Understanding Data Flow Diagrams

Donald S. Le Vie, Jr.

Data flow diagrams (DFDs) reveal relationships among and between the various components in a program or system. DFDs are an important technique for modeling a system's high-level detail by showing how input data is transformed to output results through a sequence of functional transformations. DFDs consist of four major components: entities, processes, data stores, and data flows. The symbols used to depict how these components interact in a system are simple and easy to understand; however, there are several DFD models to work from, each having its own symbology. DFD syntax does remain constant by using simple verb and noun constructs. Such a syntactical relationship of DFDs makes them ideal for object-oriented analysis and parsing functional specifications into precise DFDs for the systems analyst.

DEFINING DATA FLOW DIAGRAMS (DFDs)

When it comes to conveying how information data flows through systems (and how that data is transformed in the process), data flow diagrams (DFDs) are the method of choice over technical descriptions for three principal reasons.

1. DFDs are easier to understand by technical and nontechnical audiences
2. DFDs can provide a high level system overview, complete with boundaries and connections to other systems
3. DFDs can provide a detailed representation of system components[1]

DFDs help system designers and others during initial analysis stages visualize a current system or one that may be necessary to meet new requirements. Systems analysts prefer working with DFDs, particularly when they require a clear understanding of the boundary between existing systems and postulated systems. DFDs represent the following:

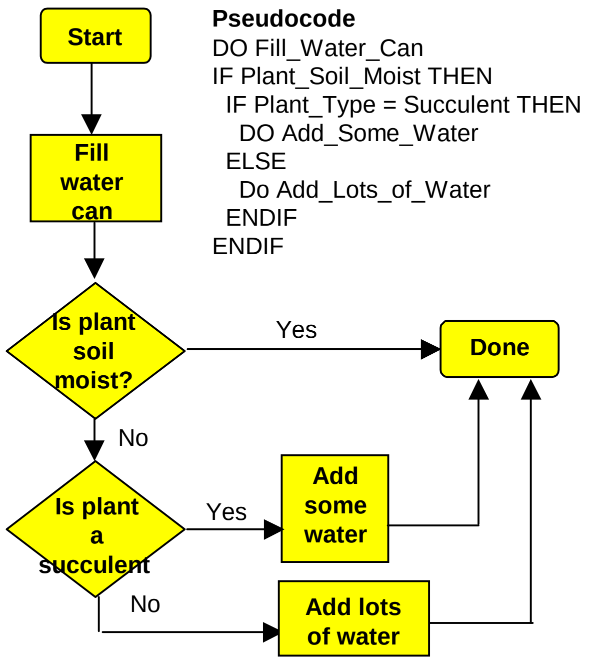
1. External devices sending and receiving data
2. Processes that change that data
3. Data flows themselves
4. Data storage locations

The hierarchical DFD typically consists of a top-level diagram (Level 0) underlain by cascading lower level diagrams (Level 1, Level 2…) that represent different parts of the system.

Before There Were DFDs…

Flowcharts and Pseudocode

Years ago, programmers used a combination of flowcharts and pseudocode (a combination of English and the programming language being written) to design programs. Pseudocode can actually be simpler to read than corresponding flowcharts, as Figure 1 illustrates.



Pseudocode

```
DO Fill_Water_Can
IF Plant_Soil_Moist THEN
  IF Plant_Type = Succulent THEN
    DO Add_Some_Water
  ELSE
    Do Add_Lots_of_Water
  ENDIF
ENDIF
```

Figure 1. A Flowchart With Corresponding Pseudocode for Watering House Plants.

Entity-Relationship Diagrams (ERDs)

Entity-Relationship Diagrams (ERDs) are another way of showing information flow for a process. An ERD shows what data is being used in the process or program, and how the files are related. The E-R (entity-relationship) data model views the real world as a set of basic objects (entities) and relationships among these objects. It is intended primarily for the database design process by allowing for the specification of an enterprise scheme. This enterprise scheme represents the overall logical structure of the database. ERDs do not show any program functions, nor data flow. An ERD is shown in Figure 2.

