

Designing the Program

From DFD to SC

- Transaction Analysis
- Transform Analysis

System Design

- What we have till now
 - System Requirements
 - Statement of Purpose
 - Event List
 - ERD (Data)
 - DFD (Process)
 - Mini Specifications (Process Logic)
 - “WHAT” the system should do, is determined
 - Architectural Overview
 - Bird’s eye view of whole system

Creating Blueprints for Actual Construction

- Construction blue print is the next step, which includes “HOW”
- Data Perspective
 - Conceptual (specified in requirements) to Logical Model
 - Logical Model into Physical Data Structure
- Process Perspective
 - DFD into Structure Chart (program specification)
 - Module specification
 - Forms & Report
 - User Interface

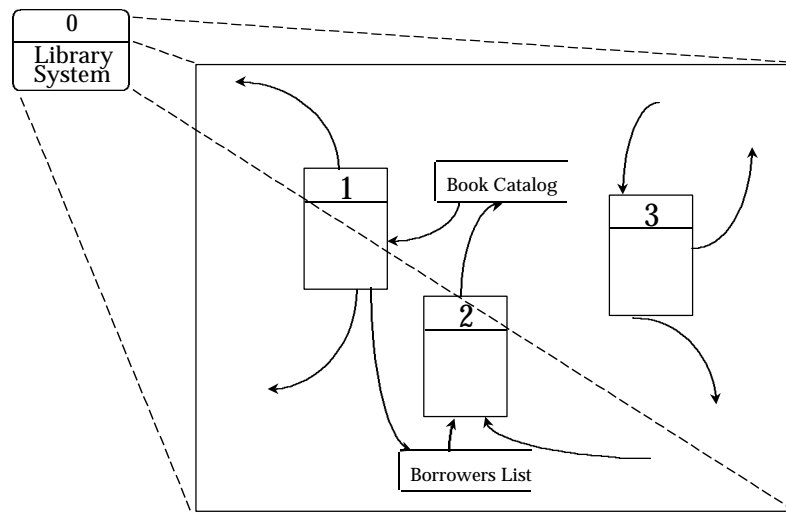
Process-oriented Design Activities

- Conduct Transformational and/or Transactional Analysis
 - identify all transactions and see if there is common processing
 - do while also doing transformational analysis
- Develop Structure Charts
- Package Program Units
- Write Program Specifications

Structured Design

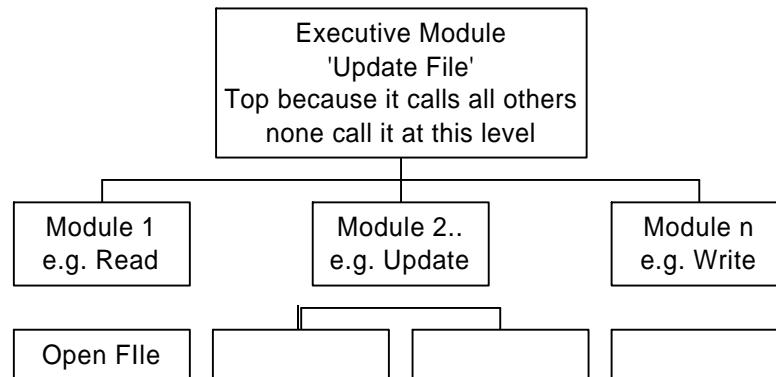
- Notation
 - Structure Charts
 - Pseudo Code
- Heuristics
 - Mapping procedures
 - Design evaluation guidelines

Overview Data Flow Diagram



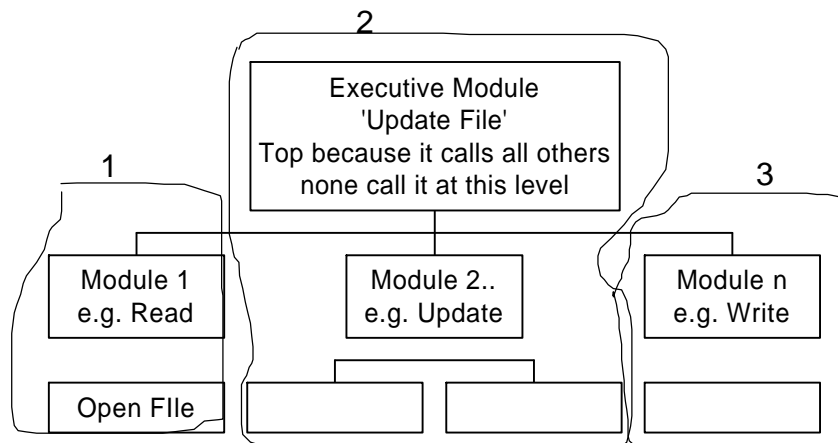
Structure Chart

Q: Where do the modules come from?

