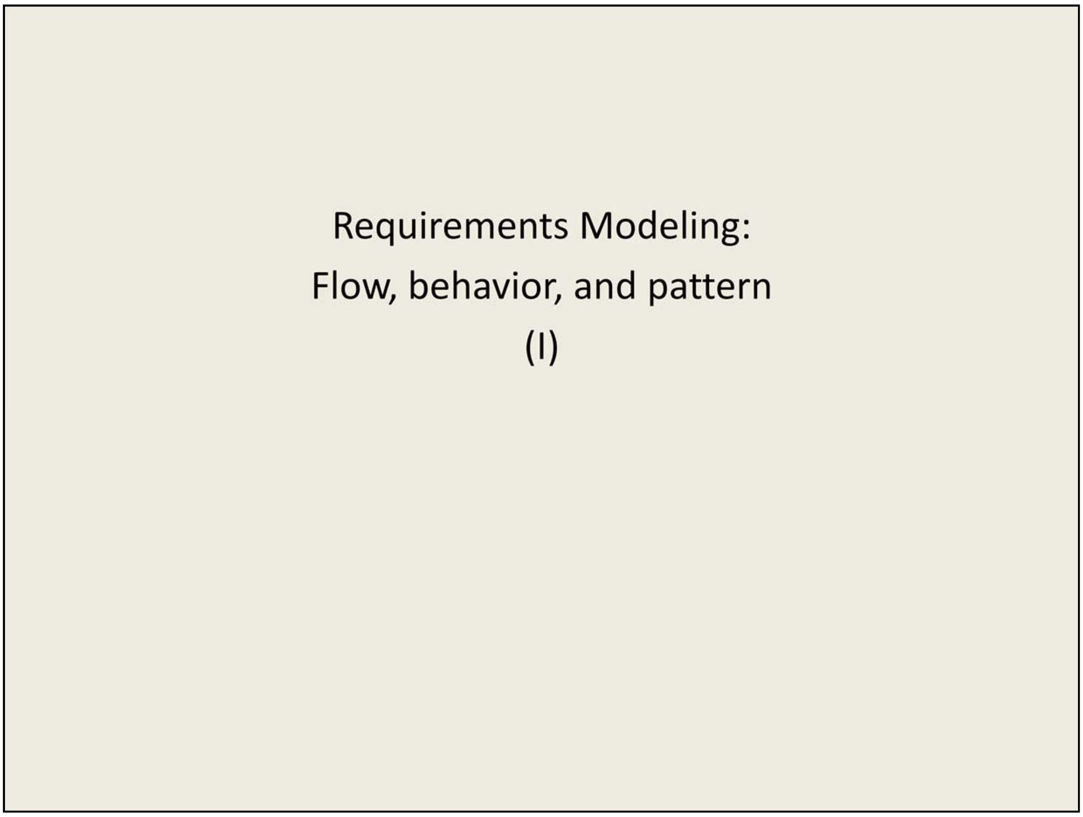
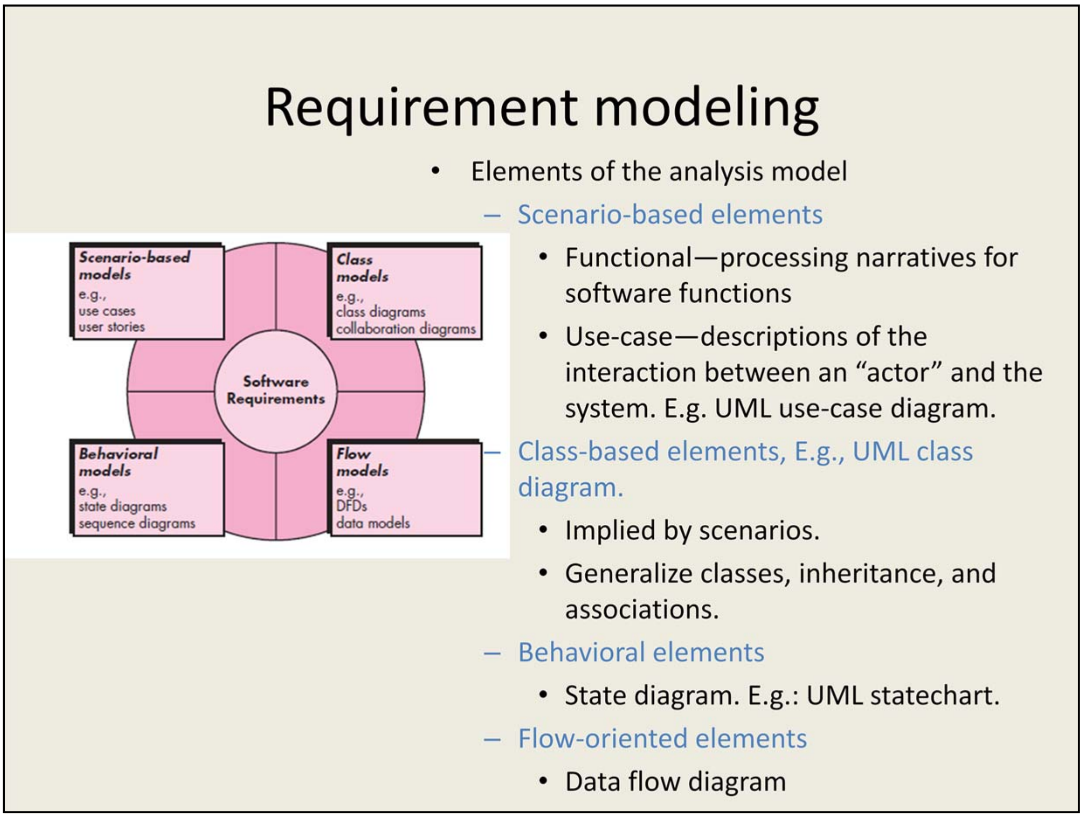


Requirements Modeling:
Flow, behavior, and pattern
(I)