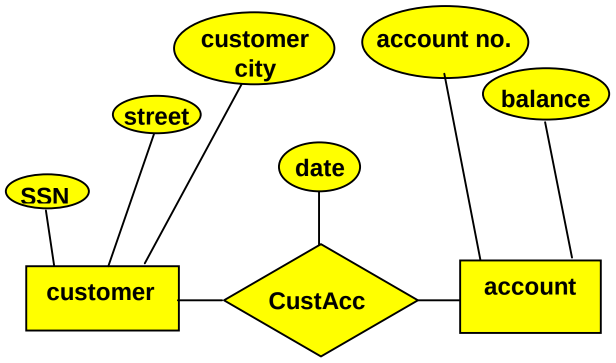


Figure 2. An Example of an Entity-Relationship Diagram.

Data Flow Diagrams

Data flow diagrams have replaced flowcharts and pseudocode as the tool of choice for showing program design. A DFD illustrates those functions that must be performed in a program as well as the data that the functions will need. A DFD is illustrated in Figure 3.

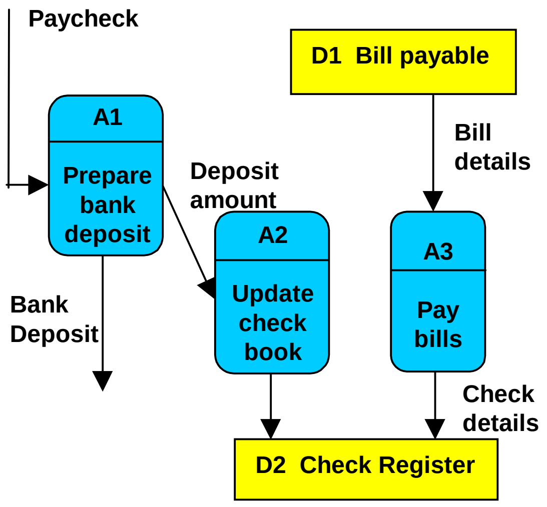


Figure 3. An Example of a Data Flow Diagram.

Defining DFD Components

DFDs consist of four basic components that illustrate how data flows in a system: entity, process, data store, and data flow.

Entity

An *entity* is the source or destination of data. The source in a DFD represents these entities that are outside the context of the system. Entities either provide data to the system (referred to as a source) or receive data from it (referred to as a sink). Entities are often represented as rectangles (a diagonal line across the right-hand corner means that this entity is represented somewhere else in the DFD). Entities are also referred to as *agents*, *terminators*, or *source/sink*.

Process

The *process* is the manipulation or work that transforms data, performing computations, making decisions (logic flow), or directing data flows based on business rules. In other words, a process receives input and generates some output. Process names (simple verbs and dataflow names, such as "Submit Payment" or "Get Invoice") usually describe the transformation, which can be performed by people or machines. Processes can be drawn as circles or a segmented rectangle on a DFD, and include a process name and process number.

Data Store

A *data store* is where a process stores data between processes for later retrieval by that same process or another one. Files and tables are considered data stores. Data store names (plural) are simple but meaningful, such as "customers," "orders," and "products." Data stores are usually drawn as a rectangle with the right-hand side missing and labeled by the name of the data storage area it represents, though different notations do exist.

Data Flow

Data flow is the movement of data between the entity, the process, and the data store. Data flow portrays the interface between the components of the DFD. The flow of data in a DFD is named to reflect the nature of the data used (these names should also be unique within a specific DFD). Data flow is represented by an arrow, where the arrow is annotated with the data name.

These DFD components are illustrated in Figure 4.