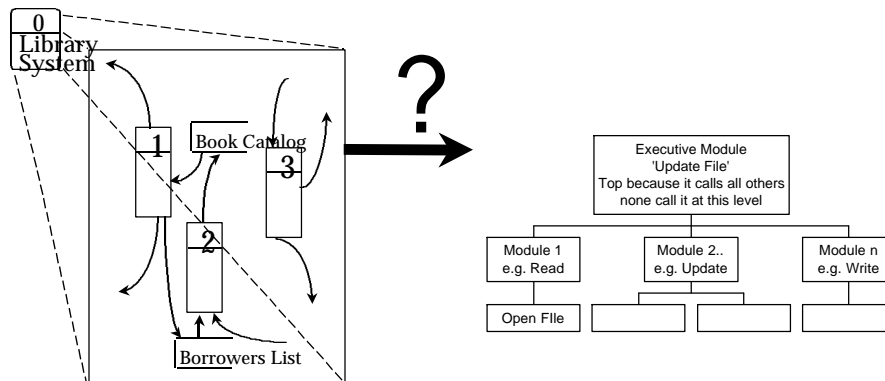


Structure Chart Example

Q: Where do the modules come from?



DFD to Structure Charts?



Converting Data Flow Diagrams to Structure Charts

- Processes or groups of processes become Modules
- Basic Issues:
 - 1) not all DFDs are converted
 - 2) generally the lower level DFDs provide sufficient detail
 - >top level DFDs are too general yet may describe the control modules
 - 3) several levels along the 'explosion path' require conversion to STCs

Converting DFDs to STCs

General Steps:

- Obtain a first cut STC
 - 1) identify the set of DFDs to convert to STCs
 - 2) identify the 'boss' (control) module
 - 3) perform transform/transactional analysis on DFDs
- Refine the first cut STC
 - 4) insert data flags and /or control flags as needed
 - 5) determine the cohesiveness of the control module
 - 6) perform coupling/decoupling activities on program modules
 - 7) evaluate the span of control for program modules

Identifying the 'Boss' Module

- Select the DFD with the most I/O occurrences (the largest number of flows in and out)
- if no single process with the largest number exists, choose the area where two or more processes with the largest number of flows. Then either:
- select one of the flows as the 'boss', or
 - create a new artificial 'boss' and connect it to the process with the most flows

Identifying the 'Boss' Module

- The 'boss' module is the functional equivalent of the control routine in a structured program
- The only purpose of the 'boss' module is to control calls to the remainder of the STCs.
- Therefore, it should not be a module that itself decomposes/explodes as a peer of other modules at the same level
- may need to choose from the parent DFD
- Summary: The 'boss' may be a process from an upper (parent) level from which explosion occurs

Partitioning the Data Flow Diagram

- Transformation-centered Application
 - relates to linear/sequential tasks
 - transforming afferent flows into efferent flows
 - transformation of data is central
- Transaction-centered Application
 - multiple transaction types
 - logical OR (selection) situation where values returned by data or control flags triggers the selection
 - case structure analysis with multiple procedures which do not always get called
 - transaction is central

Rules of Transform Analysis

- Basically...
 - identify central transform, afferent and efferent flows
 - create a first-cut structure chart
 - refine the high-level structure chart
 - decompose processes into functions
 - refine the STCs iteratively

Steps in Transform Analysis (1)

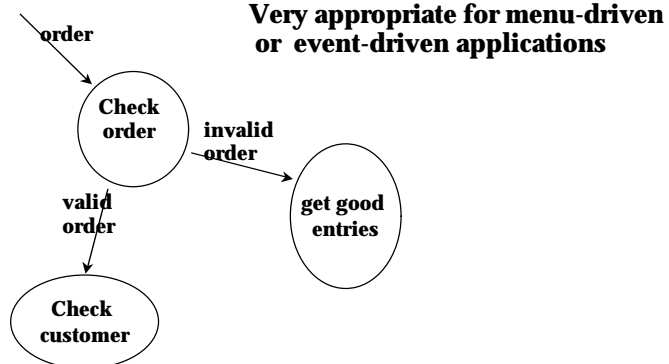
- 1) identify central transform, afferent and efferent flows
- 2) draw the boss process at the top, to which afferent flows, central transforms and efferent flows are dangling
- 3) convert data stores and external entities into STC input or output (I/O) modules
- 4) make calls
- 5) rename modules to indicate how they occur
- 6) add data flag symbols to indicate data objects being passes along the call path
- 7) Add control flags as needed