Requirement modeling

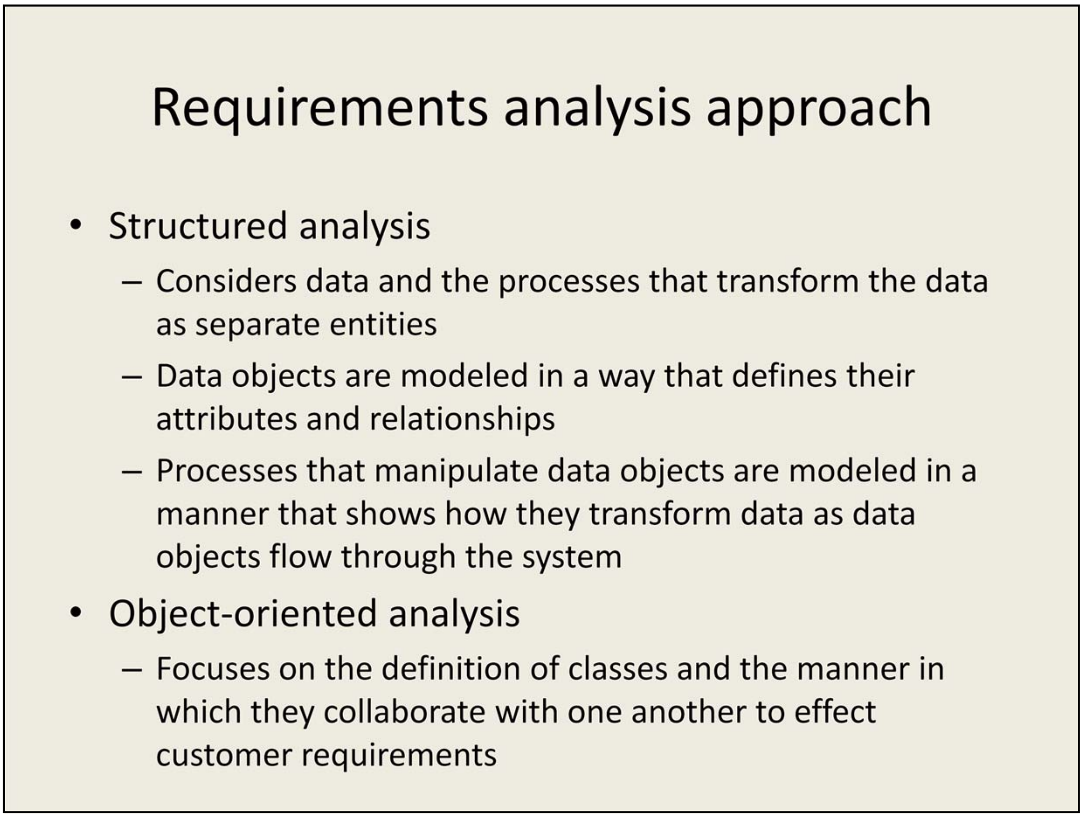
- Elements of the analysis model
  - Scenario-based elements
    - Functional—processing narratives for software functions
    - Use-case—descriptions of the interaction between an "actor" and the system. E.g. UML use-case diagram.
  - Class-based elements, E.g., UML class diagram.
    - Implied by scenarios.
    - Generalize classes, inheritance, and associations.
  - Behavioral elements
    - State diagram. E.g.: UML statechart.
  - Flow-oriented elements
    - Data flow diagram

Aren't those requirements modeling representations enough?"

For some types of software, the use case may be the only requirements modeling representation that is required.

For others, an object-oriented approach is chosen and class-based models may be developed.

But in other situations, complex application requirements may demand an examination of how data objects are transformed as they move through a system; how an application behaves as a consequence of external events;



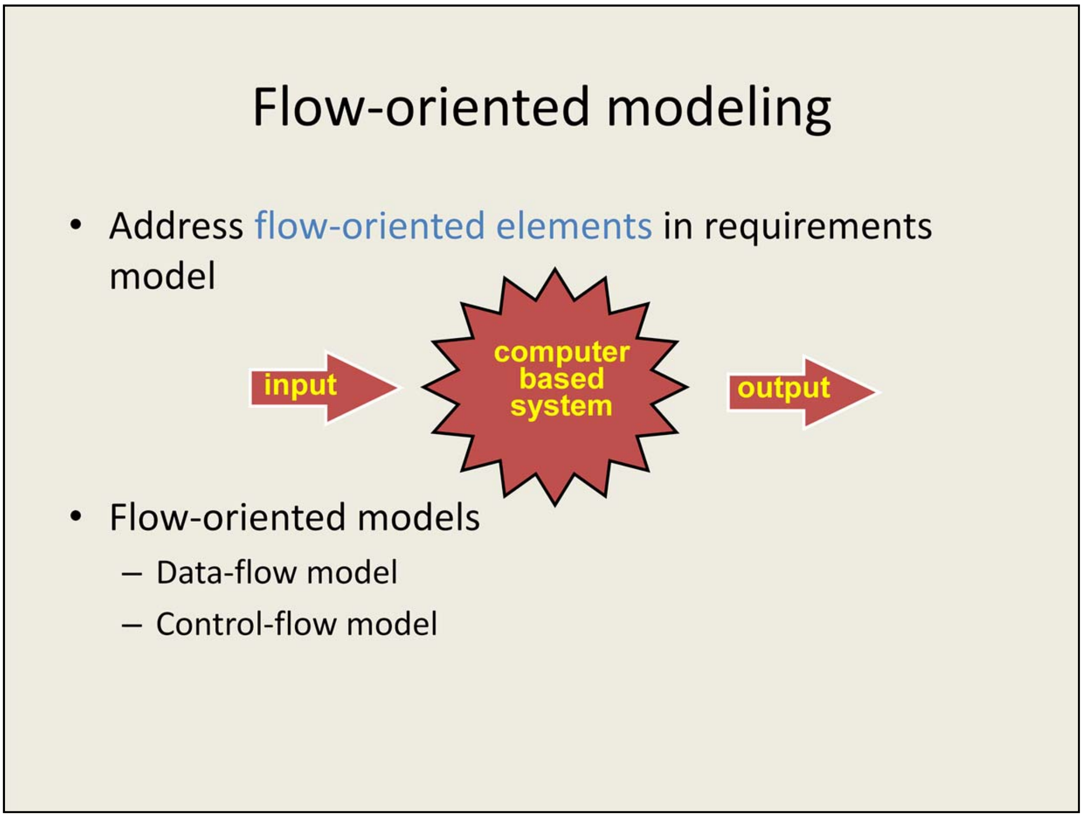
Requirements analysis approach

- Structured analysis
  - Considers data and the processes that transform the data as separate entities
  - Data objects are modeled in a way that defines their attributes and relationships
  - Processes that manipulate data objects are modeled in a manner that shows how they transform data as data objects flow through the system
- Object-oriented analysis
  - Focuses on the definition of classes and the manner in which they collaborate with one another to effect customer requirements

structured analysis, considers data and the processes that transform the data as separate entities.

object-oriented analysis, focuses on the definition of classes and the manner in which they collaborate with one another to effect customer requirements

The question is not which is best, but rather, what combination of representations will provide stakeholders with the best model of software requirements and the most effective bridge to software design.