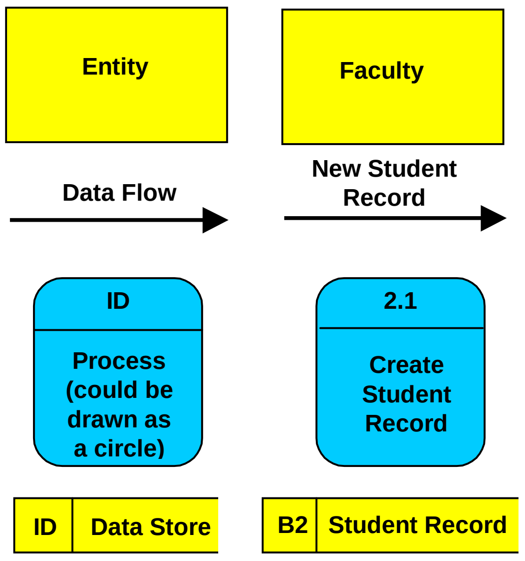


Figure 4. The Four Major DFD Components.

Process for Developing DFDs

Data flow diagrams can be expressed as a series of levels. We begin by making a list of business activities to determine the DFD elements (external entities, data flows, processes, and data stores). Next, a *context diagram* is constructed that shows only a single process (representing the entire system), and associated external entities. The *Diagram-0*, or Level 0 diagram, is next, which reveals general processes and data stores (see Figures 5 and 6). Following the drawing of Level 0 diagrams, *child diagrams* will be drawn (Level 1 diagrams) for each process illustrated by Level 0 diagrams.

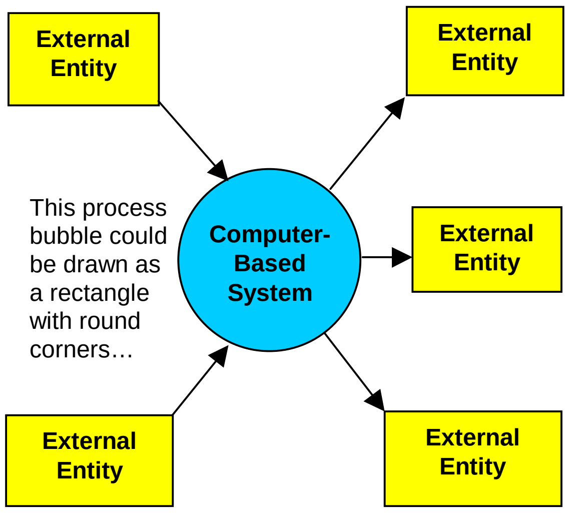


Figure 5. General Form of a Level 0 DFD.

Figure 6 offers a more specific Level 0 DFD.

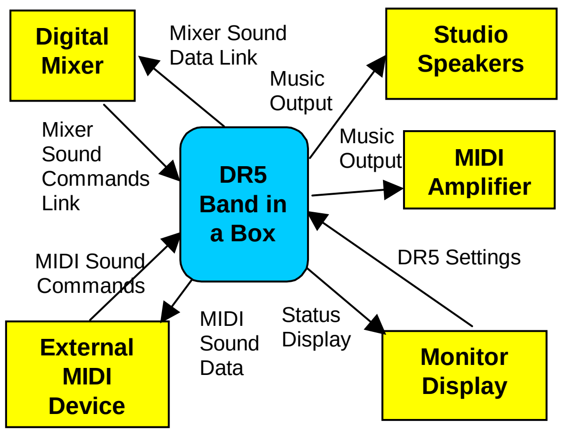


Figure 6. Specific Level 0 DFD

GUIDELINES FOR PRODUCING DFDS

Why They Aren't Called "Rules"

The most important thing to remember is that there are no hard and fast rules when it comes to producing DFDs, but there are when it comes to valid data flows. For the most accurate DFDs, you need to become intimate with the details of the use case study and functional specification. This isn't a cakewalk necessarily, because not all of the information you need may be present. Keep in mind that if your DFD looks like a Picasso, it could be an accurate representation of your current physical system. DFDs don't have to be art; they just have to accurately represent the actual physical system for data flow.

Preliminary Investigation of Text Information

The first step is to determine the data items, which are usually located in documents (but not always). Once you identify the data items, you'll need to determine where they come from (source) and where they go (destination). Construct a table to organize your information, as shown in Table 1.

