Lecture Notes

Structured Systems Analysis

Lecture 3

Structured Analysis & Data Flow Diagrams

Written by Dr. Fred Grossman

Structured Systems Analysis

Structured analysis is a methodology for determining and documenting the requirements for a system. As we shall see, the operative term is structured. The goal of structured analysis is to determine and document the requirements for a system so that the correct system is constructed. Key to this is the ability for the system owner to understand (and validate) that the system requirements are correctly and completely specified, and that the designer interprets the requirements in exactly the same way as the system owner.

The two major documentation tools for a structured analysis are the Data Flow Diagram (DFD) and the Data Dictionary. The data dictionary defines and describes all of the names which appear in the system documentation. We will have more to say about the data dictionary later.

Structured analysis is a pure requirements methodology. It should be completely devoid of design. It specifies WHAT and not HOW. The design methodology that follows the requirements phase of development is not predetermined by the structured analysis. The design may be a structured design or an object design, or any other design methodology. Whatever the design methodology (structure) that may be used, it is necessary that the design faithfully represent the requirements. Structured analysis supports this.

Data Flow Diagrams (DFDs)

Data flow diagrams provide graphical documentation for systems analysis. They are analogous to blue prints in a construction project. They are leveled set of diagrams so that it is easier to understand the system requirements with fewer pieces than textual documentation. The leveling process also helps control complexity.

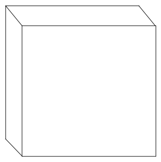
Using DFDs to document a system does not necessarily imply that we are doing a structured analysis. DFDs are a documentation tool and can be used to document various things. DFDs can be used to document design flows or control flows as well as requirements. In this discussion we are concerned with structured analysis and will use the DFD to document the analysis.

DFDs help manage complexity. Complexity almost always means the number of things that we need to manage at once. Through the leveling process that the structured analysis methodology mandates, a diagram will always have a manageable number of components. Miller's law states that humans tend to lose control when they have to manage more than 7 plus or minus 2 things at one time. For example, consider a juggler.

Data Flow Diagram Symbols

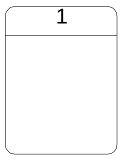
There are four symbols in a DFD - the **External Entity** (Source/Sink), the **Process**, the **Data Store** and the **Data Flow**.

External Entity



An external entity is in the system environment. It is a source or sink of data flows. Data flows from an external entity to the system and from the system to an external entity. The system cannot control when or if the data flows from the external entity since these entities are not in the system. There is an implied contract that it will flow as expected, but we don't control external entities. The name of an external entity is a singular noun.

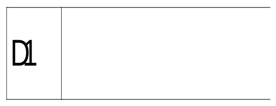
Process



A process transforms data flows in to data flows out. It is the only active component in the DFD. There must be at least one flow in and one flow out. The name of a process is traditionally an imperative sentence, i.e., a verb and object but no subject, since a subject implies who is performing the action. That would be a design decision. When the requirements analysis is for an object-oriented design, the name of a process can be a noun phrase describing the role and responsibility. For example, we could name a process Manage Inventory or we could name it Inventory Management or Inventory Manager.

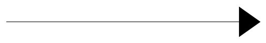
Processes are numbered hierarchically. The top level processes are numbered 1, 2, etc. The child diagram (explosion) of process 2 will have processes numbered 2.1, 2.2, etc. This numbering scheme facilitates navigation so that you can always know where you are in the leveled set of diagrams.

Data store



A data store contains a data structure at rest. Data stores are not necessary, but aid in understanding and documenting the important data resources of the system. It is a logical structure, not a physical structure. When considering Object-Oriented design, the data store will belong to only one class. In the DFD, it is shared by two or more processes, but only one of these processes "owns" the data. Data stores are named with plural nouns since they represent a collection if things.