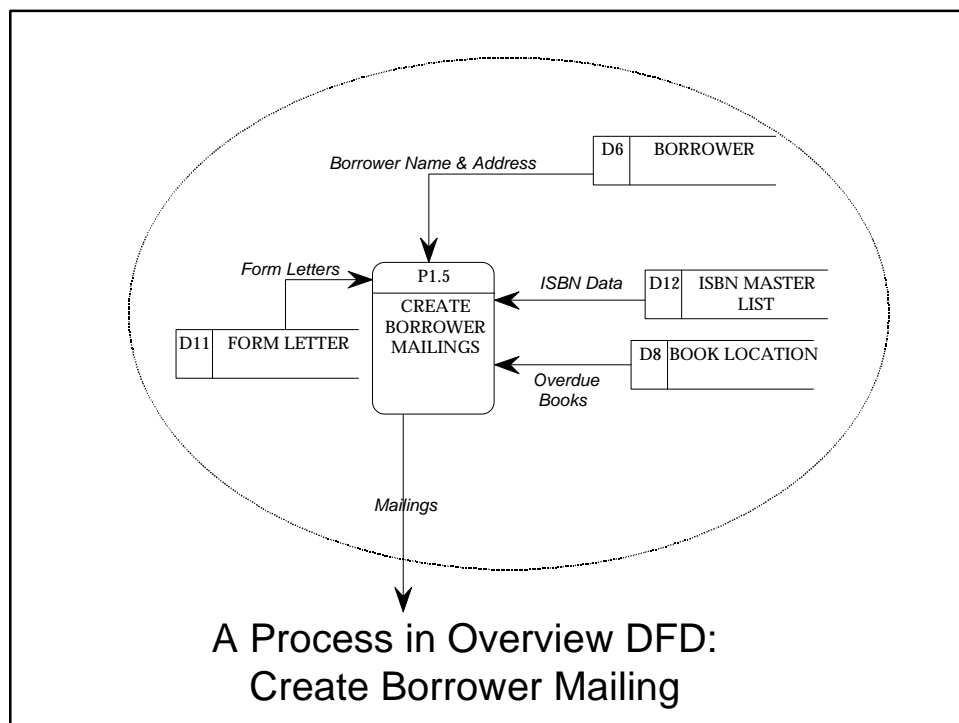
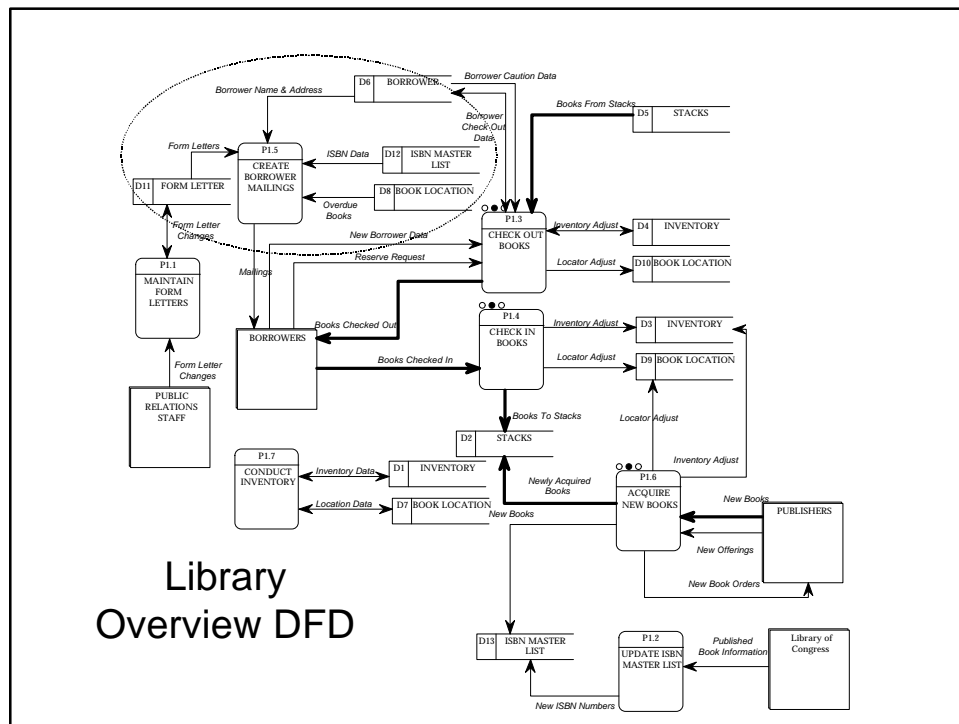
Rules for Transaction Analysis

