Unit – 4 Data Flow Diagram and System Prototype Design

- Meaning and Significance of Data flow diagram (DFDs)
- Symbols used in DFDs
- Rules for Constructing DFDs
- Introduction and comparison between Physical and Logical DFDs
- Introduction to System Prototype
- Reasons for System Prototyping
- Prototype Model: Diagram and Steps of Prototype Development Model
- Merits and Demerits of Prototypes

Data Flow Diagram (DFD)

What is a data flow diagram?

A data flow diagram shows the way information flows through a process or system. It includes data inputs and outputs, data stores, and the various subprocesses the data moves through. DFDs are built using standardized symbols and notation to describe various entities and their relationships.

Data flow diagrams visually represent systems and processes that would be hard to describe in a chunk of text. You can use these diagrams to map out an existing system and make it better or to plan out a new system for implementation. Visualizing each element makes it easy to identify inefficiencies and produce the best possible system.

What is a Purpose (Use) of Data Flow Diagrams?

Data flow diagrams are used by information technology professionals and systems analysts to document and show users how data moves between different processes in a system. Analysts generally start with an overall picture and then move on to the finer details of each process.

Data flow diagrams provide a graphical representation of how information moves between processes in a system. Data flow diagrams follow a hierarchy; that is, a diagram may consist of several layers, each unique to a specific process or data function.

Elements of DFD

All data flow diagrams include four main elements: entity, process, data store and data flow.

External Entity – Also known as actors, sources or sinks, and terminators, external entities produce and consume data that flows between the entity and the system being diagrammed. These data flows are the inputs and outputs of the DFD. Since they are external to the system being analyzed, these entities are typically placed at the boundaries of the diagram. They can represent another system or indicate a subsystem.

Process – An activity that changes or transforms data flows. Since they transform incoming data to outgoing data, all processes must have inputs and outputs on a DFD. This symbol is given a simple name based on its function, such as “Ship Order,” rather than being labeled “process” on a diagram. In Gane-Sarson notation, a rectangular box is used and may be labeled with a reference number, location of where in the system the process occurs and a short title that describes its function. Processes are typically oriented from top to bottom and left to right on a data flow diagram.

Data Store – A data store does not generate any operations but simply holds data for later access. Data stores could consist of files held long term or a batch of documents stored briefly while they wait to be processed. Input flows to a data store include information or operations that change the stored data. Output flows would be data retrieved from the store.

Data Flow – Movement of data between external entities, processes and data stores is represented with an arrow symbol, which indicates the direction of flow. This data could be electronic, written or verbal. Input and output data flows are labeled based on the type of data or its associated process or data store, and this name is written alongside the arrow.

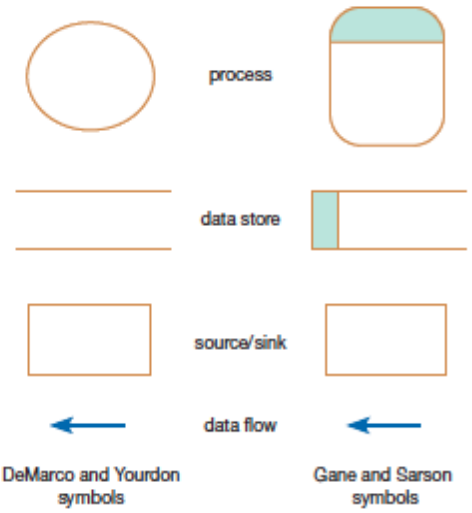


FIGURE 7-2
Comparison of DeMarco and Yourdon and Gane and Sarson DFD symbol sets

Types of DFD (Levels of DFD)

Although all data-flow diagrams are composed of the same types of symbols, and the validation rules are the same for all DFDs, there are three main types of data-flow diagram:

- **Context diagrams** — context diagram DFDs are diagrams that present an overview of the system and its interaction with the rest of the "world".
- **Level 1 data-flow diagrams** — Level 1 DFDs present a more detailed view of the system than context diagrams, by showing the main sub-processes and stores of data that make up the system as a whole.
- **Level 2 (and lower) data-flow diagrams** — a major advantage of the data-flow modelling technique is that, through a technique called "levelling", the detailed complexity of real-world systems can be managed and modeled in a hierarchy of abstractions. Certain elements of any data-flow diagram can be decomposed ("exploded") into a more detailed model a level lower in the hierarchy.

