CS360
Business Information Systems
Analysis and Modeling

Process Design
Process Design

- USD’s, CD’s, ERD’s describe the relationship of data and responsibilities
- ERD’s define the information structure
- CD’s define the responsibility of classes for information processing
- Process design defines what happens in the interaction of classes
- Process design depends on
  - the granularity of the processing tools (3GL, 4GL, DML)
  - the mode of software component integration
“Programmable” offers more flexible application of system functions and requires more responsibility / discipline from the users in defining and applying system functions (“field configurable”)

“Transaction” both limits the users’ options and control as well as tightly binds the system’s behavior to the requirement (“factory installed”)

The choice depends on the users’ capabilities and requirements
System Integration

* System integration:
  * the quality of inter operation of a system's various specific (business) functions
  * the convenience of specifying / controlling the inter operation among system components

* Integration factors:
  * information / file exchange between system functions
  * formats, capacities, speed
  * "transaction model" support across components
  * concurrent access to data / transactions
  * journaling, audit trails and business transaction recovery

* "transparency" of process control
  * manual versus automatic
Software Scope Models

- **Vertical software:** provides focused function on very specific application requirement

- **Horizontal software:** provides broad application utility without “complete” application support

- **Integrated software:** collects functions from the former and attempts to provide a “turnkey” collection of function to a customer or business “class”

- Each has a strength and weakness relating to breadth and depth of function
Basic OO Process Design

1. List user visible USD, UC, and Menu functions
2. Match system “outputs” to a class in CD
3. Match each “inputs” to a class in CD
4. Identify behaviors based upon class responsibility (may require new classes):
   4.1. Add “maintenance” for each class
   4.2. Add “control” for each class
5. Devise a “set of actions” for each behavior
   5.1. What is the effect of each “action?”
   5.2. In what order do the “actions” occur?
   5.3. Document in a sequence diagram
6. Pseudocode behaviors needing programming
7. Denote “real” software with system flowcharts
8. Update system resource requirements
9. Update data dictionary with new processes