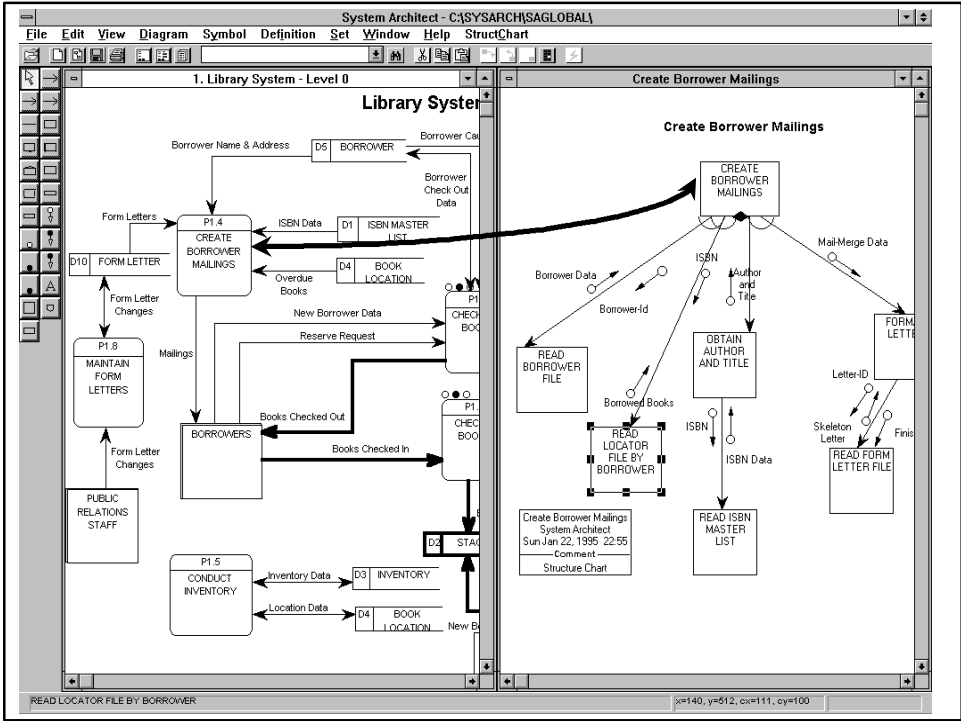
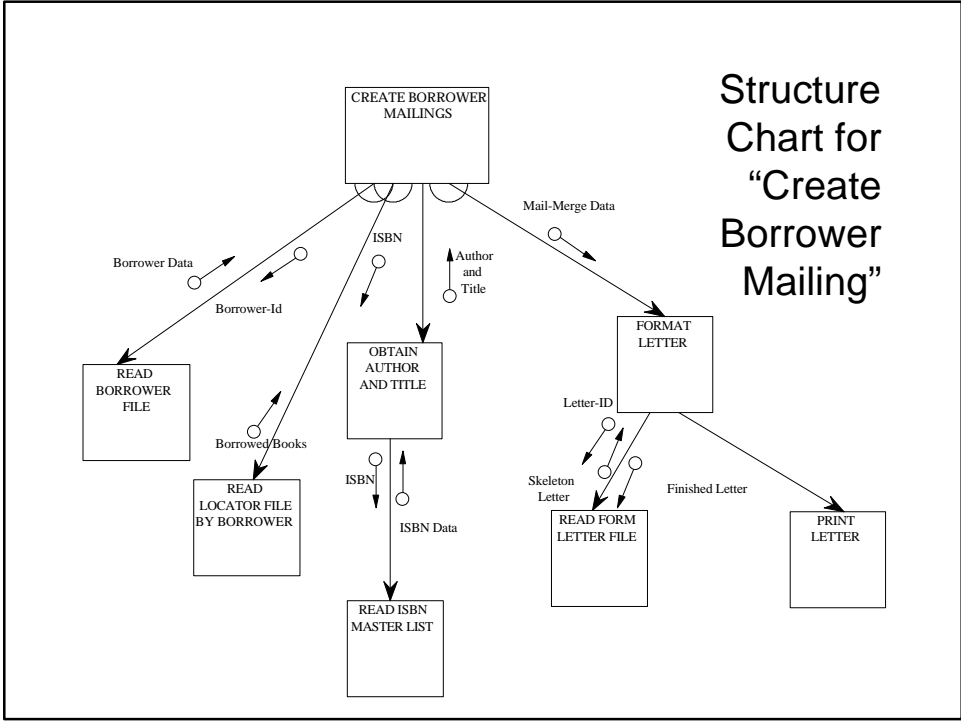
- define transaction types and processing
- develop a structure chart with transaction center as a boss module
- further define structure chart details
- transaction branches may contain transform

