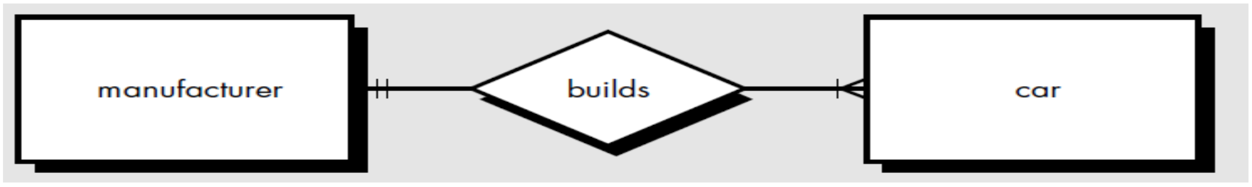
Entity Relationship Diagrams



The object/relationship pair can be represented graphically using the *entity/relationship diagram*.
A set of primary components are identified for the ERD: data objects, attributes, relationships, and various type indicators.
The primary purpose of the ERD is to represent data objects and their relationships.
Data objects are represented by a labeled rectangle. Relationships are indicated with a labeled line connecting objects. In some variations of the ERD, the connecting line contains a diamond that is labeled with the relationship. Connections between data objects and relationships are established using a variety of special symbols that indicate cardinality and modality.

FUNCTIONAL MODELING AND INFORMATION FLOW

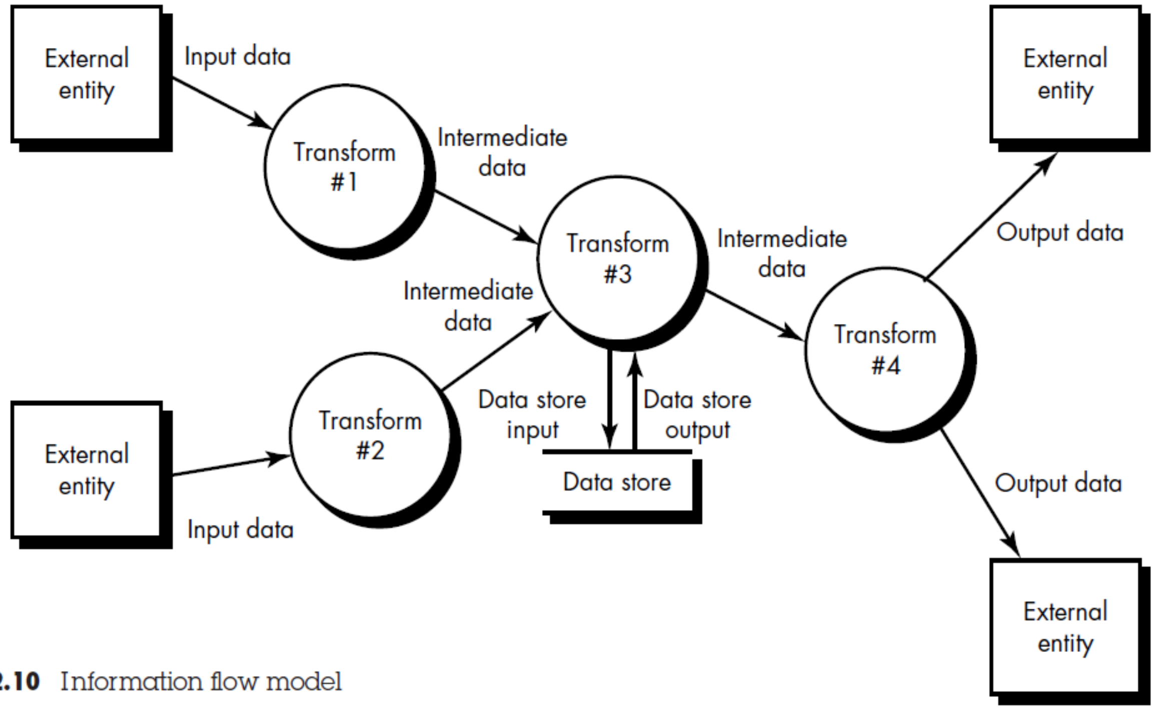
Information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms; applies hardware, software, and human elements to transform it; and produces output in a variety of forms.
Input:
-may be a control signal transmitted by a transducer, a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data file retrieved from secondary storage.
Transform:
-may comprise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system.
Output:
-may light a single LED or produce a 200-page report

We can create a *flow model* for any computer-based system, regardless of size and complexity.
Structured analysis began as an information flow modeling technique.