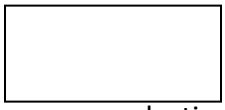
Symbols Used in Data Flow Diagram

1. Data process

A data process transforms data values. Here flow of data is transformed. E.g. Verify credits, updates inventory file.

You can make a distinction between the following types of processes:

2. External Entity



A source or destination of data which is external to the system. E.g supplier, customer etc.

3. Data store

A data store stores data passively for later access. A data store responds to requests to store and access data. It does not generate any operations. A data store allows values to be accessed in an order different from the order in which they were generated.

Input flows indicate information or operations that modify the stored data such as adding or deleting elements or changing values. Output flows indicate information retrieved from the store; this information can be an entire value or a component of a value.

4. Data flow

A data flow moves data between processes or between processes and data stores. As such, it represents a data value at some point within a computation and an intermediate value within a computation if the flow is internal to the diagram. This value is not changed.

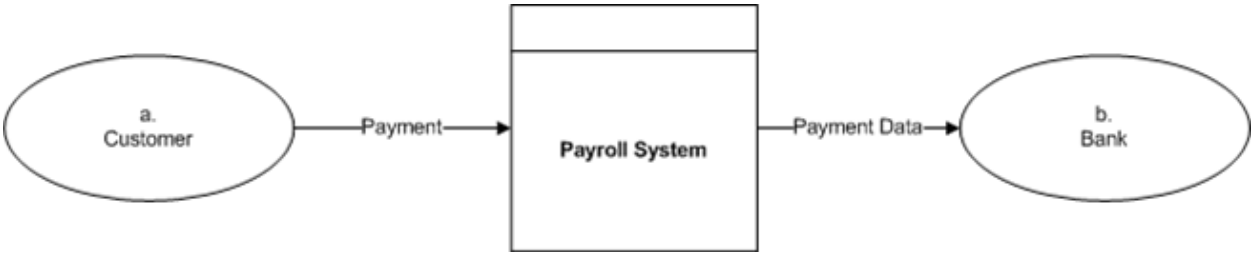
The names of input and output flows can indicate their roles in the computation or the type of the value they move. Data names are preferably nouns. The name of a typical piece of data, the data aspect, is written alongside the arrow.

Rules for Constructing DFD

When creating data flow diagrams (DFD's), there are certain rules which must be followed. These rules allow for the DFD to be make sense and also to be easily understood. In this blog I will go through the rules which must be followed and show practical examples of these rules.

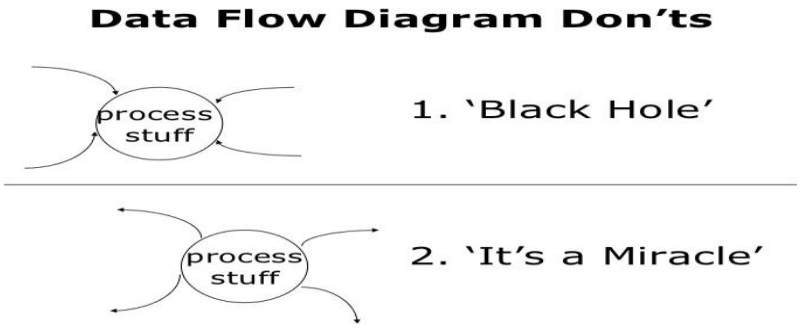
1. All data flows must flow to or from a process

All flows of data must be either coming from or going to a process. External entities cannot flow directly to each other. A data flow cannot link a data store to an external entity. Data cannot move between data stores without first being processed.



