single context process into major processes. For the library system, the processes may be 1.0 Manage Catalog, 2.0 Manage Loans, and 3.0 Manage Fines. Data stores may include D1 Catalog, D2 Member Records, and D3 Loan Records.

A lower-level diagram decomposes one process, such as 2.0 Manage Loans, into 2.1 Check Member, 2.2 Check Availability, 2.3 Create Loan, and 2.4 Send Confirmation. The decomposition should preserve the inputs and outputs of the parent process.

1.6 5. Balancing

Balancing means that the external inputs and outputs of a parent process match the external inputs and outputs of its child diagram. Internal flows may be added during decomposition, but parent-level flows must not disappear or change meaning without an explicit modeling decision.

Balancing helps prevent scope errors. If a lower-level model introduces “Credit Score” as an input that is not present at the parent level, the analyst must decide whether the context or parent model is incomplete or whether the new data is truly internal.

1.7 6. DFD rules

Data should move through a process. A data flow should not normally connect two external entities directly because the system would not be modeling the interaction. A data store should not exchange data directly with another data store without a process that transforms or manages the data.

Every process should have at least one input and one output in the logical model, although a process may legitimately produce an error output. Avoid black holes, in which inputs disappear, and miracles, in which outputs appear without sufficient inputs.

1.8 7. Logical versus physical DFDs

A **logical DFD** describes what the system does without committing to technology or organizational implementation. A **physical DFD** describes how the work is implemented, including people, applications, devices, files, or specific databases.

For example, “Store Loan Record” is logical. “Write to PostgreSQL Loan table through Loan API” is physical. Analysts often begin with logical DFDs and add physical details when the design or deployment decision requires them.

1.9 8. Worked example: order processing

A context diagram shows a customer submitting order details and receiving an order confirmation. A level-0 diagram may contain 1.0 Validate Order, 2.0 Calculate Total, 3.0 Authorize Payment, and 4.0 Fulfill Order. Data stores may include Product Catalog, Customer Records, and Orders.

The process “Calculate Total” reads product prices and discounts, then outputs an order total. The process “Authorize Payment” sends payment details to an external payment service and receives an authorization result. The DFD makes the data dependencies explicit without describing the exact programming language or user-interface layout.

1.10 9. Common mistakes

Do not confuse a DFD with a flowchart. A flowchart commonly emphasizes control decisions and execution order. A DFD emphasizes data movement and transformation. Do not use arrows without labels, because an unlabeled arrow does not tell the reader what is moving.

Do not place a data store on a context diagram. Do not decompose processes unevenly so that one process has ten detailed children while another remains a vague box. Finally, keep the diagram readable. If the model becomes crowded, split it into levels and provide a short explanation.

1.11 10. Practice activity

Create a context diagram and level-0 DFD for a university registration system. External entities should include Student, Instructor, and Payment Service. Processes should cover course search, registration, payment, and schedule publication. Identify at least three data stores and verify that the diagrams are balanced.

1.12 Summary

A DFD models logical movement and transformation of data. Its essential elements are external entities, processes, data stores, and named data flows. Context diagrams establish scope, lower-level diagrams explain internal behavior, and balancing preserves consistency across decomposition levels.

1.13 References