Data Item	Source	Destination
Needs Analysis	Account Executive	Project Manager
ROI Study	Pre-Sales Support	Proposal Manager

Table 1. Data Item Table

Determining System Boundaries

Once you have the data items, sources, and destinations in a table, determine which entities (sources and destinations) belong internal to the system and which ones are external to the system. Sometimes it helps to have some knowledge about what may be happening at deeper levels (Level 1+) to work backwards to help you develop a Level 0 DFD. Such a method depends on the system being modeled or personal preference, but knowing that DFD development is an iterative process prepares you for the many DFD drafts you may have to generate.

Developing the Level 0 DFD

At this point, you should have a good idea of the system boundary. All components within the system boundary are included within a single system/process box in the DFD. External entities lie outside the system boundary; internal entities will become locations for processes. The data flow arrows to and from the external entities will indicate the system's relationship with its environment. Remember that information always flows to or from a process, an external entity, or a data store. You can use a dashed line to show data flows between external entities that are strictly external to the system at hand if it will help make the DFD easier to understand.

Child (Level 1+) Diagrams

DFDs can be expressed as a series of levels. The outermost level (Level 0) is concerned with how the system interacts with the outside world. Subsequent levels examine the system in more detail, and use the same symbols and syntax as with Level 0. See Figure 7.

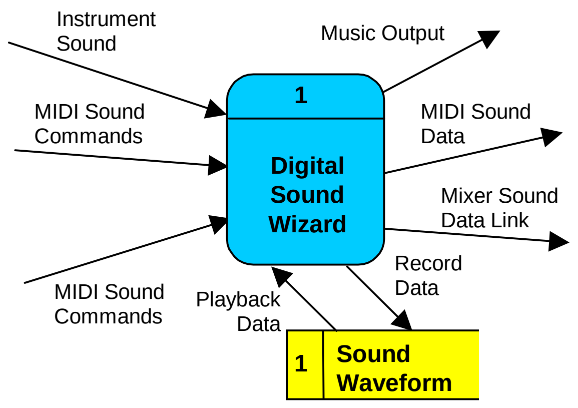


Figure 7. Example of a Level 1 DFD Showing the Data Flow and Data Store Associated With a SubProcess "Digital Sound Wizard."

When producing a first-level DFD, the relationship of the system with its environment must be preserved. In other words, the data flow in and out of the system in the Level 1 DFD must be exactly the same as those data flows in Level 0. If you discover new data flows crossing the system boundary when drawing the Level 1 DFD, then the Level 0 DFD must be amended to reflect the changes in the Level 1 DFD.

Developing the Level 1 DFD

It is important that the system relationship with its environment be preserved no matter how many levels deep you model. In other words, you can't have new data flows crossing the system boundary in Level 1. The next section deals with such non-valid data flows.

The Level 1 DFD provides a high-level view of the system that identifies the major processes and data stores. Identify or list each incoming and outgoing data flow with a corresponding process that receives or generates data. Make sure you refer to your data item table for any missing internal data flows and to identify data stores. If your table contains documents with the same source and destination, they might be data stores. Some processes share data stores while some data stores are used by one process. It may be possible to move the *single process* • *data store* inside the process itself. Identify those processes that only address internal outputs and outputs, and use one process for each source or destination from the DFD.

Revising the Level 1 DFD

Once you've finished your first attempt at a Level 1 DFD, review it for consistency and refine it for balance by asking yourself these questions:

1. Do the Level 1 processes correspond with the major functions that a user expects from the system?
2. Is the level of detail balanced across the DFD?
3. Can some processes be merged?
4. Can I remove data stores not shared by more than one process?
5. Have I avoided crossed data flow lines by making use of duplicated components (external entities and data stores)?

Some Guidelines About Valid and Non-Valid Data Flows

Before embarking on developing your own data flow diagram, there are some general guidelines you should be aware of.