Flow-oriented modeling

- Address flow-oriented elements in requirements model

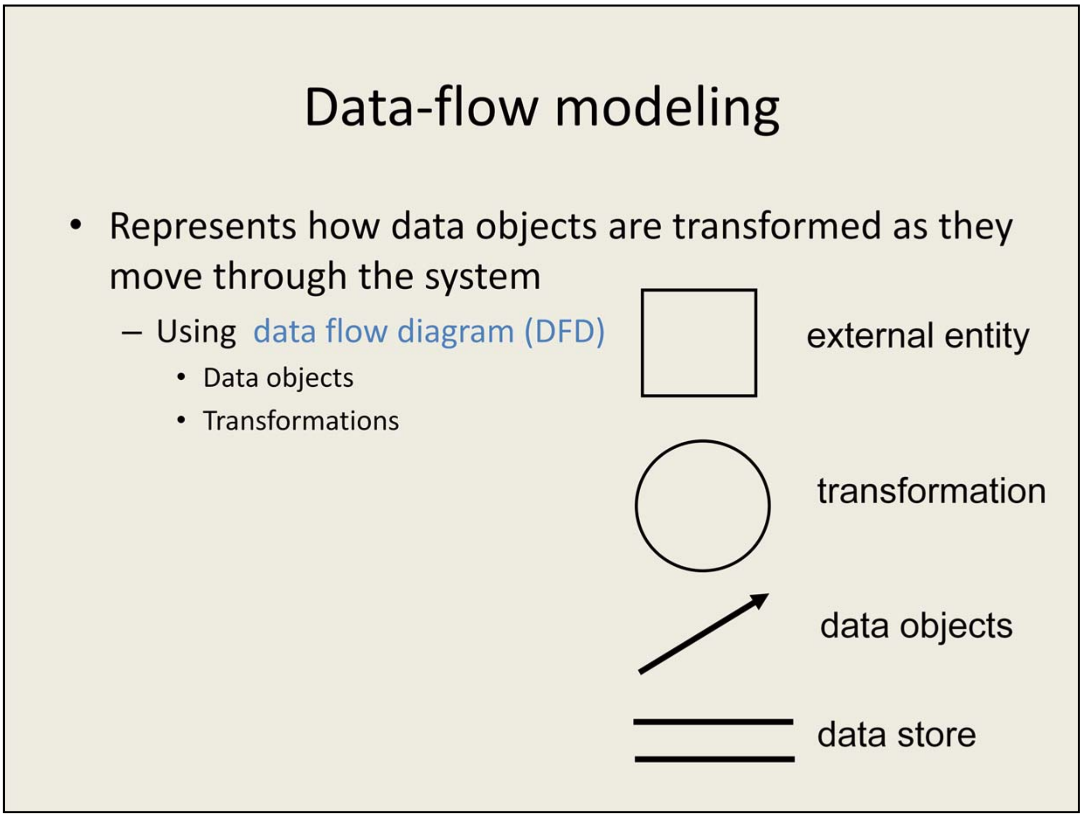
input
computer based system
output

- Flow-oriented models
  - Data-flow model
  - Control-flow model

Information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms, applies functions to transform it, and produces output in a variety of forms.

Every computer-based system is an information transformer ..

Considered by many to be an ‘old school’ approach, flow-oriented modeling continues to provide a view of the system that is unique—it should be used to supplement other analysis model elements



Data-flow modeling

- Represents how data objects are transformed as they move through the system
  - Using data flow diagram (DFD)
    - Data objects
    - Transformations

external entity

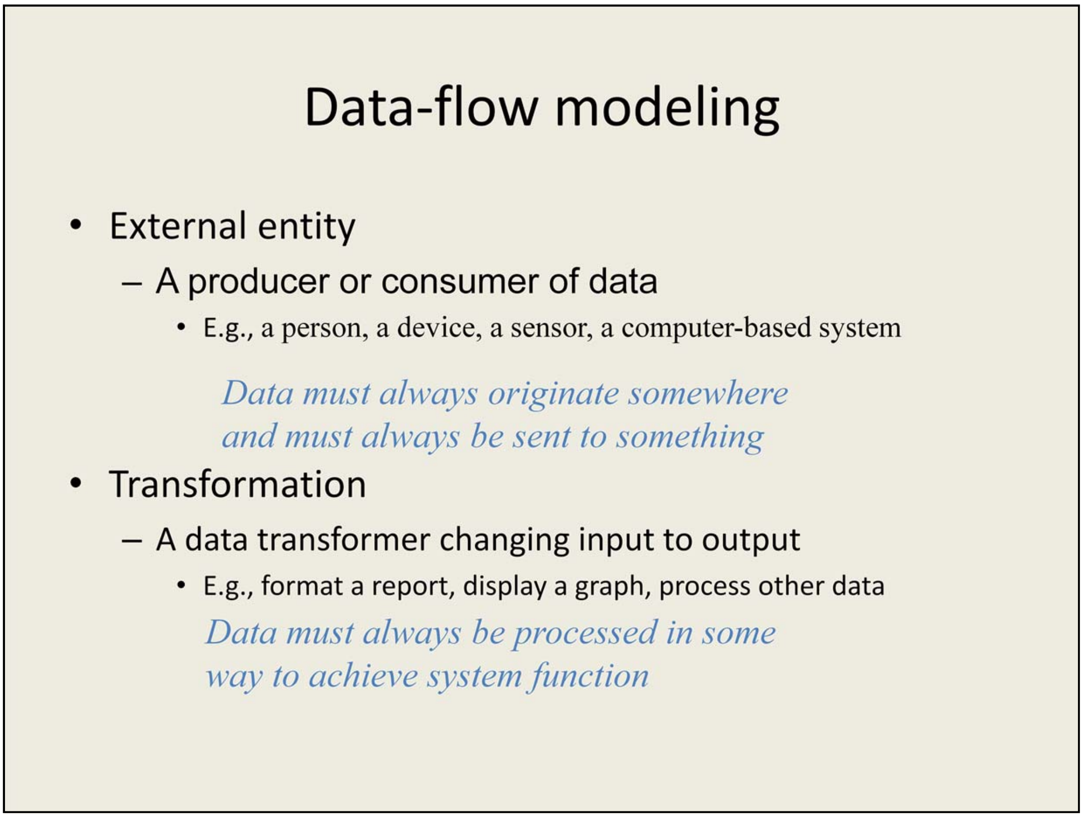
transformation

data objects

data store

Data flow modeling is a core modeling activity in *structured analysis.*

data flow diagram (DFD) and related diagrams and information are not a formal part of UML, they can be used to complement UML diagrams and provide additional insight into system requirements and flow.

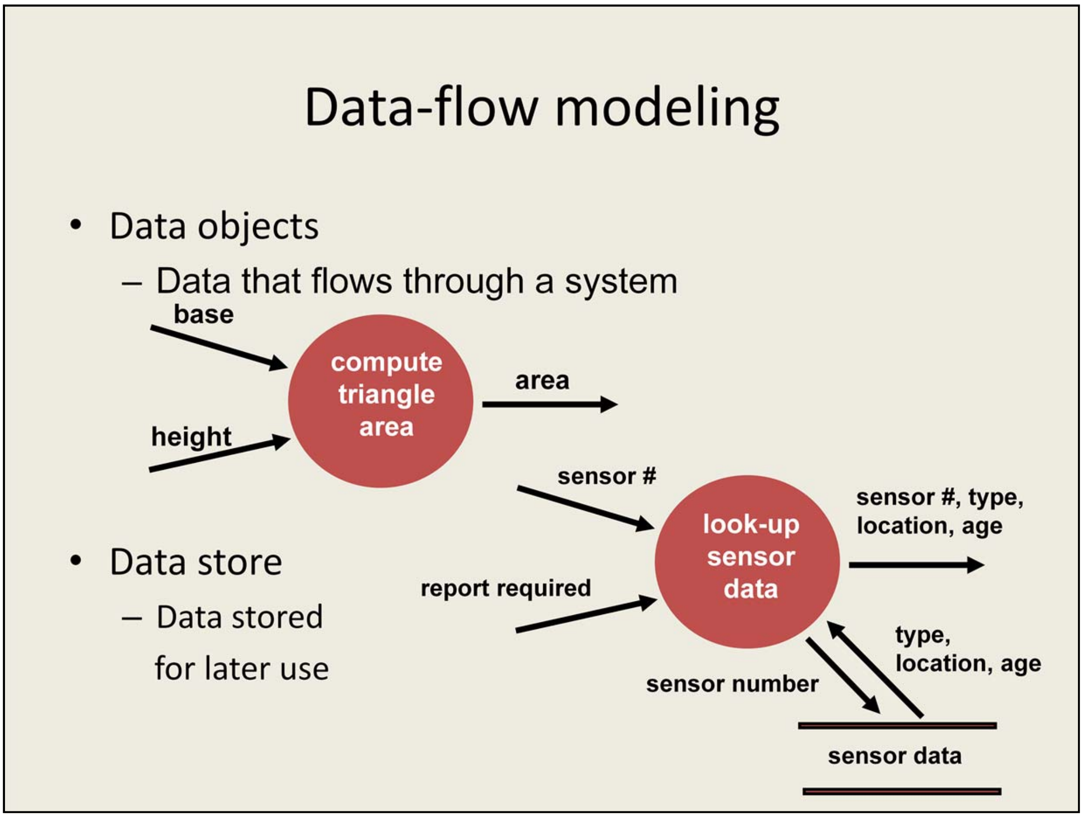


Data-flow modeling

- External entity
  - A producer or consumer of data
    - E.g., a person, a device, a sensor, a computer-based system