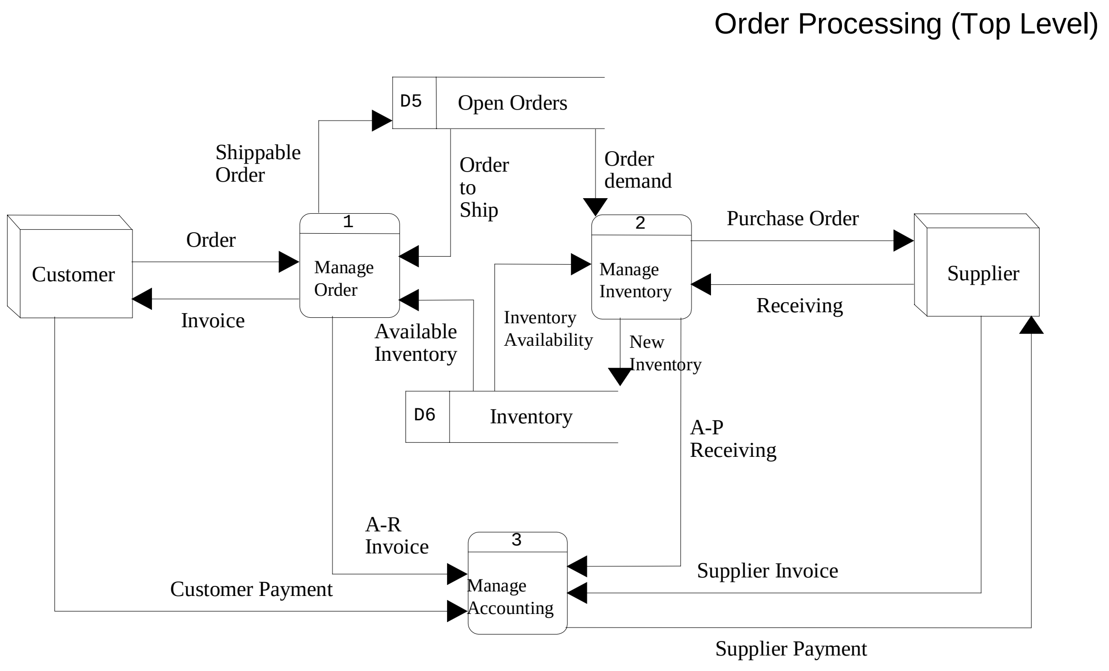
Data flow



A data flow is a data structure in motion. A data flow is named with a singular noun.

DFD Example - Order Processing Model

As an example of a DFD, consider a system with responsibility for managing customer orders, billing, inventory and general business accounting functions. The following is a top-level data flow diagram. The top-level DFD specifies the major components of the system under consideration -- functions, external entities and major data resources.



What does this diagram specify?

There are three major functions in the system under analysis -- **managing orders**, **managing inventory** and **managing accounting**. Two major external entities (not in the system but in the system environment) interact with the system -- the **Customer** and **Supplier**. The most important data resources for this system are the **Open Orders** and the **Inventory**.

The **Customer** sends in an **Order** to the system. The system responds with an **Invoice** indicating that the **Order** has been processed and the goods shipped to the **Customer**. Note: DFD's do not show the flow of materials, only information regarding their flow.

In order to satisfy the **Customer**'s request (**Order**), the **Manage Order** process must ensure that when an **Order** is due to be shipped, there is **Available Inventory**. The information about an **Order To Ship** is contained in the **Open Orders** data store, and the **Available Inventory** is contained in the **Inventory** data store.

The **Manage Inventory** process is responsible for ensuring that there is available inventory at the requested ship date. In order to accomplish this, the **Manage Inventory** process examines the **Order Demand** and compares it with the **Inventory Availability**. If there will not be enough inventory to satisfy the demand, a **Purchase Order** is sent to the **Supplier**. When the **Supplier** sends the ordered items, the **New Inventory** will be entered into the **Inventory** data store.