Data stores are storage areas and are static or passive; therefore, having data flow directly from one data store to another doesn't make sense because neither could initiate the communication.

Data stores maintain data in an internal format, while entities represent people or systems external to them. Because data from entities may not be syntactically correct or consistent, it is not a good idea to have a data flow directly between a data store and an entity, regardless of direction.

Data flow between entities would be difficult because it would be impossible for the system to know about any communication between them. The only type of communication that can be modeled is that which the system is expected to know or react to.

Processes on DFDs have no memory, so it would not make sense to show data flows between two asynchronous processes (between two processes that may or may not be active simultaneously) because they may respond to different external events.

Therefore, data flow should only occur in the following scenarios:

- Between a process and an entity (in either direction)
- Between a process and a data store (in either direction)
- Between two processes that can only run simultaneously

Figure 8 illustrates these valid data-flow scenarios.

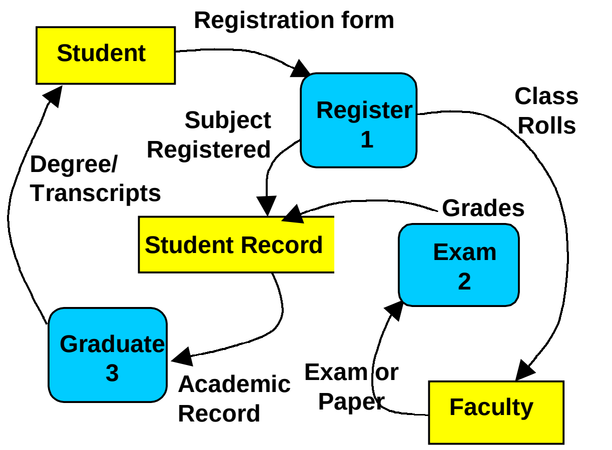


Figure 8. A Valid DFD Example Illustrating Data Flows, Data Store, Processes, and Entities.

In Figure 8, *Student* and *Faculty* are the source and destination of information (the entities), respectively. *Register 1*, *Exam 2*, and *Graduate 3* are the processes in the program. *Student Record* is the data store. *Register 1* performs some task on Registration Form from *Student*, and the Subject Registered moves to the data store. The Class Rolls information flows on to *Faculty*. *Graduate 3* obtains Academic Record information from *Student Record*, and Degree/Transcript information is moved to *Student. Exam 2* obtains exam/paper information from *Faculty*, and moves the Grades to the *Student Record* for storage.

Here are a few other guidelines on developing DFDs:

- Data that travel together should be in the same data flow
- Data should be sent only to the processes that need the data
- A data store within a DFD usually needs to have an input data flow
- Watch for Black Holes: a process with only input data flows
- Watch for Miracles: a process with only output flows
- Watch for Gray Holes: insufficient inputs to produce the needed output
- A process with a single input or output may or may not be partitioned enough
- Never label a process with an IF-THEN statement
- Never show time dependency directly on a DFD (a process begins to perform tasks as soon as it receives the necessary input data flows)

ANALYSIS AND DESIGN METHODS

Analysis and design methods are tools used during the project life cycle for bringing order and structure to the analysis and design process. They specify the steps, notation, rules, and guidelines for conducting object-oriented analysis and design (OOAD). We will briefly focus on object-oriented analysis (OOA).

Steps in Object-Oriented Analysis

The first step in OOA is often the identification of important classes, which is done by identifying the nouns in the requirement specification. Many times, classes are defined from tangible things, roles, or incidents.

Analysis also includes defining object attributes and object and class or other relationships. Analysis also attempts to identify the behavior of objects and classes as well as data flow between objects.

Object Modeling Technique (OMT)

The Object Modeling Technique (OMT) is an approach to object-oriented analysis and design that is also referred to as the Rumbaugh Method, named after the principal author of the technique. The OMT methodology is programming language-independent, instead relying on a consistent, single notation for both analysis and design.