Data Flow Diagrams

As information moves through software, it is modified by a series of transformations.
A *data flow diagram* is a graphical representation that depicts information flow and the transforms that are applied as data move from input to output.
A rectangle is used to represent an *external entity;* that is, a system element (e.g., hardware, a person, and another program) or another system that produces information for transformation by the software or receives information produced by the software.
A circle (sometimes called a *bubble*) represents a *process* or *transform* that is applied to data (or control) and changes it in some way.

An arrow represents one or more *data items* (data objects). All arrows on a data flow diagram should be labeled.
The double line represents a data store—stored information that is used by the software.
The basic form of a data flow diagram, also known as a *data flow graph* or a *bubble chart.*
The data flow diagram may be used to represent a system or software at any level of abstraction.
DFDs may be partitioned into levels that represent increasing information flow and functional detail.
Therefore, the DFD provides a mechanism for functional modeling as well as information flow modeling.
A level 0 DFD, also called a *fundamental system model* or a *context model*, represents the entire software element as a single bubble with input and output data indicated by incoming and outgoing arrows, respectively.



12.10 Information flow model

Additional processes (bubbles) and information flow paths are represented as the level 0 DFD is partitioned to reveal more detail. For example, a level 1 DFD might contain five or six bubbles with interconnecting arrows.
Each of the processes represented at level 1 is a sub function of the overall system depicted in the context model.
Each of the bubbles may be refined or layered to depict more details.
Refinement from one DFD level to the next should follow an approximate 1:5 ratio, reducing as the refinement proceeds
A Data store represents some organized collection of data.

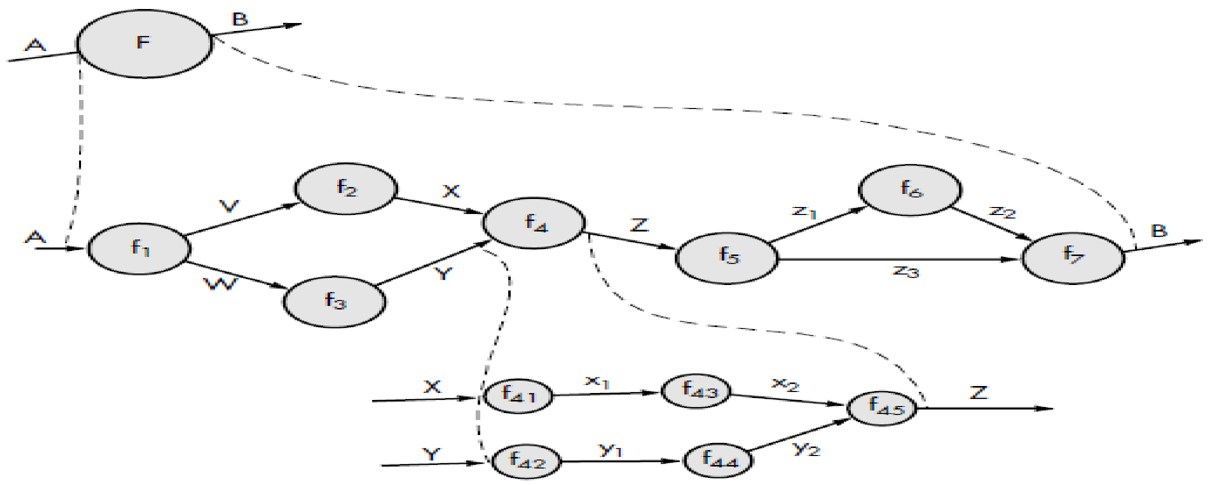


fig: Information flow refinement

DFD graphical notation must be augmented with descriptive text. A *process specification* (PSPEC) can be used to specify the processing details implied by a bubble within a DFD. The process specification describes the input to a function, the algorithm that is applied to transform the input, and the output that is produced. In addition, the PSPEC indicates restrictions and limitations imposed on the process (function), performance characteristics that are relevant to the process, and design constraints that may influence the way in which the process will be implemented.

