Design elements:

1. ***Data Design:-***

It is the first of three design activities that are conducted design software engineering. The impact of data structure on program structure and procedural complexity causes data design to have a profound influence on software quality.

At the component level:

- Refine data objects and develop a set of data abstractions
- Implement data object attributes as one or more data structures
- Review data structures to ensure that appropriate relationships have been established
- Simplify data structures as required

2. ***Architectural Design:-***

The primary objective of architectural design is to develop a modular program structure and represent the control relationships between modules.

The designer specifies the structure of the system by defining and refining software components. Each style describes a system category that encompasses:

(1) a set of components (e.g., a database, computational modules) that perform a function required by a system,
(2) a set of connectors that enable "communication, coordination and cooperation" among components,
(3) constraints that define how components can be integrated to form the system, and
(4) semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

3. ***Procedural Design:-***

Procedural design occurs after data and program structure have been established. The procedural specification required to define algorithmic details would be stated in a natural language such as English. Once the information domain has been represented (using DFDs and DD), the

analyst describes each function (transform) using natural language or some other notation. Such notation is called structural English (also called process or **program design language PDL**).

The PDL is a pidgin language in that it uses the vocabulary of one language (i.e. English) and the overall syntax of another (i.e. structural programming language). The difference between PDL and a real high order programming language lies in the use of narrative text (e.g. English) embedded directly within PDL statements given the combined use of narrative text embedded directly into a syntactical structure, PDL cannot be complied. However, PDL "processors" currently exists to translate PDL into a graphical representation (e.g. a flowchart) of design and to produce nesting maps, a design operation index, cross reference tables, and a variety of other information.

Top-Down and Bottom-Up Design:

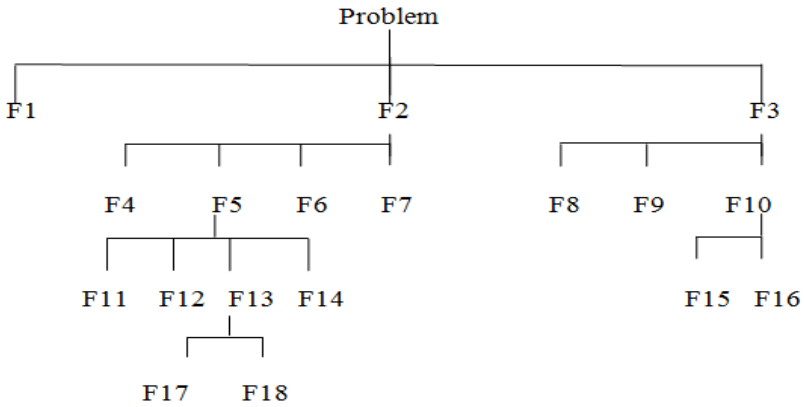
In top down strategy, each step, one of several functions or tasks is decomposed into a number of sub functions or sub tasks. This strategy, sometimes called functional decomposition, Develops the architecture of a program by successively refining levels of procedural details. The refinement can be continued through intermediate level design, where by more and more details developed concerning how the software is to be organized. Finally, if the process is continued to low level design, it will terminate when all instructions are expressed in pseudo-code or a computer language.

As each function or task is refined, the data may also need to be refined, decomposed, or structured to accommodate the means of communication between the sub functions and tasks. Program and data refinement should be done in parallel with each level of refinement.

An important feature of top-down design is that at each level of refinement, the details of the design at lower levels of refinement are hidden. Only the data and control information required to interface with the next level need to be defined. If a data structure is local to a lower level function or task, it need not be specified until that level is reached in the design process. However, if a data structure is shared by a several functions or tasks at the same level, it must be specified before proceeding to a lower level in the design.

The design process is one of trial and error. As we make refinement, we may eventually wish to modify a previous design n decision. This modification of design may require us to change other parts of the design and associated data structures. The modification should be done in a systemic way: one technique is to backup to an earlier level in the design that the modification does not effect and redo the entire top-down refinements, taking the modification into account.