2. A Process must have at least one input flow and one output flow.

When a process has input flow but no output flow, it is called a "black hole". When a process has output flows but no input flows, it is called a "miracle". A process must have at least one input flow and one outflow flow.

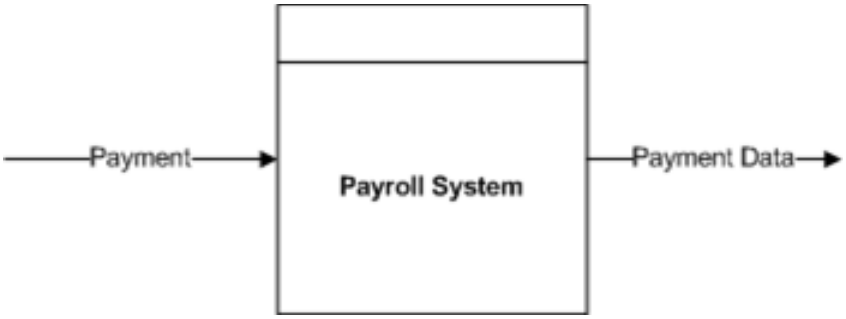


3. The inputs to a process must be sufficient to produce output flows.

A "grey hole" is when the outputs of a process are greater than the sum of its inputs. For example, if a customer's name and address is an input, their bank details cannot be an output, as the process doesn't have enough information to produce it.

4. Processes must transform data.

When naming data flows, adjectives should be used which show how processing has changed the data flow.



5. Data Flows cannot cross each other.

The flows of data cannot cross each other. To overcome this problem, data stores and entities can be duplicated. However, processes cannot be duplicated. Data flows must be unidirectional.

6. Entities must be labelled in lower case.

Physical and Logical DFDs

The DFDs that show “What is going on” instead of “How it is going on” is a logical DFD. DFDs that show how things happen and which are the actual physical components involved are known as physical DFDs. Logical DFDs help to get a clear idea of what the system has to achieve without getting into details like who is going to do it? How one is going to do it? Etc. But physical models are easier to visualize. Hence analyst begins with physical DFD before converting it to logical DFD.

Physical DFD	Logical DFD
Data flow names include the implementation facts as names, numbers, media, timing etc.	Data flow names describe the data they contain. They do not refer to the form or document on which they reside.
Process names include the name of the processor i.e. person, department, computer system etc.	Process names describe the work done without referring to e.g. Account Receivable, Order processing etc.
Data Stores identify their computer and manual implementation.	Physical location of data stores is irrelevant. Many times, the same data store may be shared by subsystems and processes.
This is more realistic and implementation oriented. The PDFD are more detailed in nature.	As the name suggests, this is more logical in format. This is more abstract than PDFD and less dependent on implementation steps.

Merits of DFD

- It aids in describing the boundaries of the system.
- It is beneficial for communicating existing system knowledge to the users.
- A straightforward graphical technique which is easy to recognize.
- DFDs can provide a detailed representation of system components.
- It is used as the part of system documentation file.
- DFDs are easier to understand by technical and nontechnical audiences
- It supports the logic behind the data flow within the system.

Demerits of DFD

- It makes the programmers little confusing concerning the system.
- The biggest drawback of the DFD is that it simply takes a long time to create, so long that the analyst may not receive support from management to complete it.
- Physical considerations are left out.

Prototype

- The term prototype refers to a working model of an information system application.
- The prototype does not contain all the features or perform all the necessary functions of the final system.
- Rather, it includes sufficient elements to enable individuals to use the proposed system to determine what they like and don't like and to identify features to be added or changed.