****

BEHAVIORAL MODELING

Behavioral modeling is an operational principle for all requirements analysis methods but, only extended versions of structured analysis provide a notation for this type of modeling.

The state transition diagram represents the behavior of a system by depicting its states and the events that cause the system to change state.

STD indicates what actions (e.g., process activation) are taken as a consequence of a particular event.

A state is any observable mode of behavior.

A state transition diagram indicates how the system moves from state to state.

A simplified state transition diagram for the photocopier software is shown in figure below.

T he rectangles represent system states and the arrows represent transitions between states. Each arrow is labeled with a ruled expression. The top value indicates the event(s) that cause the transition to occur. The bottom value indicates the action that occurs as a consequence of the event. Therefore, when the paper tray is **full** and the **start** button is pressed, the system moves from the *reading commands* state to the *making copies* state.

Control Flow Diagram (CFD) can also be used for behavioral Modeling

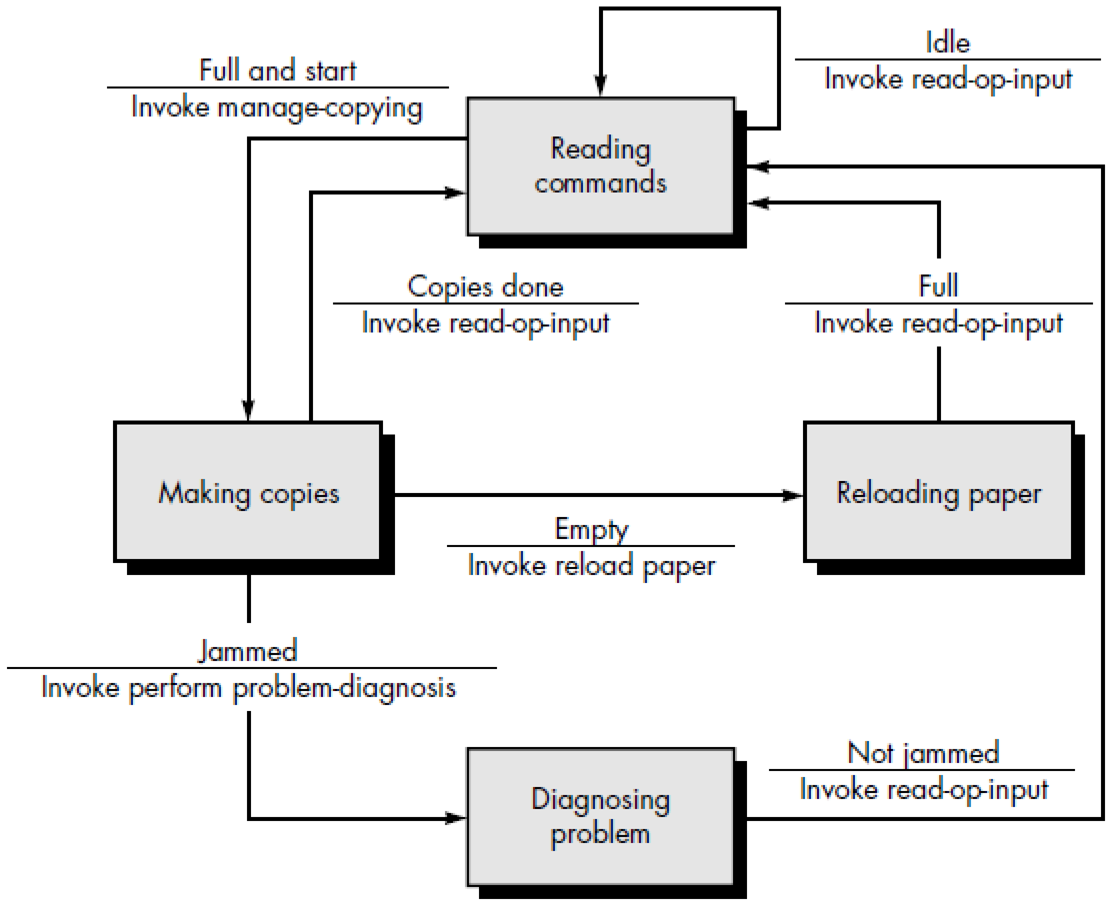


Fig: state transition diagram

****

The Mechanics of Structured Analysis

We examine each of the steps that should be applied to develop complete and accurate model using structured analysis.

1. Creating an Entity/Relationship Diagram

The entity/relationship diagram enables a software engineer to fully specify the data objects that are input and output from a system, the attributes that define the properties of these objects, and their relationships. Like most elements of the analysis model, the ERD is constructed in an iterative manner.

The following approach is taken:

1. During requirements elicitation, customers are asked to list the "things" that the application or business process addresses. These "things" evolve into a list of input and output data objects as well as external entities that produce or consume information.
2. Taking the objects one at a time, the analyst and customer define whether or not a connection (unnamed at this stage) exists between the data object and other objects.
3. Wherever a connection exists, the analyst and the customer create one or more object/relationship pairs.
4. For each object/relationship pair, cardinality and modality are explored.
5. Steps 2 through 4 are continued iteratively until all object/relationships have been defined.

6. The attributes of each entity are defined.
7. An entity relationship diagram is formalized and reviewed.
8. Steps 1 through 7 are repeated until data modeling is complete.
(Give example of any ER diagram now)

2. Creating Data Flow Diagram

The data flow diagram enables the software engineer to develop models of the information domain and functional domain at the same time.

<u>Basic Guidelines:</u>

(1) The level 0 data flow diagram should depict the software/system as a single bubble;
(2) Primary input and output should be carefully noted;
(3) Refinement should begin by isolating candidate processes, data objects, and stores to be represented at the next level;
(4) All arrows and bubbles should be labeled with meaningful names;
(5) Information flow continuity must be maintained from level to level, and
(6) One bubble at a time should be refined.
Write a PSPEC for each bubble in the final DFD
The refinement of DFDs continues until each bubble performs a simple function. That is, until the process represented by the bubble performs a function that would be easily implemented as a program component.