Backing up in this manner is essential if a correct design is to be achieved. The more complicated the software system, the more important it is to back up in a systematic manner. This goal of bottom-up design is to establish a set of general-purpose resources while maintaining application independence. Bottom-up design proceeds from the specific software development environment by making available more powerful instructions to the software developers. Suppose that a top-down analysis has produced, the refinement illustrated in the following figure:



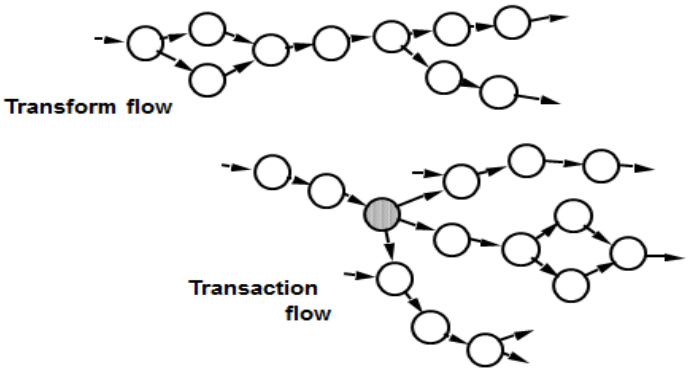
Where F1 through F18 represent program units or pseudo code for the operations that need to be performed, suppose that in attempting to refine F18 , we discover an error, or we learn that a change in data structure will make F18 more efficient in terms of execution speed and\or storage. To make the change, we must backup to the point where the change has no effect. If this is F13 (i.e. F17 and F18). If this is F2, we must redo all refinements of F2 (i.e. F4-F7, F11- F14, F17 and F18).

Structured Design

Objective: **T**o derive program architecture that is partitioned

Approach: The DFD is mapped into program architecture

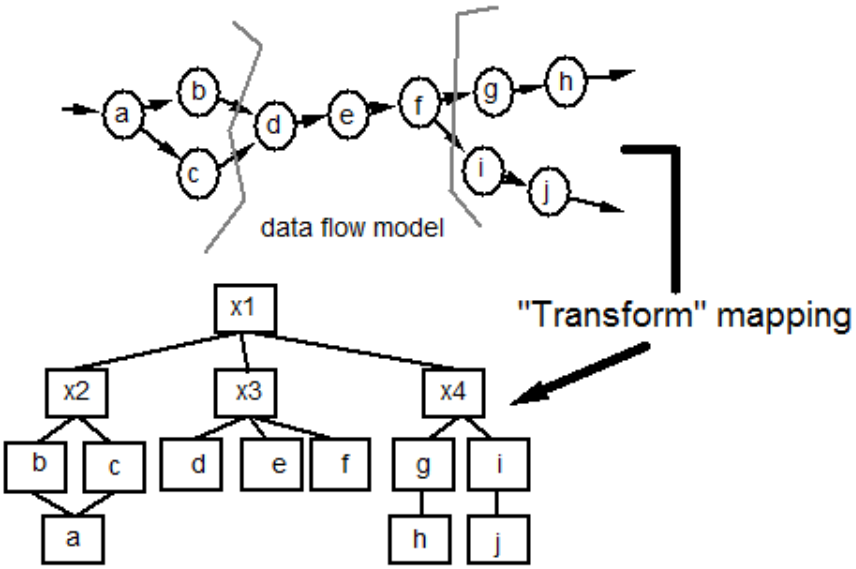
Flow Characteristics:



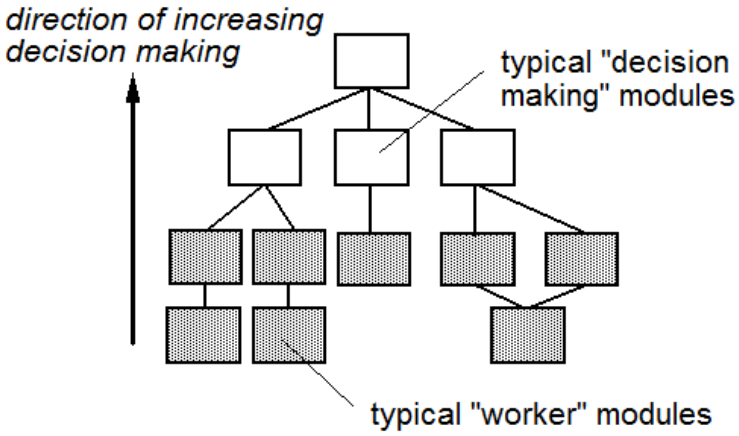
General Mapping Approach:

- In ***Transform flow*** isolate incoming and outgoing flow boundaries; for ***transaction flows***, isolate the transaction center.
- Working from the boundary outward, map DFD transforms into corresponding modules.
- Add control modules as required
- Refine the resultant program structure using effective modularity concepts

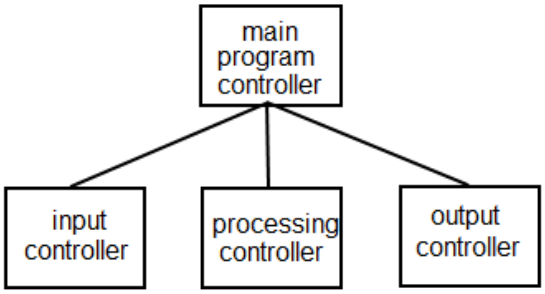
Transform Mapping:



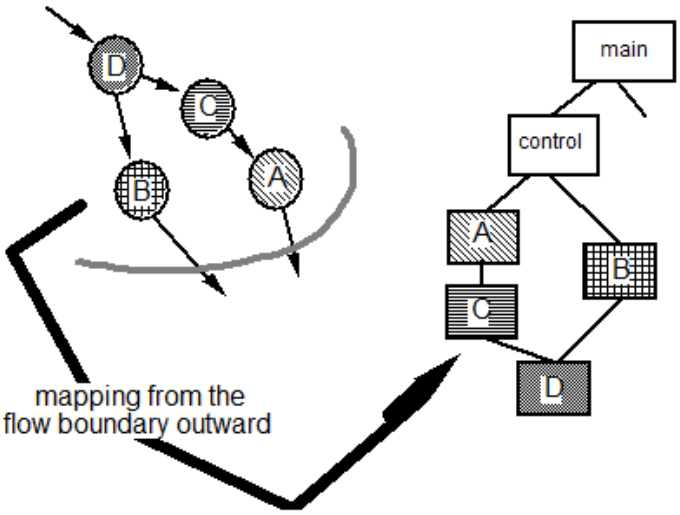
Factoring



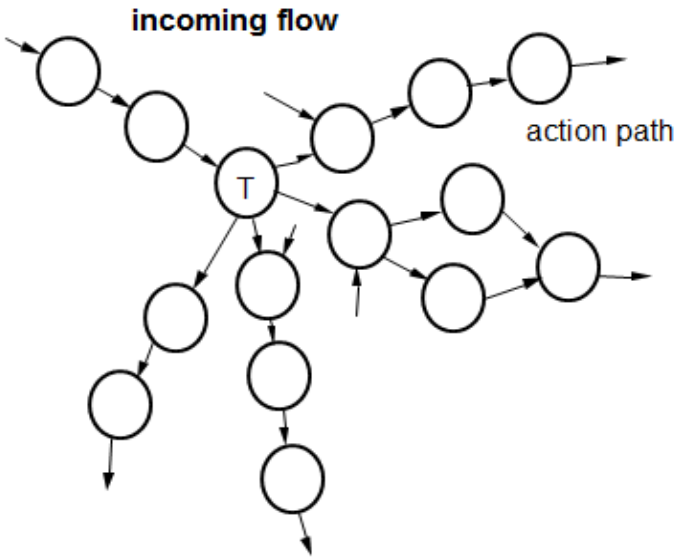
First Level Factoring:



Second Level Mapping:



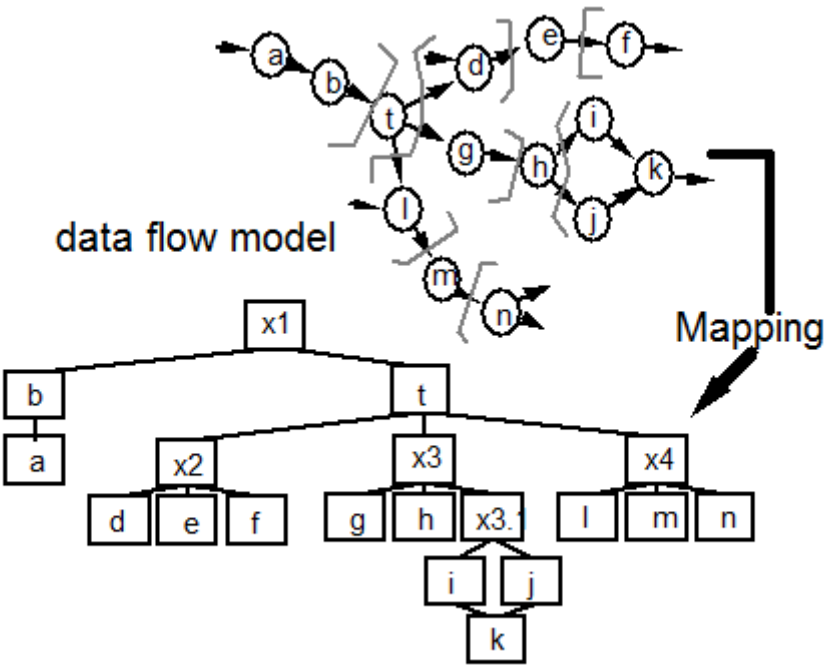
Transaction Flow:



Transaction Mapping Principles:

- Isolate the incoming flow path
- Define each of the action paths
- Assess the flow on each action path
- Define the dispatch and control structure
- Map each action path flow individually

Transaction Mapping:



Transaction Example