Characteristics

1. The prototype is a live, working application
2. The purpose of prototyping is to test out assumptions made by analysts and users about required system features
3. Prototypes are created quickly
4. Prototypes evolve through an iterative process
5. Prototypes are relatively inexpensive to build

Reasons (Need) for System Prototyping

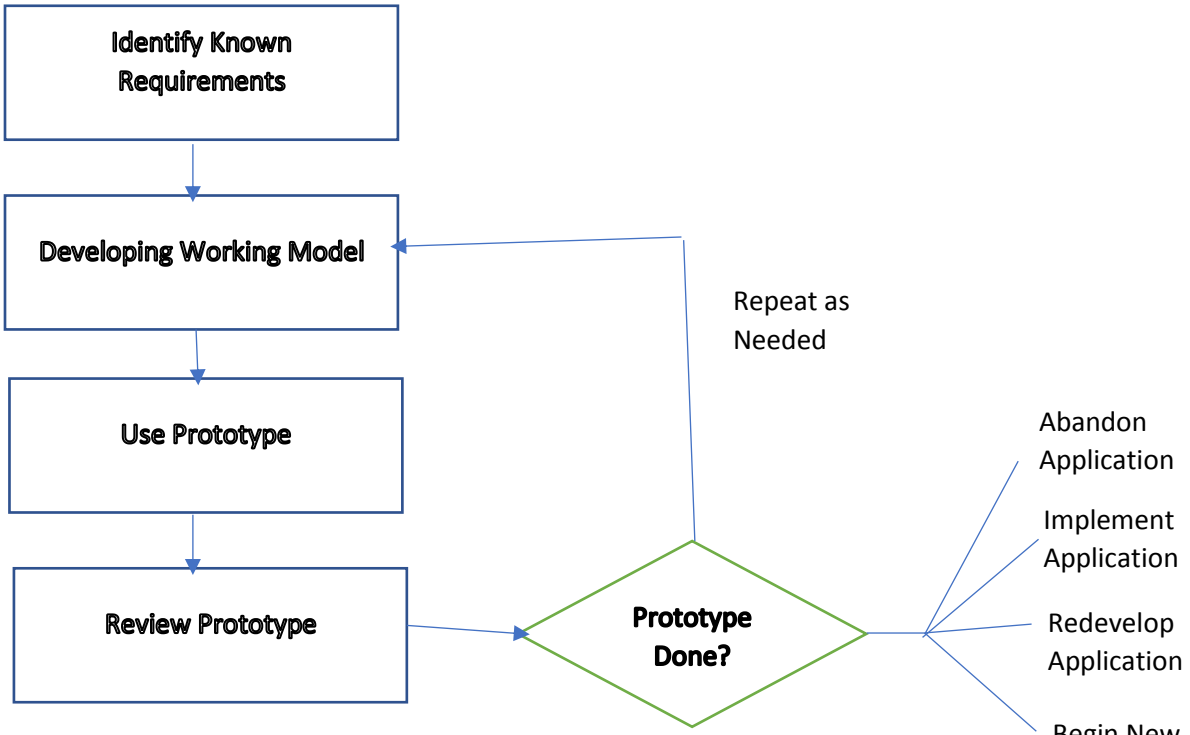
Application prototyping is most effective in the development of information systems when certain conditions are met. Any of the following five application conditions suggests the need for prototyping:

1. **Requirements Not Known**
   The nature of the applications is such that there is little information available about the features the system must have to meet its users' requirements.
   Example: A national consumer products firm wishes to develop a voice mail system that can also trigger the preparation of selected printed reports. The firm has not used voice mail in the past.
2. **Requirements Need Evaluation**
   Apparent information requirements of the organization and end users are known, but verification and assessment are needed.
   Example: A large university wishes to reduce the congestion and delay students face each term when registering for courses. A plan has been devised to allow students to enter identification information and course numbers from and push-button, touch-tone telephone. The system will accept local or long-distance telephone calls. The specifications for a system to facilitate this process while minimizing errors and maintaining system integrity have been articulated. Security requirements have also been developed to prevent unauthorized use of the automated registration system.