Transaction Analysis

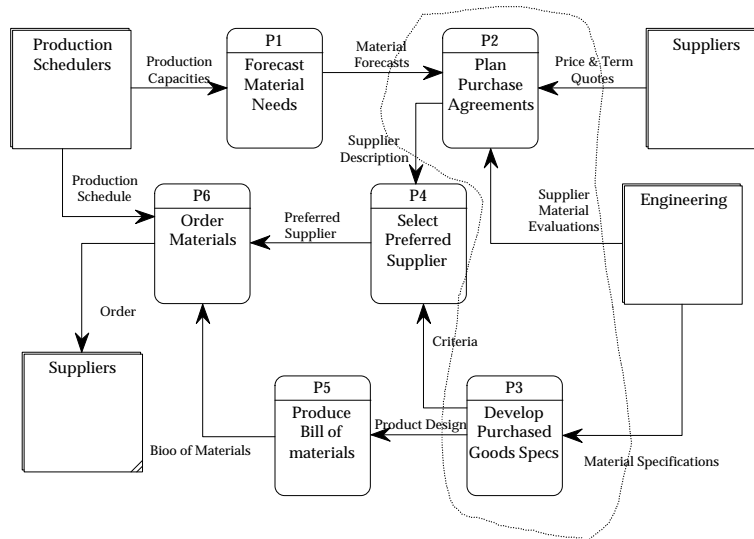
- detect places in a DFD where the data flow leaving or going to a process do not all occur
 - situation where a process depends upon a condition being met