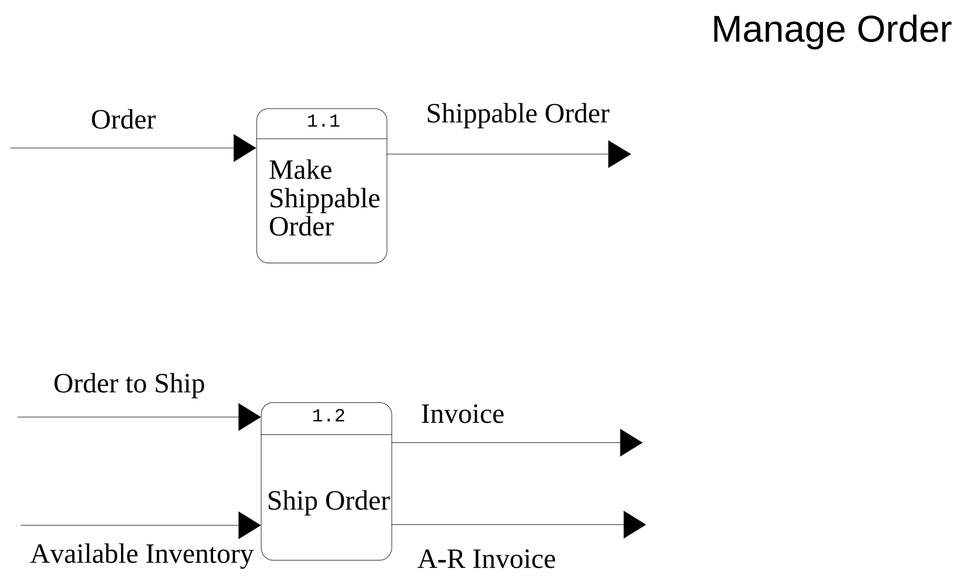
Since the Supplier will send an invoice expecting payment, the **Manage Accounting** process must be notified as to what it should pay for. When the order is shipped to the **Customer**, an accounts receivable will be established in the **Manage Accounting** function. When the **Customer** sends a **Payment**, the **Manage Accounting** function will process it.

DFD's Are Leveled Diagrams

The top-level data flow diagram describes the system requirements at the highest level. More specificity about a process is documented by exploding the process into a child diagram. Each diagram provides requirements for a particular subsystem. Each process may be viewed as a system. Thus the **Manage Order** process on the top level is a system (subsystem of the main system) and thus has a top-level DFD to specify its requirements (its child diagram). Every item on a diagram should be of equal importance (relevance) to that particular system view. This is the major principle of the leveling process in the structured analysis methodology.

For example, suppose we want to understand the requirements for the **Manage Order** process. We can explode it as is shown below.