    Data must always originate somewhere and must always be sent to something
- Transformation
  - A data transformer changing input to output
    - E.g., format a report, display a graph, process other data

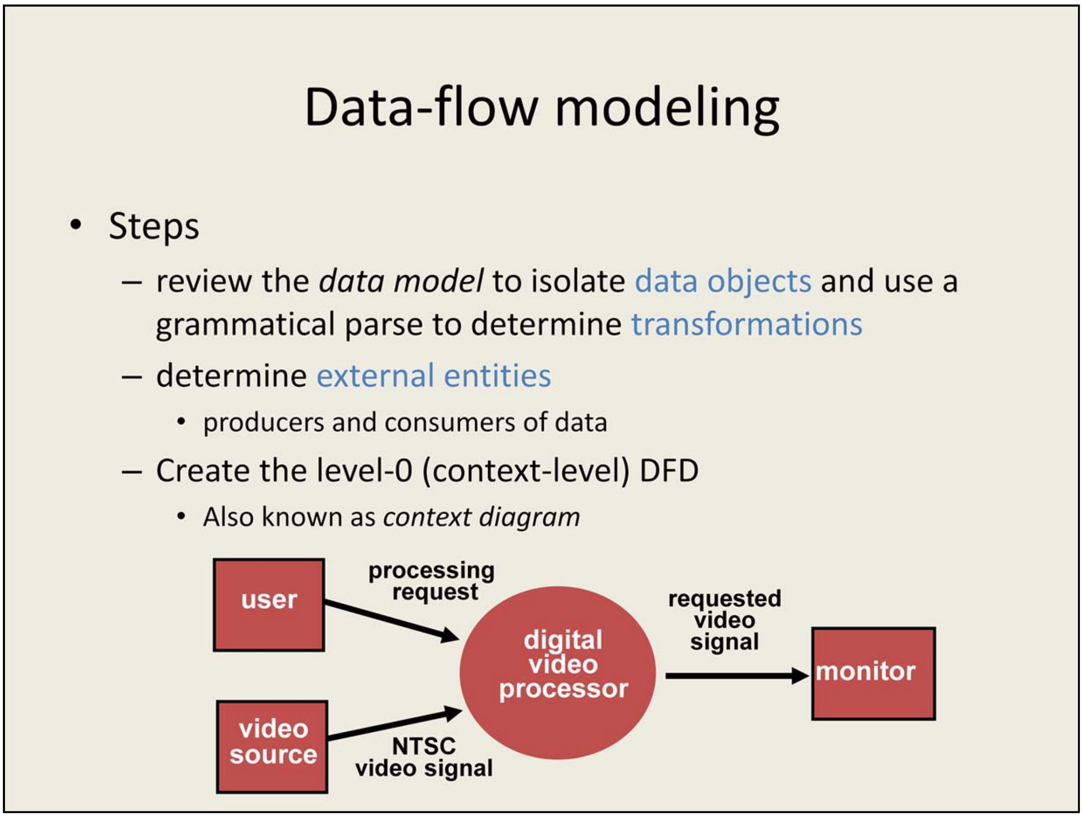
    Data must always be processed in some way to achieve system function



Data-flow modeling

- Data objects
  - Data that flows through a system

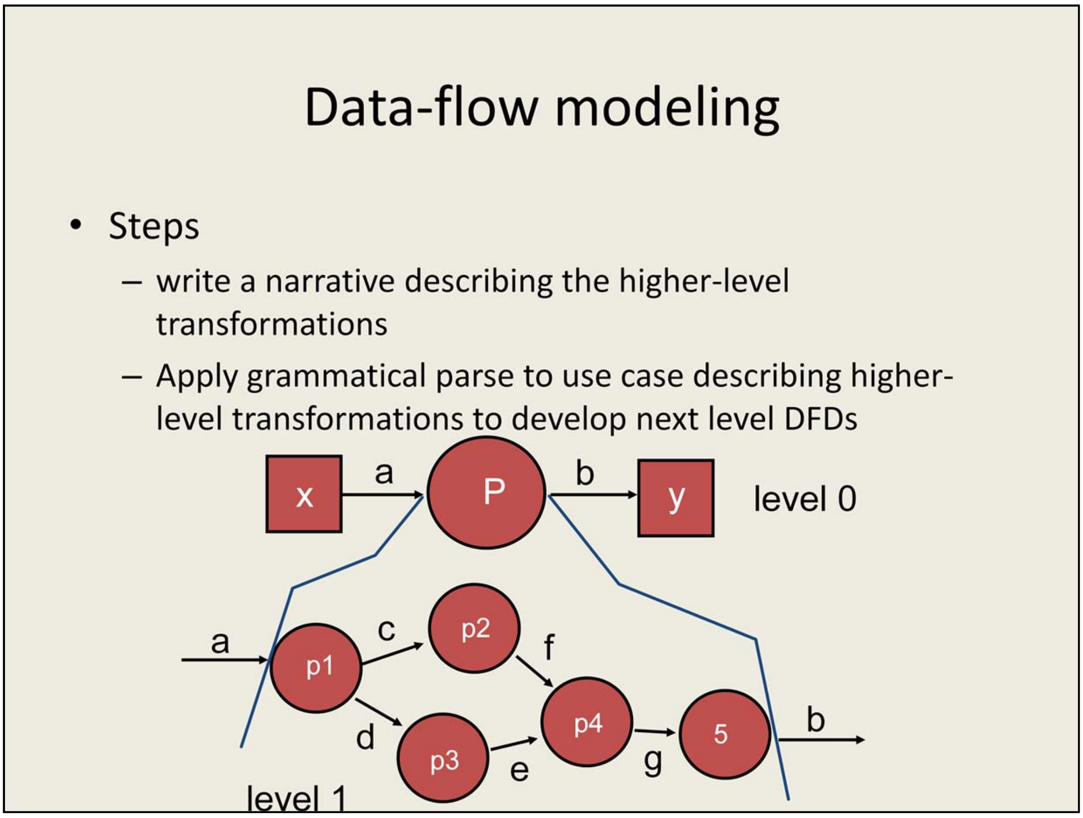
- Data store
  - Data stored for later use



Data-flow modeling

- Steps
  - review the *data model* to isolate data objects and use a grammatical parse to determine transformations
  - determine external entities
    - producers and consumers of data
  - Create the level-0 (context-level) DFD
    - Also known as *context diagram*