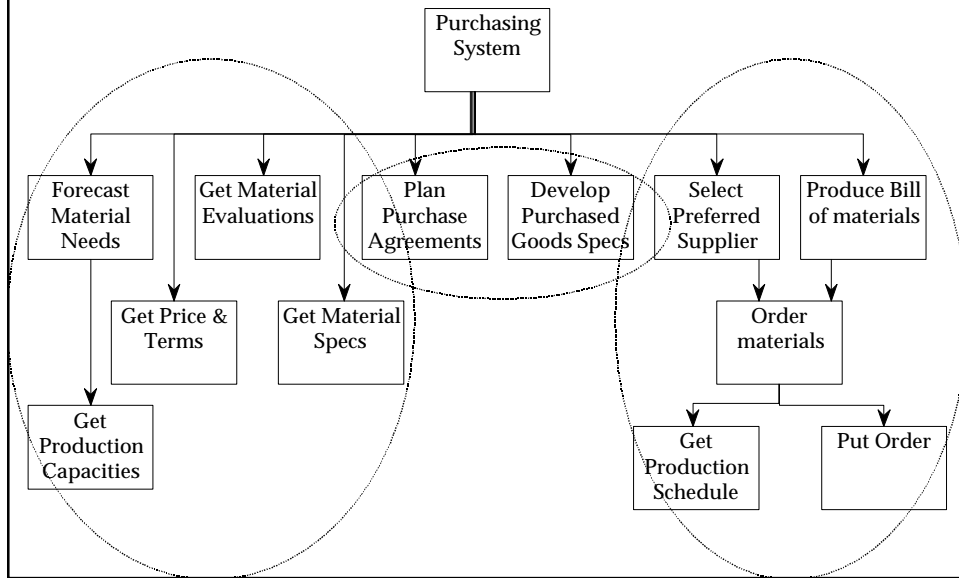
Furniture Company Purchasing Process



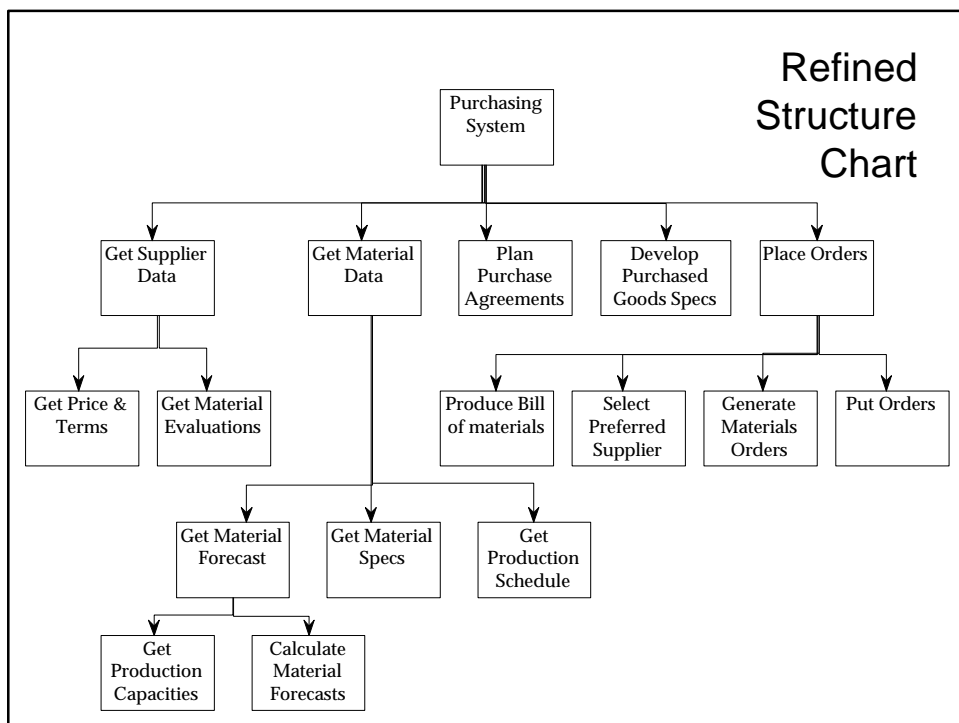
Identifying Central Transform

- Process 1: single input and a single output
- Process 6: using just existing data
- Process 4: selection rather than transform
- Process 5: takes product designs and converts into bills of material: but one input and one output
- Working inward by process of elimination
- Central transform: 2 and 3
- Why both? 2 and 3 are independent of each other, both serve as starting points for streams of transformed data to the assembly point: process 6

First Cut Structure Chart (Dangling from CT)



Refined Structure Chart



General Mapping Approach

- Bound “group” processes
 - Isolate incoming and outgoing flow boundaries
 - for transaction flows, isolate the transaction center
- Working from boundaries outward, map DFD transforms into corresponding modules
- Add control modules as necessary
- Refine program structure to account for good modularity
 - i.e., cohesion and coupling

Basic Design Goals

- Fitness for purpose
 - the system must work, and work correctly
 - it should perform all tasks as specified within the constraints of the resources
- Robustness
 - the design should be stable against changes to features such as file or data structures

Three Design Principles

- **Simplicity**
 - the design should be simple as possible, but no simpler
- **Separation of Concerns**
 - the different concepts should be separated out
- **Information Hiding**
 - information about the detailed form of objects (e.g., data structures) should be kept local and “visible” to outside modules

Specific Design Guidelines

- **Cohesion**
 - the degree modules are sufficient to carry out one, single, well-defined function. A measure of internal strength. Each module is a system unto itself.
 - Coupling
 - Module independence preferred to tight interdependence. (e.g. less is better)
- **Information Hiding**
 - only the data needed is made available to each module
- **Modularity**
 - small self-contained units for maintainability; each module is a system unto itself

Specific Design Guidelines (cont'd)

- Module Size
 - reasonable size
- Span of Control
 - calling of others: known and limited: generally seven
- Scope of Effect/Control
 - clear path of relationships in the hierarchy