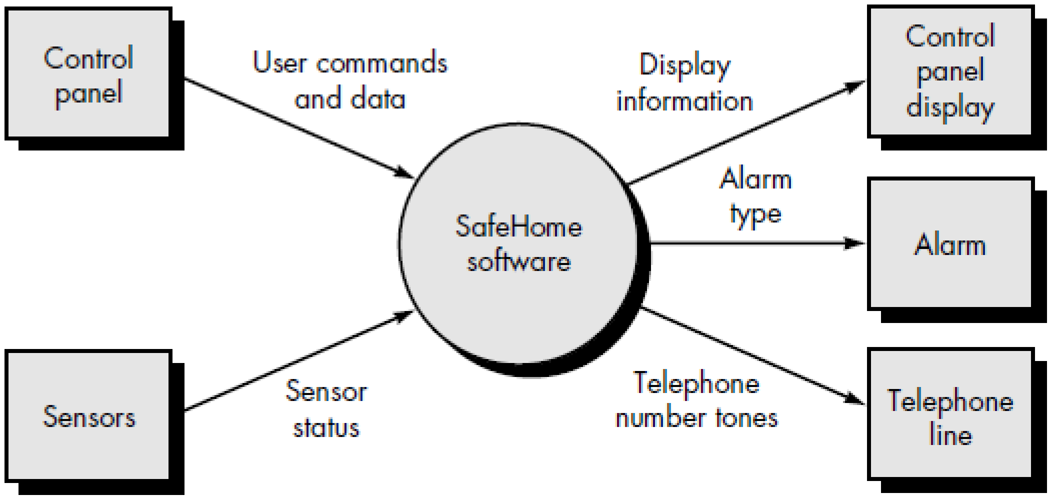
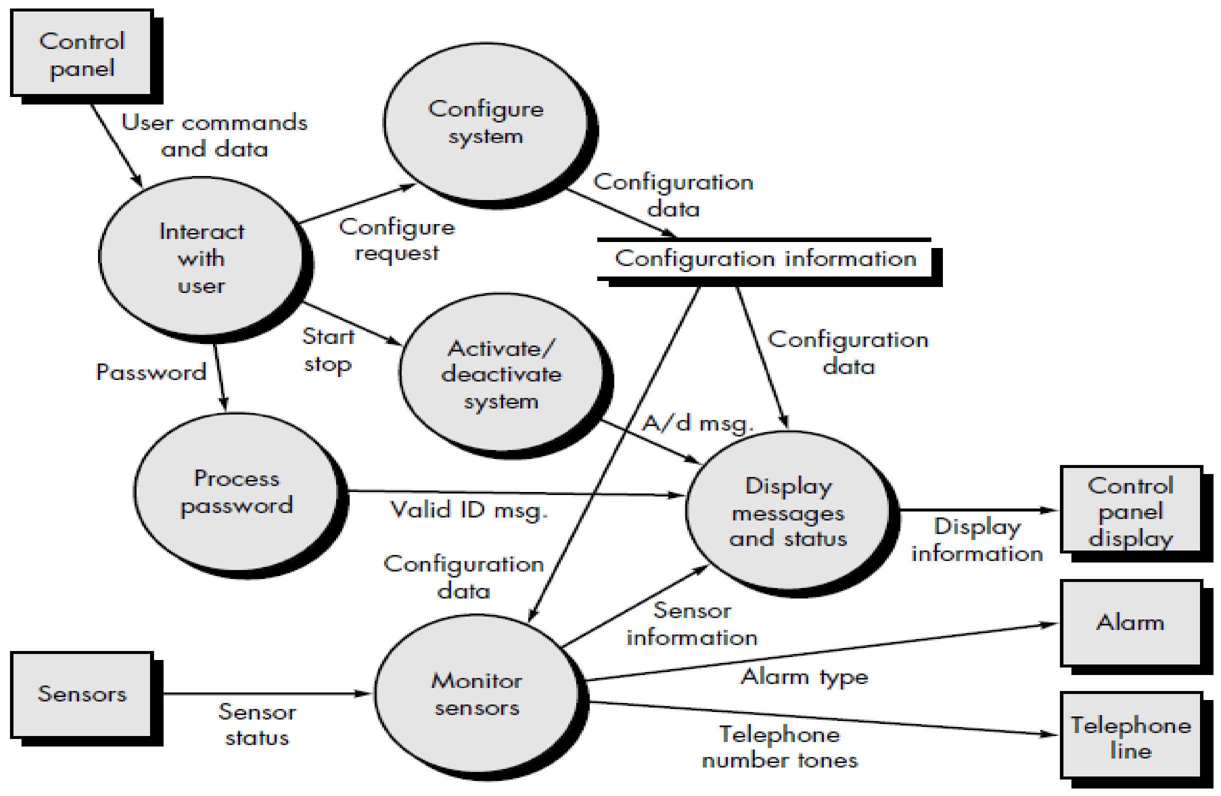