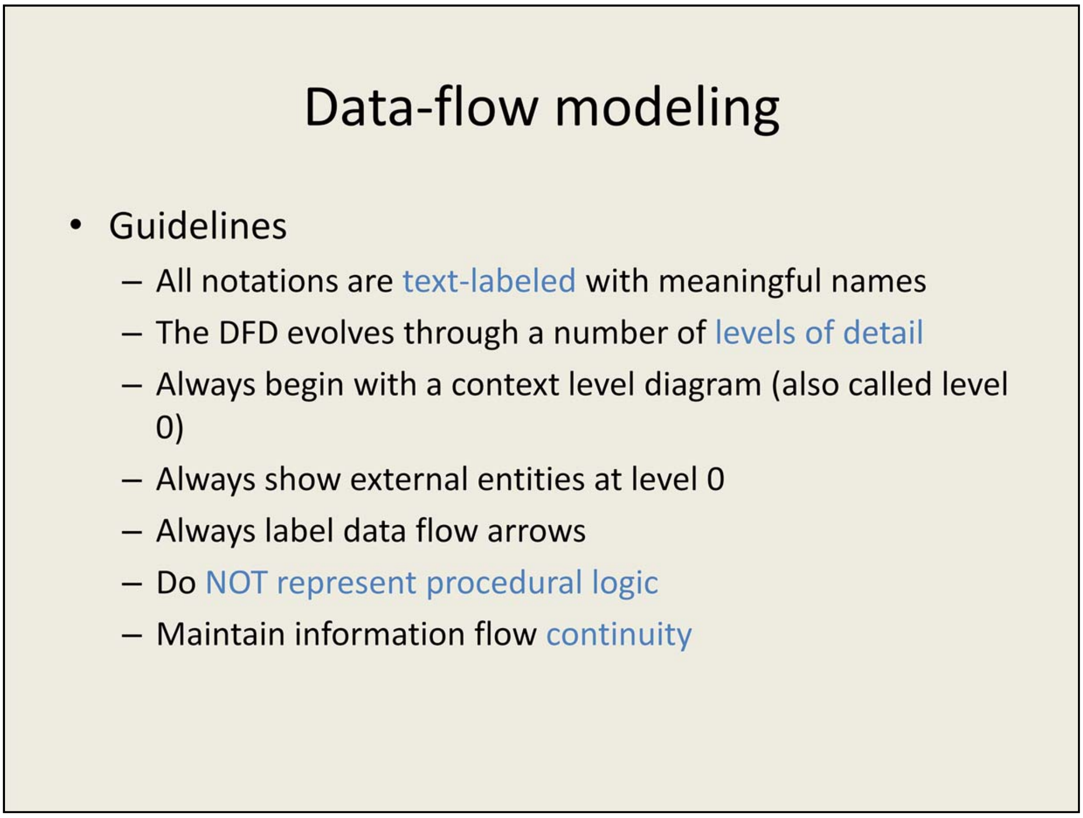
The DFD is presented in a hierarchical fashion. That is, the first data flow model (sometimes called a level 0 DFD or *context diagram*) represents the system as a whole.



Data-flow modeling

- Steps
  - write a narrative describing the higher-level transformations
  - Apply grammatical parse to use case describing higher-level transformations to develop next level DFDs

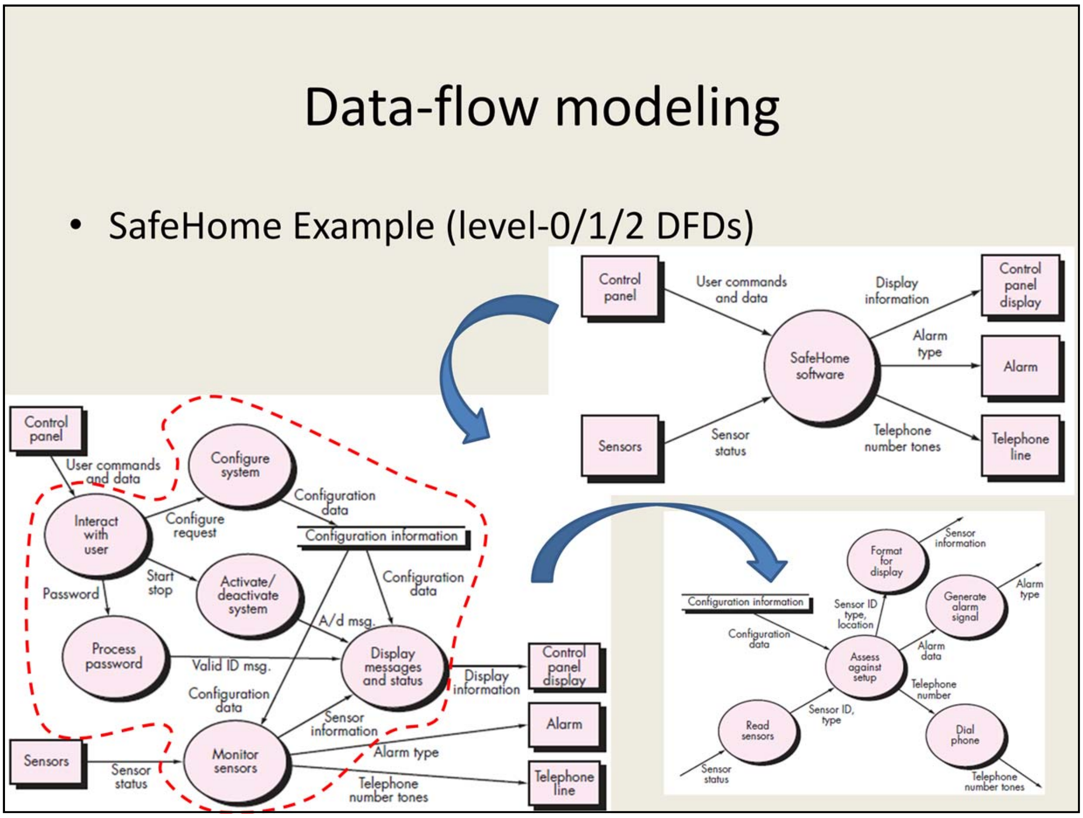
Subsequent data flow diagrams refine the context diagram, providing increasing detail with each subsequent level.



Data-flow modeling

- Guidelines
  - All notations are text-labeled with meaningful names
  - The DFD evolves through a number of levels of detail
  - Always begin with a context level diagram (also called level 0)
  - Always show external entities at level 0
  - Always label data flow arrows
  - Do NOT represent procedural logic
  - Maintain information flow continuity

Continuity: the data objects that flow into the system or into any transformation at one level must be the same data objects (or their constituent parts) that flow into the transformation at a more refined level.