Manage Order Explosion

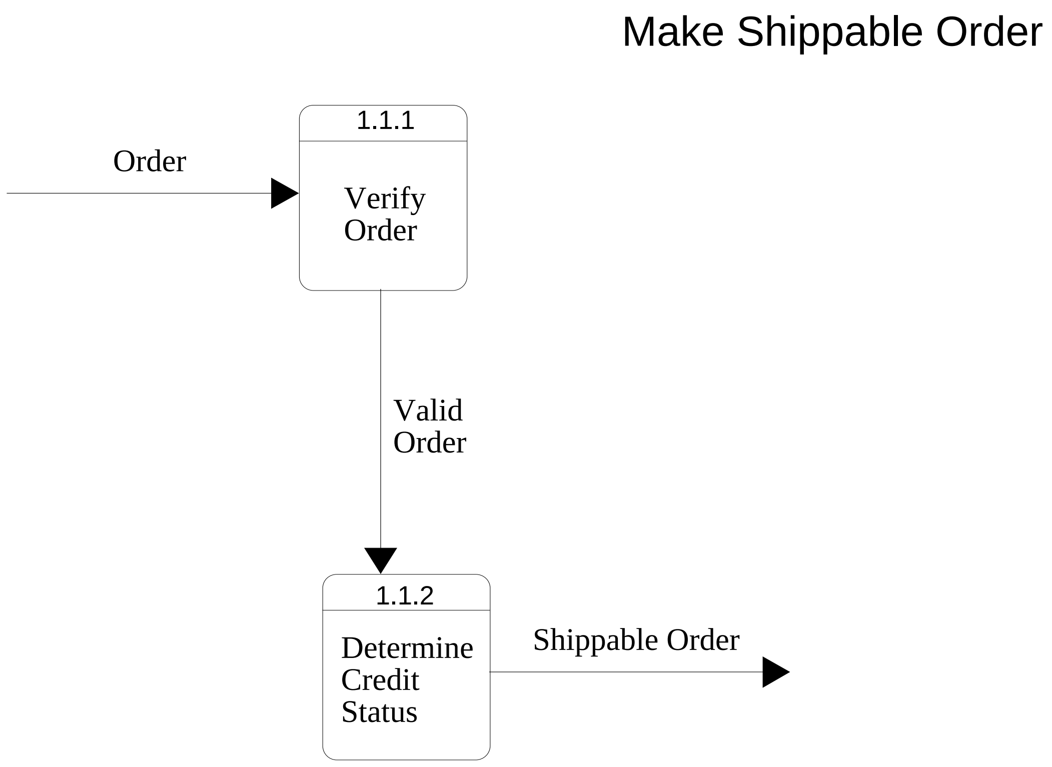


Level-to-level Balancing

A child diagram must account for every in-flow and out-flow of the parent process. This is known as level-to-level balancing. The **Manage Order** process has three in-flows and three out-flows. These are accounted for in the explosion. The child diagram must describe the requirements for transforming these in-flows into the out-flows.

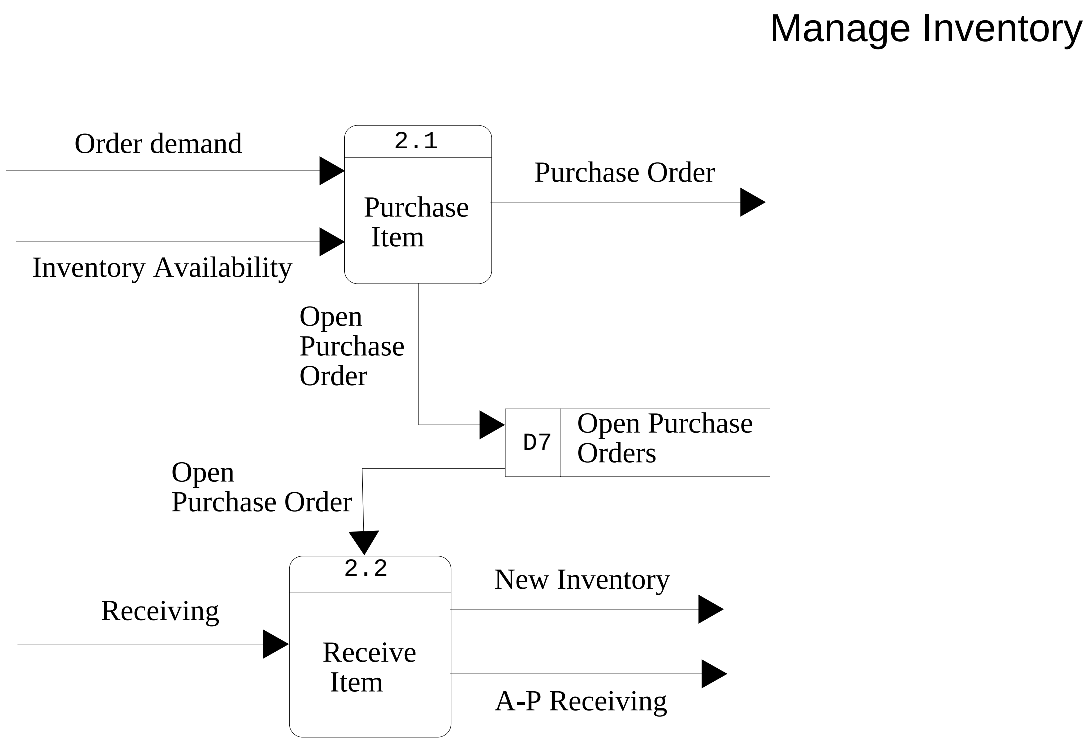
We will illustrate this leveling process further by exploding the **Make Shippable Order** process.

Make Shippable Order Explosion



Manage Inventory Explosion

When we explode the **Manage Inventory** process, we introduce a new data store to hold the **Open Purchase Orders**. This is an important data resource for the organization since it is an source of data for cash flow projections, i.e., how much we will expect to pay to suppliers and when these payments are likely to be required.


Manage Inventory
Order demand
2.1
Purchase Item
Purchase Order
Inventory Availability
Open Purchase Order
D7
Open Purchase Orders
Open Purchase Order
2.2
Receive Item
Receiving
New Inventory
A-P Receiving