Fig: Context level DFD (level 0) for safe home



Level 1 DFD for SafeHome

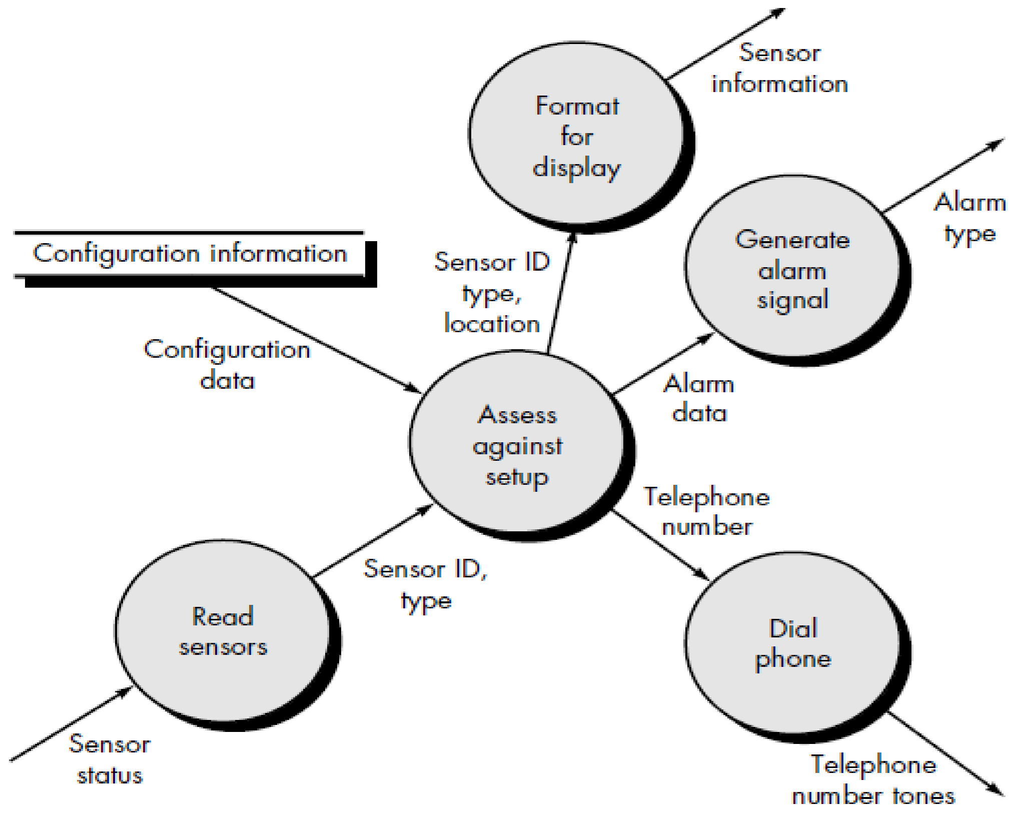


Fig: Level 2 DFD for safe Home

3. Creating a Control Flow Model

For many types of data processing applications, the data model and the data flow diagram are all that is necessary to obtain meaningful insight into software requirements. As we have already noted, however, a large class of applications are "driven" by events rather than data; produce control information rather than reports or displays. Such applications require the use of control flow modeling in addition to data flow modeling.

The graphical notation required to create a control flow diagram is given below.

To review the approach for creating a CFD, a data flow model is "stripped" of all data flow arrows. Events and control items (dashed arrows) are then added to the diagram and a "window" (a vertical bar) into the control specification is shown.

To select potential candidate events, the following guidelines are suggested:

List all sensors that are "read" by the software.

List all interrupt conditions.

List all "switches" that are actuated by an operator.

List all data conditions.

Recalling the noun/verb parse that was applied to the processing narrative, review all "control items" as possible CSPEC inputs/outputs.

Describe the behavior of a system by identifying its states; identify how each state is reached; and define the transitions between states.

Focus on possible omissions—a very common error in specifying control; for example, ask: "Is there any other way I can get to this state or exit from it?"