Data-flow modeling

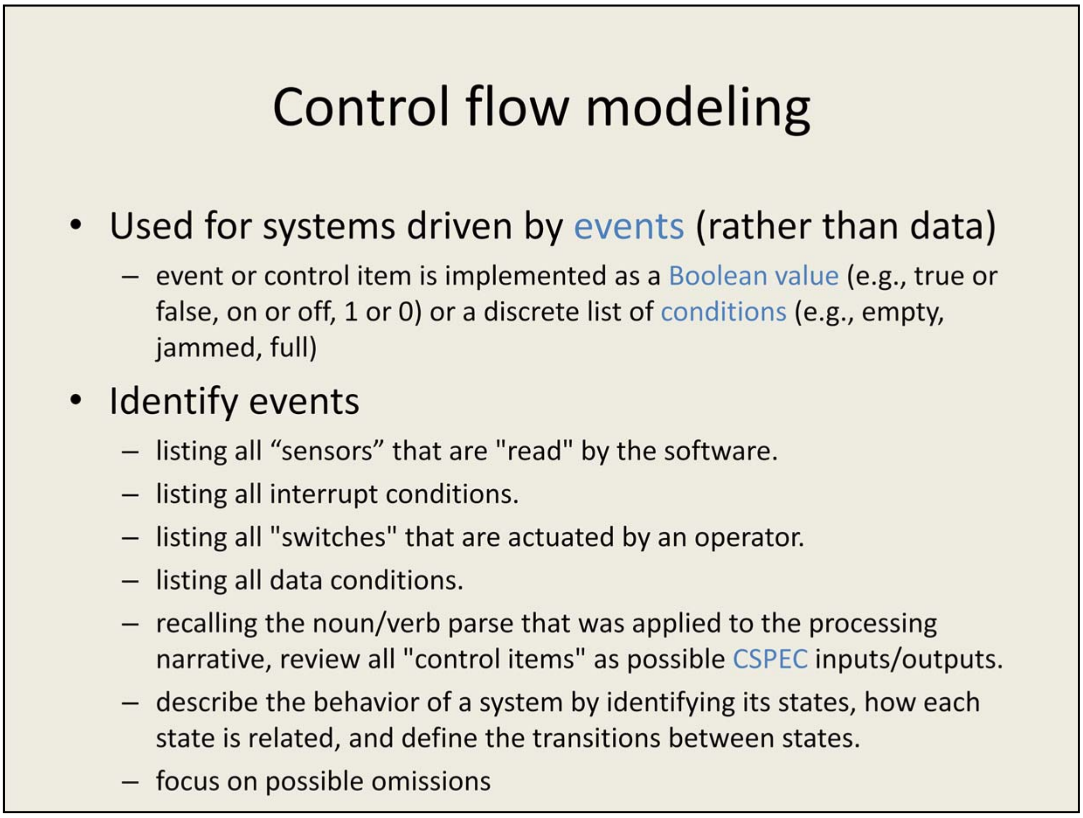
- SafeHome Example (level-0/1/2 DFDs)

Each bubble is refined until it does just one thing

The expansion ratio decreases as the number of levels increase

Most systems require between 3 and 7 levels for an adequate flow model

A single data flow item may be expanded as levels increase.



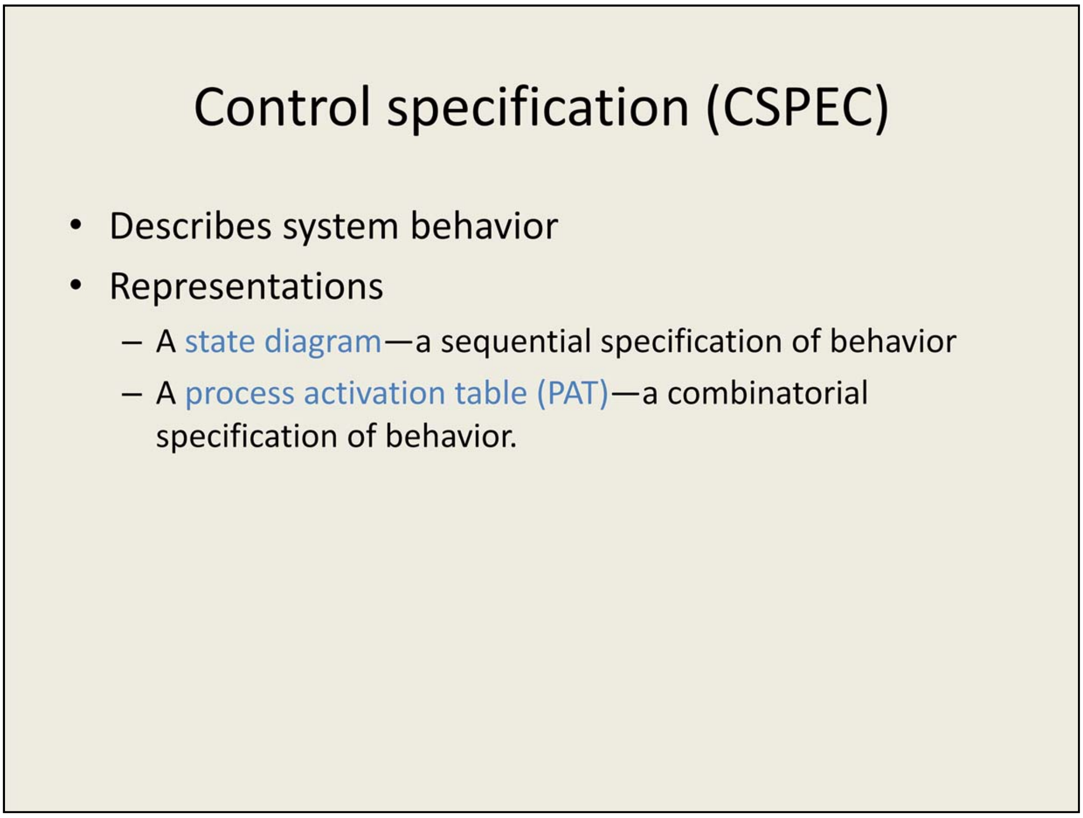
Control flow modeling

- Used for systems driven by events (rather than data)
  - event or control item is implemented as a Boolean value (e.g., true or false, on or off, 1 or 0) or a discrete list of conditions (e.g., empty, jammed, full)
- Identify events
  - listing all “sensors” that are "read" by the software.
  - listing all interrupt conditions.
  - listing all "switches" that are actuated by an operator.
  - listing 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, how each state is related, and define the transitions between states.
  - focus on possible omissions

For some types of applications, the data model and the data flow diagram are all that is necessary to obtain meaningful insight into software requirements.

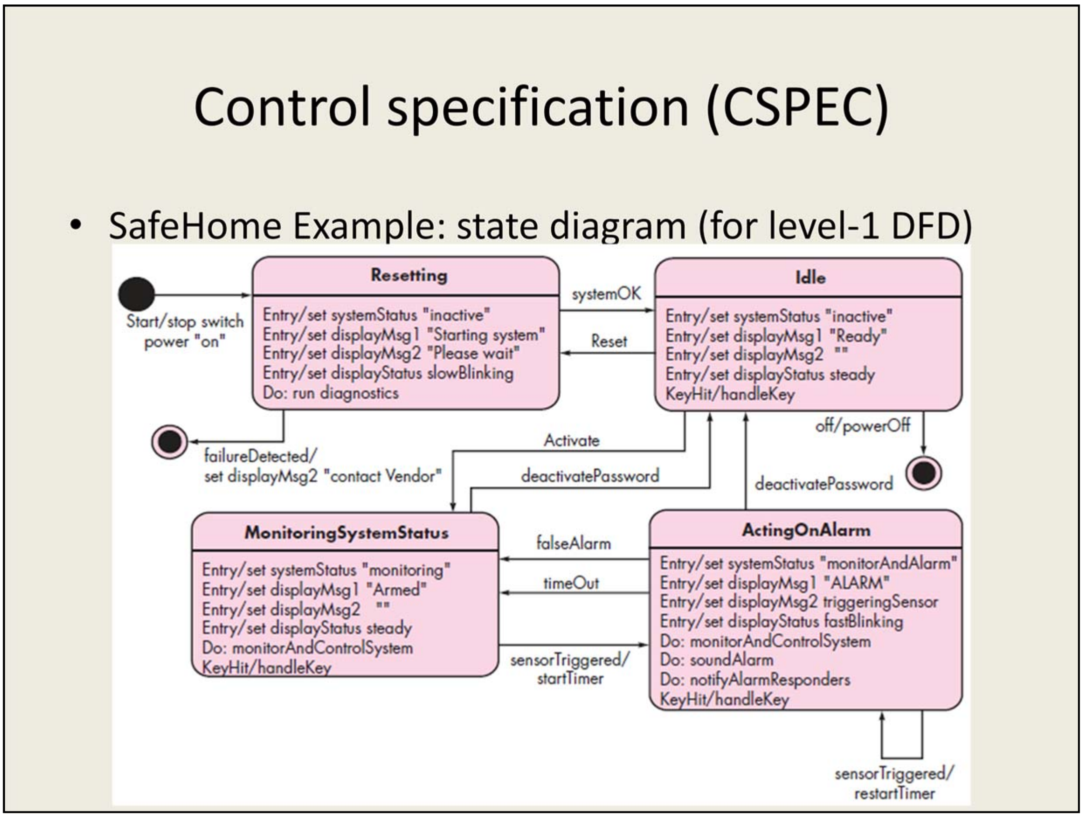
applications are “driven” by events rather than data, produce control information rather than reports or displays, and process information with heavy concern for time and performance. Such applications require also the use of *control flow modeling*