3. **High Cost**
   The investment of financial resources, human effort, and time required to bring an application into existence is substantial. Other projects also complete for the same resources.
   Example: A city-wide online teller system will be installed in over one hundred branch offices by a large metropolitan bank. The costly system will be designed to provide for the instantaneous capture of information and updating of a central database.
4. **High Risk**
   Inaccurate evaluation of system requirements or incorrect development of an application places an organization, its employees, or its resources in jeopardy.
   Example: A manufacturing floor control system will be designed to move materials in inventory from the company's warehouse through the production process. The organization wishes to keep a minimum inventory on hand and yet not experience delay in the manufacturing process. A delay in any area, caused by error or insufficient materials, could cause a shutdown of the entire production process. Unplanned shutdowns damage material, destroy production schedules, and lead to the risk of losing those customers whose orders are not filled on time.
5. **New Technology**
   A desire to install new technology, whether in the computer field, data communications, or other related areas, will open a new frontier for the organization. Many firms have no experience in using certain technology and neither do other organizations with whom they communicate.
   Example: A company wishes to develop a system permitting voice entry of customer orders: staff personnel will repeat order details aloud at a voice input unit, rather than keying the data for processing.

Steps in Prototype Method

The development of a prototype application proceeds in an orderly fashion, regardless of the particular tools used.



Steps in prototype development method

1. **Identify Known Requirements**
   - Before a prototype is created, both user and systems analyst work together to identify the known requirements that must be met. To do so, they determine the purpose the system will serve and the scope of its capabilities.

2. **Develop Working Model**
   - Prototyping uses an iterative development process.
   - Prior to the first iteration, systems analysts describe the prototype method to the users, explaining what activities will occur and in what sequence, and discussing the responsibilities of each participant.
   - It is helpful to begin the prototyping process by developing a general plan so that individuals know what to expect from each other and from the development process.
   - A timetable for the startup and proposed completion of the initial iteration should be constructed at this time.
   - To begin the first iteration, user and analyst jointly identify the data that are needed in the system and specify the output the application must produce. This means describing (1) the individual reports and documents the system should provide and (2) the layout of each.
   - The analyst also estimates the prototyping cost.

- In developing the prototype, these components are prepared:
  - Command language dialogue or conversation between user and system
  - Input screens and formats
  - Essential processing modules
  - System output

3. **User Prototype**
   - It is the user's responsibility to work with the prototype and evaluate its features and operation. Experience with the system in the actual application setting should provide the familiarity needed to determine what changes or enhancements are necessary or which inadequate or undesirable features to eliminate.
4. **Review Prototype**
   - During the evaluation, systems analysts will want to capture information on what users like and dislike, noticing why they react as they do. The information will influence the features the next version of the application should have.
   - It also provides insight into characteristics of the users and the business setting for the application- details that will influence not only the application, but the way it is later implemented.
   - Changes to the prototype are planned with users before they are made.
5. **Repeat as Needed**
   - The process described may be repeated several times to evolve the application. Four to Six iterations are typical.
   - This process ends when both users and analyst agree that the system has evolved to include the necessary features or when it is evident that there is no benefit to additional iteration.
6. **Use of Prototypes**
   - When the prototyping process is complete, a decision is made about how to proceed. There are four ways to proceed after the information gained from developing and using the prototype has been evaluated: discard the prototype and abandon the application project, implement the prototype, redevelop the application, or begin another prototype.
     i. **Abandon Application**
        - In some instances, the decision will be to discard the prototype and to abandon development of the application. Such a conclusion does not mean that the prototype process was a mistake or a waste of resources.
        - Rather, the information and experience gained by developing and using the prototype has led to a development decision. Perhaps user and analyst have learned that the system will be unnecessary i.e. an alternative solution was discovered during the prototyping process.