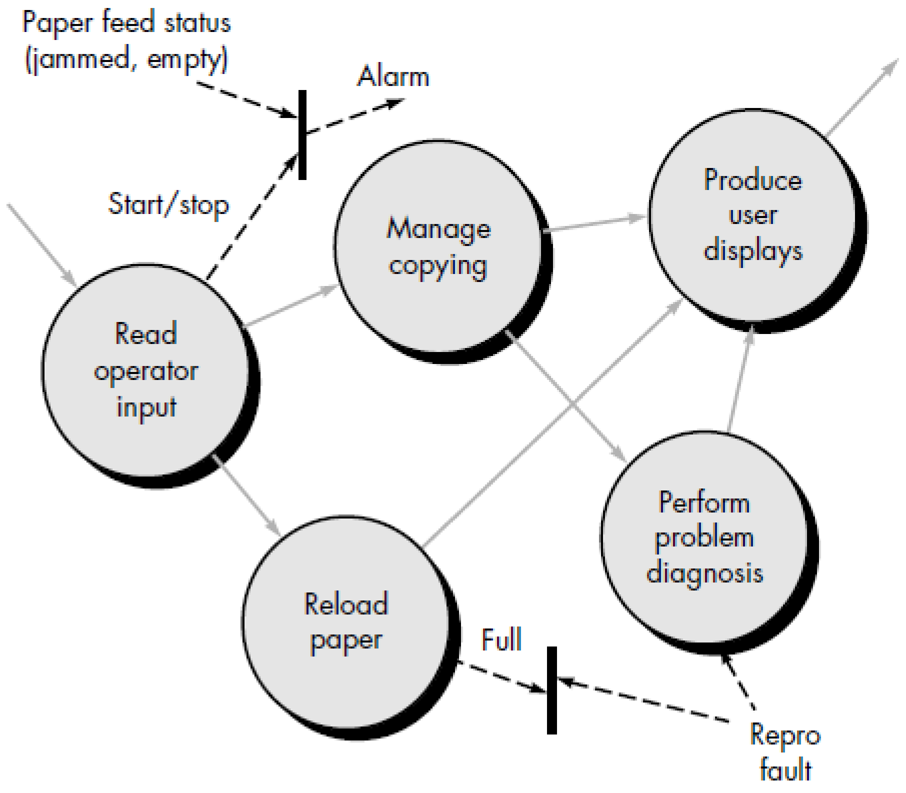


Fig: Level 1 CFD for photocopier software

THE DATA DICTIONARY

The *data dictionary* is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of stores and [even] intermediate calculations. Today, the data dictionary is always implemented as part of a CASE "structured analysis and design tool." Although the format of dictionaries varies from tool to tool, most contain the following information:

- *Name*—the primary name of the data or control item, the data store or an external entity.
- *Alias*—other names used for the first entry.
- *Where-used/how-used*—a listing of the processes that use the data or control item and how it is used (e.g., input to the process, output from the process, as a store, as an external entity.
- *Content description*—a notation for representing content.
- *Supplementary information*—other information about data types, preset values (if known), restrictions or limitations, and so forth.

Once a data object or control item name and its aliases are entered into the data dictionary, consistency in naming can be enforced. That is, if an analysis team member decides to name a newly derived data item **xyz,** but **xyz** is already in the dictionary, the CASE tool supporting the dictionary posts a warning to indicate duplicate names. This improves the consistency of the analysis model and helps to reduce errors.
"Where-used/how-used" information is recorded automatically from the flow models
When a dictionary entry is created, the CASE tool scans DFDs and CFDs to determine which processes use the data or control information and how it is used.
The notation used to develop a content description is noted in the following table:

Data Construct	Notation	Meaning
	=	is composed of
Sequence	+	and
Selection	[|]	either-or
Repetition	$\{\}^n$	n repetition of
	()	optional data
	…..	delimits comments

EXAMPLE: The data dictionary provides us with a precise definition of **telephone number** for the DFD in question. The data dictionary entry begins as follows:

- name: telephone number
- aliases: none
- where used/how used: assess against set-up (output)
- dial phone (input)
- description: telephone number = [local number/ long distance number]
- local number = prefix + access number
- long distance number = 1 + area code + local number
- area code = [800 | 888 | 561]
- prefix = *a three digit number that never starts with 0 or 1*
- access number = * any four number string *

For large computer-based systems, the data dictionary grows rapidly in size and complexity. In fact, it is extremely difficult to maintain a dictionary manually. For this reason, CASE tools should be used.