Among the many events and control items that are part of *SafeHome* software are **sensor event** (i.e., a sensor has been tripped), **blink flag** (a signal to blink the display), and **start/stop switch** (a signal to turn the system on or off).



Control specification (CSPEC)

- Describes system behavior
- Representations
  - A state diagram—a sequential specification of behavior
  - A process activation table (PAT)—a combinatorial specification of behavior.



Control specification (CSPEC)

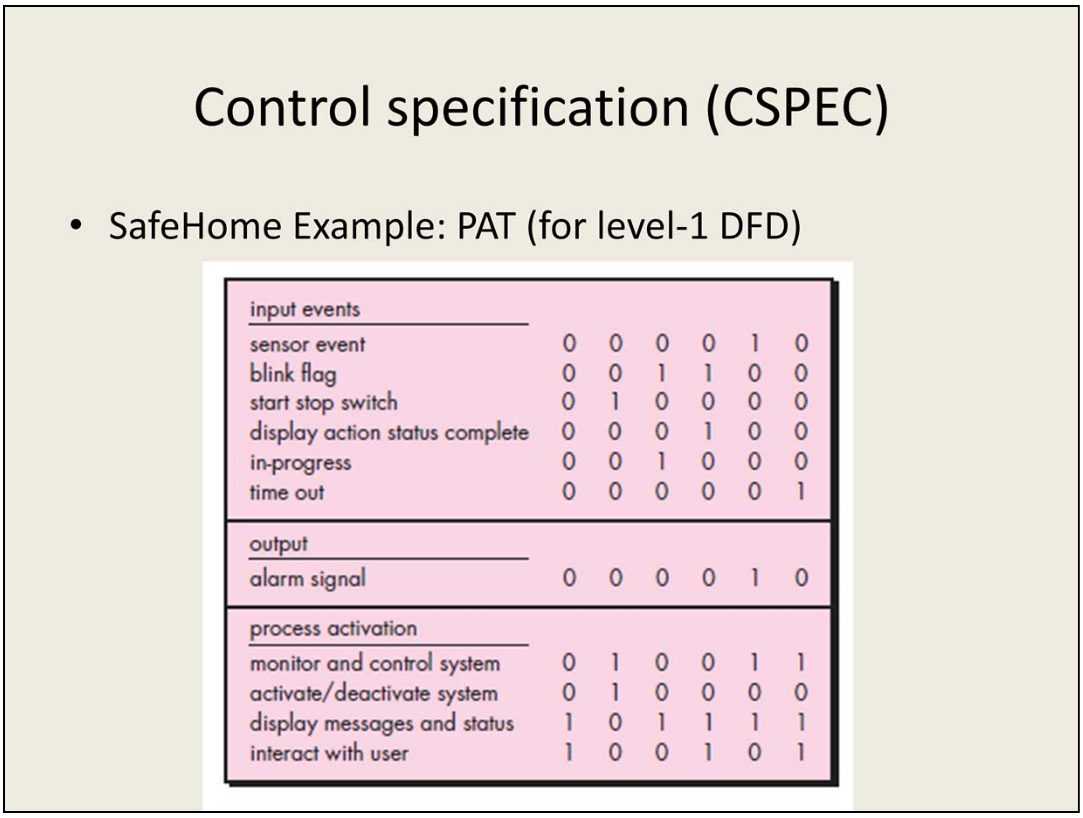
- SafeHome Example: state diagram (for level-1 DFD)

Resetting
Entry/set systemStatus "inactive"
Entry/set displayMsg1 "Starting system"
Entry/set displayMsg2 "Please wait"
Entry/set displayStatus slowBlinking
Do: run diagnostics
Start/stop switch power "on"
systemOK
Reset
Idle
Entry/set systemStatus "inactive"
Entry/set displayMsg1 "Ready"
Entry/set displayMsg2 ""
Entry/set displayStatus steady
KeyHit/handleKey
off/powerOff
failureDetected/ set displayMsg2 "contact Vendor"
Activate
deactivatePassword
deactivatePassword
MonitoringSystemStatus
Entry/set systemStatus "monitoring"
Entry/set displayMsg1 "Armed"
Entry/set displayMsg2 ""
Entry/set displayStatus steady
Do: monitorAndControlSystem
KeyHit/handleKey
falseAlarm
timeOut
sensorTriggered/ startTimer
ActingOnAlarm
Entry/set systemStatus "monitorAndAlarm"
Entry/set displayMsg1 "ALARM"
Entry/set displayMsg2 triggeringSensor
Entry/set displayStatus fastBlinking
Do: monitorAndControlSystem
Do: soundAlarm
Do: notifyAlarmResponders
KeyHit/handleKey
sensorTriggered/ restartTimer

The diagram indicates how the system responds to events as it traverses the four states defined at this level.

e.g., when the system is activated, a transition to the **Monitoring-SystemStatus** state occurs, display messages are changed as shown, and the process *monitorAndControlSystem* is invoked.

We will study more details of state diagram in the next lecture (for behavior modeling)



Control specification (CSPEC)