Factoring

- is a process of decomposing a DFD into a hierarchy of program components which eventually become modules, functions, and control structures
- examine each DFD stream and analyze the IPO structure
- places each unbroken strand of processes in DFDs into its own control structure and creates new control processes for split strands at the point of the split

Deliverables

- Structure Charts
 - Fully factored
 - Complete descriptions of data couples and control flags
- Module Specifications
 - Input Specifications
 - Processing Specifications
 - Output Specifications

Module Specification Methods

I/O Spec
Psuedo Code

Module Specification

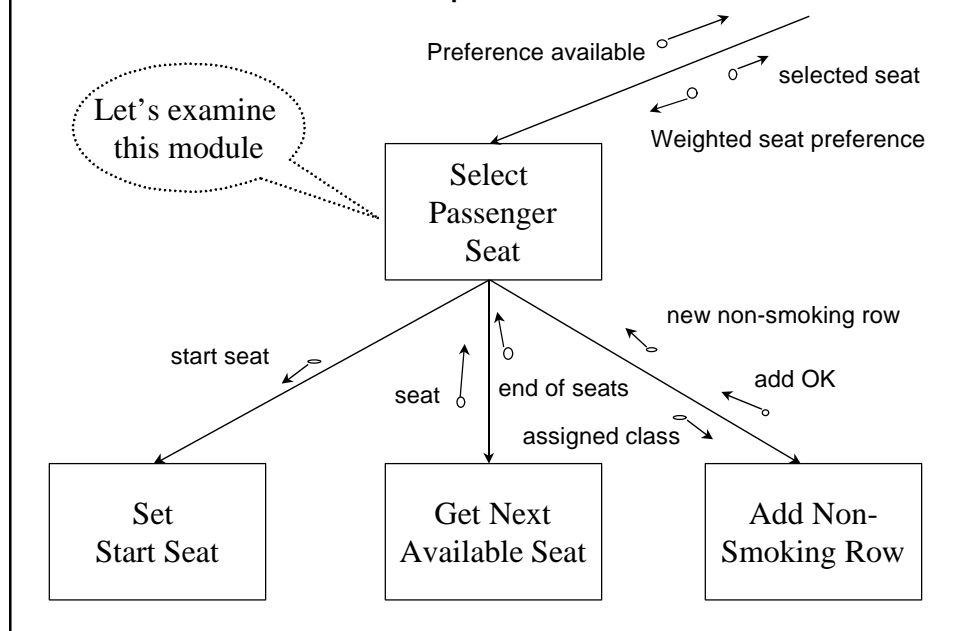
◆ Two Common Methods

- I/O Spec
- Pseudo Code
- (also, Structure English, but not preferred)

◆ Purpose

- Guide programmer without doing writing code
- Document module/program
- Describe what to do
 - * sometimes move towards how to do it

Example Module



I/O Specification

Module: Select Passenger Seat
Purpose: To choose for a customer a seat that is valid for his class and is as close as possible to his (weighted) seating preference
Uses: weighted seat pref
Returns: selected seat, pref available

Functional Details

Scan the available seats, beginning in the passenger's assigned class and working to lower classes.

Note for each set its differences from the customer's preference

Select the seat with the least difference: this is `selected_seat`. ($\text{Difference} = \text{smoking_diff} * \text{smoking_weight} + \dots$)
etc.

Pseudo Code Specification

- Concept: More detailed procedural language independent way to describe module.
 - Similar to code, but not.
 - Less programmer margin for error.

Pseudo Code Specification

Module: Select Passenger Seat
Purpose: To choose for a customer a seat that is valid
 for his class and is as close as possible to his (weighted)
 seating preference
Uses: weighted seat pref
Returns: selected seat, pref available
Begin
 For each CLASS downward from passenger's ASSIGNED_CLASS
 Call Get First Seat (CLASS, START_SEAT)
 Repeat
 Call Get Next Available Seat(SEAT, END_OF_SEATS)
 Until END_OF_SEATS = "Y"
 EndFor
 etc.....
End

A Preferred Module Specification Format

Module Name: Discombobulate Ringhadffers
Module Purpose: To do whatever...
Uses: the incoming data
Returns: names outgoing data elements or structures
Begin
 FOR each module
 a complete descriptions of the
 appropriate programming logic in
 a suitable pseudocode form
 show initializations where required
 show any unusual or designer preferred algorithms
 indicate program calls and the data sent and returned from
 the module invoked
 etc..... END FOR
End Proc