OMT Modeling

The output of OMT is a three-dimensional view (from three different models) of the system that stays the same during the transition from analysis to design.

The *Object Model* describes the static system components and is modeled using *object diagrams*. The *Dynamic Model* describes the dynamic system components that change over time and are modeled using *state diagrams*. The *Functional Model* describes operations performed on data in a system and uses *data flow diagrams*.

Advantages and Disadvantages of DFDs

Strengths

As we have seen, the DFD method is an element of object-oriented analysis and is widely used. Use of DFDs promotes quick and relatively easy project code development. DFDs are easy to learn with their few-and-simple-to-understand symbols (once you decide on a particular DFD model). The syntax used for designing

DFDs is simple, employing English nouns or noun-adjective-verb constructs.

Disadvantages

DFDs for large systems can become cumbersome, difficult to translate and read, and be time consuming in their construction. Data flow can become confusing to programmers, but DFDs are useless without the prerequisite detail: a Catch-22 situation. Different DFD models employ different symbols (circles and rectangles, for example, for entities).

Using DFDs in the Appropriate Application Domain

DFDs are useful tools for OO architectures and are best utilized with OO languages and compiler programming. DFD semantics can be accurately and precisely represented within OO technology, as Figure 9 illustrates. The methods of the Customer Object and Vendor Object are of the same syntactical construct as DFDs.

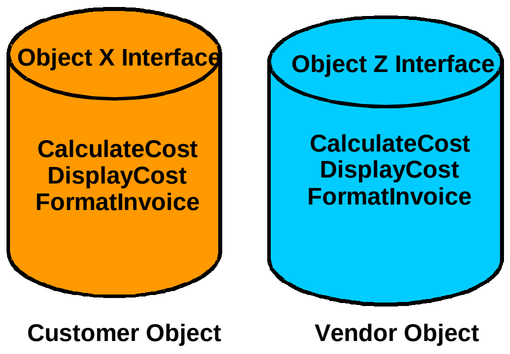


Figure 9. Example of DFD Semantics Used in Object Technology[2]

Because of the semantic relationship DFDs have with OO programming, DFDs are inappropriate for non-OO architectures. The first and most important step in translating a functional spec into a DFD is to parse it into its component verbs and nouns so that a DFD can be developed that precisely coincides with the functional specification.

CONCLUSION

Data flow diagramming is a highly effective technique for showing the flow of information through a system. DFDs are used in the preliminary stages of systems analysis to help understand the current system and to represent a required system. The DFDs themselves represent external entities sending and receiving information (entities), the processes that change information (processes), the information flows themselves (data flows), and where information is stored (data stores). The hierarchical DFDs consist of a single top layer (Level 0 or the *context* diagram) that can be decomposed into many lower level diagrams (Level 1, Level 2…Level N), each representing different areas of the system.

DFDs are extremely useful in systems analysis as they help structure the steps in object-oriented design and analysis. Because DFDs and object technology share the same syntax constructs, DFDs are appropriate for the OO domain only.

DFDs are a form of information development, and as such provide key insight into how information is transformed as it passes through a system. Having the skills to develop DFDs from functional specs and being able to interpret them is a value-add skill set that is well within the domain of technical communications.

REFERENCES

(1) Perry, Greg. *Sams Teach Yourself Beginning Programming in 24 Hours*, Sams Publishing, 1998. 492 pages.

(2) Le Vie, Jr., Donald. “An eCommerce Primer for Technical Communicators,” STC *Proceedings* of the 47th Annual Conference, 2000.

Donald (Donn) S. Le Vie, Jr.
Information Development Director
Integrated Concepts, Inc.
8834 Capital of Texas Highway North, Suite 280
Austin, Texas 78759
(512) 231-9999, ext. 231
dlevie@integratedconcepts.com

Donn directs the information development efforts at Integrated Concepts, Inc., an eCommerce software development company in Austin, Texas. He has more than 20 years’ experience in private industry, academia, and government. When he’s not writing articles and books, Donn records and tours with his band, *SnakeByte*.