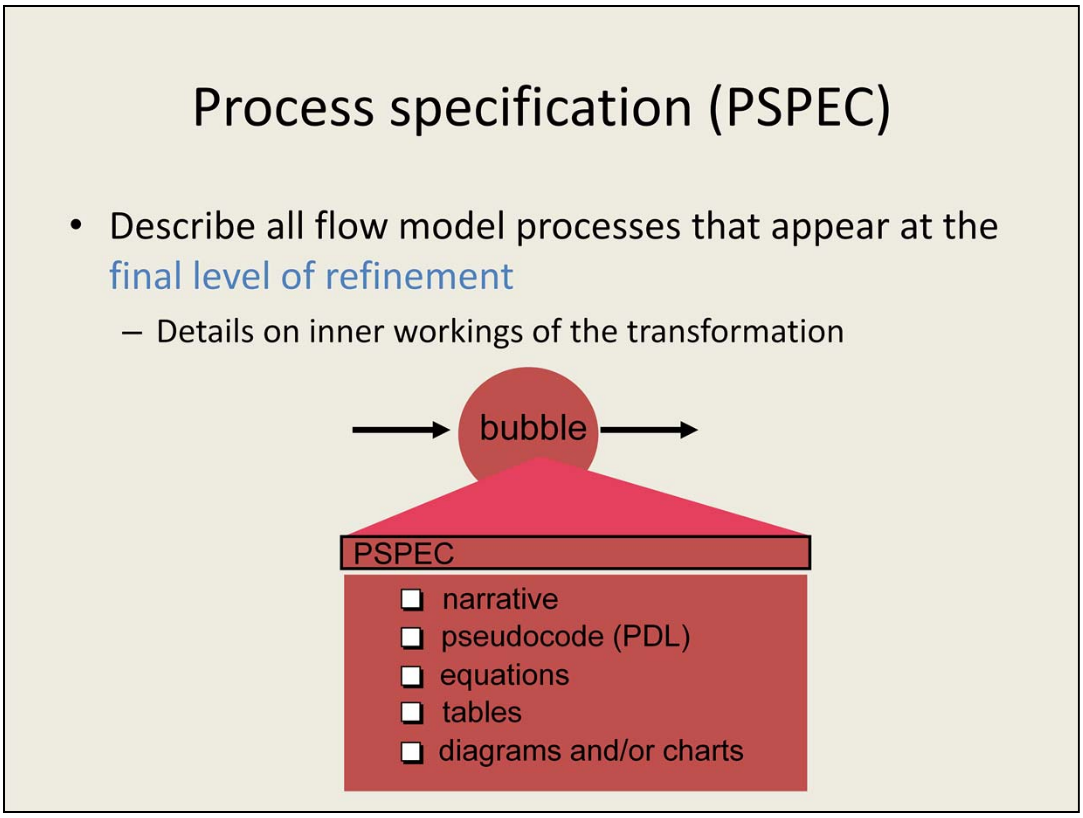
- SafeHome Example: PAT (for level-1 DFD)

input events						
sensor event	0	0	0	0	1	0
blink flag	0	0	1	1	0	0
start stop switch	0	1	0	0	0	0
display action status complete	0	0	0	1	0	0
in-progress	0	0	1	0	0	0
time out	0	0	0	0	0	1
output						
alarm signal	0	0	0	0	1	0
process activation						
monitor and control system	0	1	0	0	1	1
activate/deactivate system	0	1	0	0	0	0
display messages and status	1	0	1	1	1	1
interact with user	1	0	0	1	0	1

The PAT represents information contained in the state diagram in the context of processes (transformations), not states. That is, the table indicates which processes (bubbles) in the flow model will be invoked when an event occurs

CSPEC describes the behavior of the system, but it gives us no information about the inner working of the processes (transformations).

Process specification (PSPEC)



Process specification (PSPEC)

- Describe all flow model processes that appear at the final level of refinement
  - Details on inner workings of the transformation

bubble

PSPEC

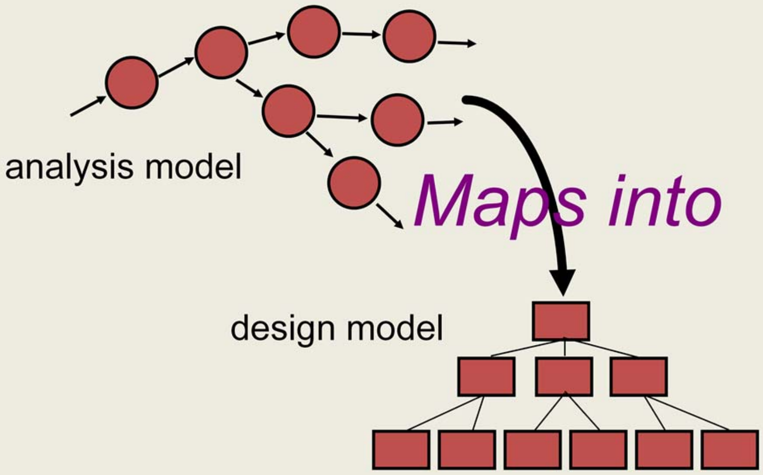
- □ narrative
- □ pseudocode (PDL)
- □ equations
- □ tables
- □ diagrams and/or charts

include narrative text, a program design language (PDL) description5 of the process algorithm, mathematical equations, tables, or UML activity diagrams.

By providing a PSPEC to accompany each bubble in the flow model, you can create a “mini-spec” that serves as a guide for design of the software component that will implement the bubble

Program design language (PDL) mixes programming language syntax with narrative text to provide procedural design detail.

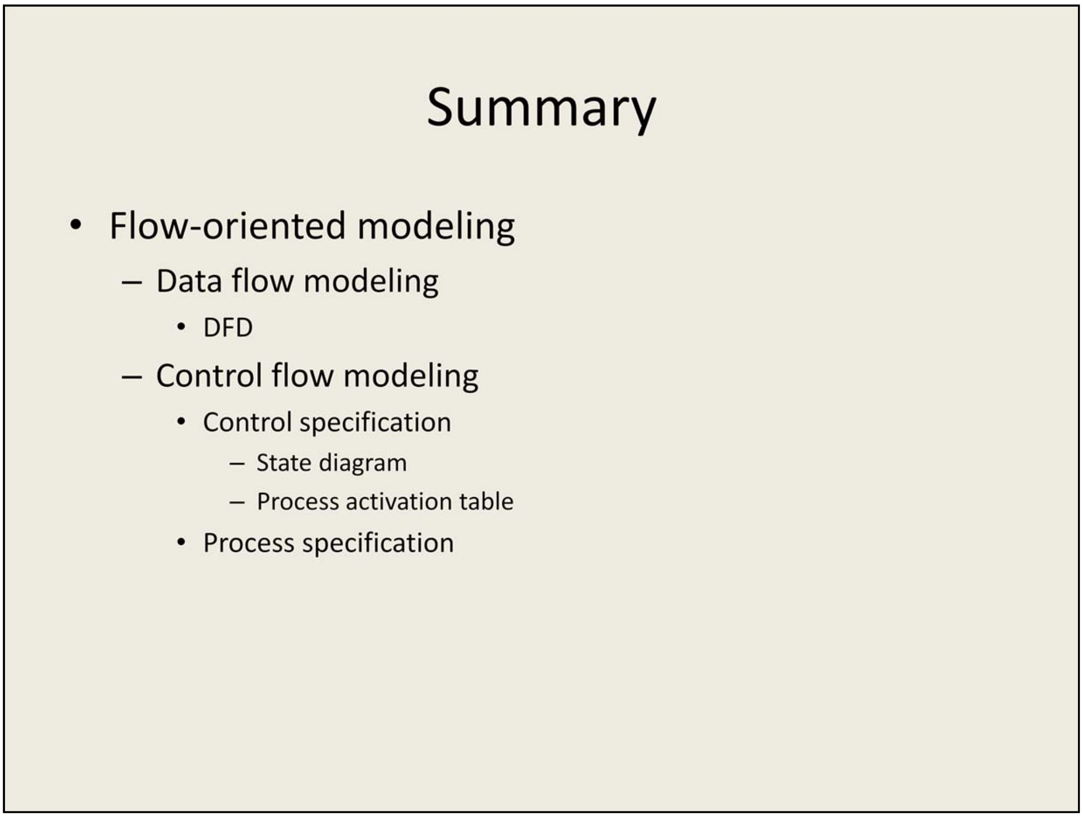
Looking ahead



analysis model

Maps into

design model



Summary

- Flow-oriented modeling
  - Data flow modeling
    - DFD
  - Control flow modeling
    - Control specification
      - State diagram
      - Process activation table
    - Process specification