- Or, the experience suggested theta the approach was inappropriate.
- In some cases, it may turn out that the event that triggered the development effort is a one-time occurrence. The prototype met the immediate need and the event is not expected to occur again.
- Any of the above instances may suggest that further pursuit of the prototype or the full-function application should stop. Decision like this saves time and resources, allowing analysts to redirect their efforts to other application needs.

ii. Implement Prototype

- Sometimes the prototype becomes the actual system needed. In this case, it is implemented as is; no further development occurs. This decision is most likely to be made under one or more of the following circumstances:
    a. The prototype evolved led to an application consisting of the required features, capabilities, and performance characteristics.
    b. The application will be used infrequently and speed or efficiency or operation is not essential.
    c. The application does not affect or interact with other applications or data in the organization and meets the needs of its immediate users.
    d. The application environment is in a state of flux; it is difficult to determine more long term or stable operation needs. Thus, other development activities cannot be justified. The prototype will do for the time being.
- When the operating environment is uncertain (meaning it is difficult to identify concrete requirements), the prototype may be implemented indefinitely.
- Approximately half of the all prototypes are implemented as the working application.

iii. Redevelop Application

- When the application is redeveloped, emphasis is also placed on making the best possible use of system resources. Processing speed and response time take on greater importance, as does efficient use of storage.
- Redeveloping an application can occur as part of the classical systems development life cycle method. The two most common ways of incorporating application prototyping are these:
    - The prototype is used as an alternative to the requirements determination activity; the prototype features serve as

requirements to be met through the development activities that follow.

- The prototype is used as a substitute for the design and implementation of the working application, a skeleton from which the remainder of the system is constructed.

iv. Begin New Prototype

- The fourth alternative is to begin a new prototyping project. The information gained by developing and using the prototype will sometimes suggest the use of an entirely different approach to meet the organizer's needs. It may reveal that the features of the application must be dramatically different if the existing prototype is inappropriate to demonstrate and evaluate those features.
- Consequently, rather than jumping into a full-scale development effort with the newly acquired information, management may support the creation of another prototype model.

Merits of Prototypes

1. **Reduced time and costs**

   Prototyping improves the quality of the specifications and requirements provided to customers. With prototyping, customers can anticipate higher costs, needed changes and potential project hurdles, and most importantly, potential end result disasters. Strong prototyping can ensure product quality and savings for years to come.

2. **Improved and increased user involvement**

   Most customer want to feel like they are involved with the intricate details of their project. Prototyping requires user involvement and enables them to see and interact with a working model of their project. With prototypes, customers can give their immediate feedback, request project changes and alter model specifications. Prototyping most importantly helps eliminate misunderstandings and miscommunications during the development process.

3. **Reduced time and costs**

   Nothing makes customers happier than projects that come in under budget. Prototyping improves the quality of requirements and specifications provided to customers. Needed changes detected later in development cost exponentially more to implement. With prototyping, you can determine early what the end user wants with faster and less expensive software.

Demerits of Prototypes

1. **Insufficient analysis**

   A focus on a limited prototype can distract developers from properly analyzing the complete project. The potential end result: A potential overlooking of better solutions, incomplete specifications or the conversion of limited prototypes into poorly engineered and developed final projects that are hard to maintain.

2. **User confusion**

   The worst-case scenario of any prototype is customers mistaking it for the finished project. Customers seeing a rough prototype may not understand it merely needs to be finished or polished. Also, customers can wrongly perceive the prototype to accurately model the performance of the final system. Customers may also grow fond of prototype features that are not part of the final system.

3. **Developer misunderstanding of user objectives**

   For every project to be successful, developers and customers must be on the same page and share the same project objectives. If customers require all proposed features of a prototype be included in the final product, this can lead to team and mission conflicts.

4. **Excessive Development Time**

   Remember, prototypes are by nature designed to be developed quickly. If a developer spends too much time developing a complex prototype, the project can run into roadblocks (especially if there are disagreements over prototype details) and run over both time and cost budgets.