Chapter 7

- **Requirements Modeling: Flow, Behavior, Patterns, and WebApps**

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 7/e
by Roger S. Pressman

Requirements Modeling Strategies
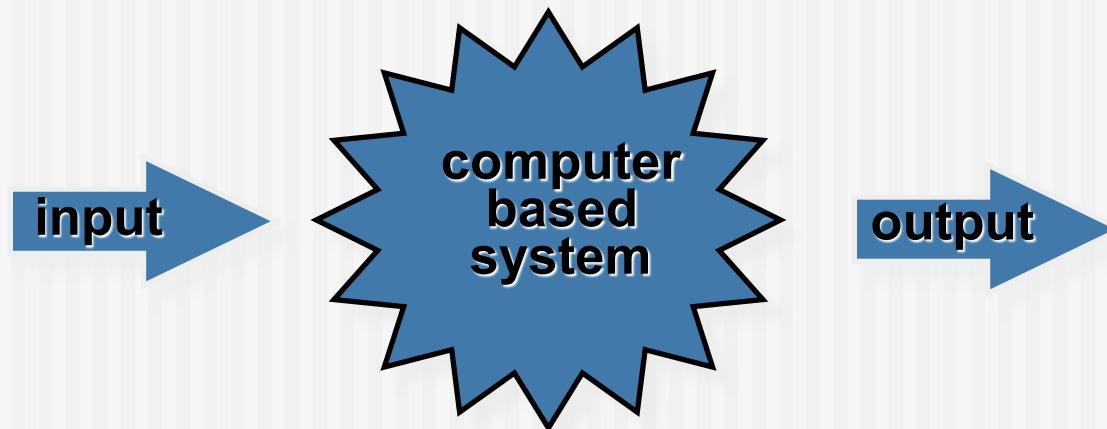
- One view of requirements modeling, called *structured analysis,* considers data and the processes that transform the data as separate entities.
    - Data objects are modeled in a way that defines their attributes and relationships.
    - Processes that manipulate data objects are modeled in a manner that shows how they transform data as data objects flow through the system.
- A second approach to analysis modeled, called *object-oriented analysis,* focuses on
    - the definition of classes and
    - the manner in which they collaborate with one another to effect customer requirements.

Flow-Oriented Modeling

- Represents how data objects are transformed at they move through the system
- **data flow diagram (DFD)** is the diagrammatic form that is used
- Considered by many to be an "old school" approach, but continues to provide a view of the system that is unique —it should be used to supplement other analysis model elements

The Flow Model
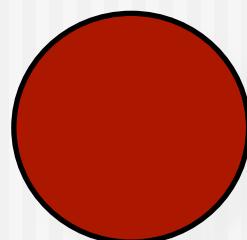
Every computer-based system is an information transform ....

input → computer based system → output

4
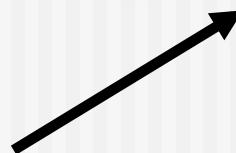
Flow Modeling Notation

external entity

process

data flow

data store

External Entity

A producer or consumer of data

Examples: a person, a device, a sensor

Another example: computer-based system

Data must always originate somewhere and must always be sent to something

Process

**A data transformer (changes input to output)**
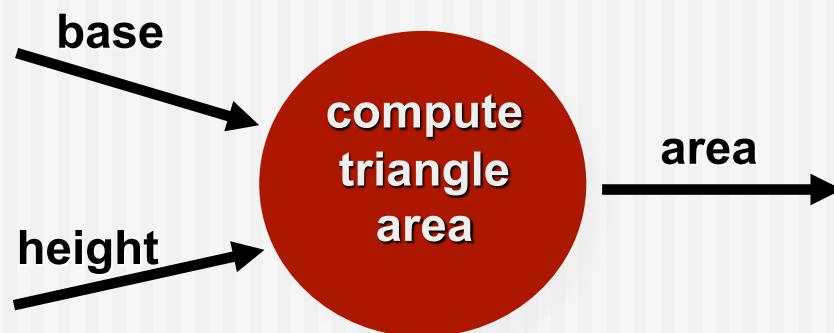
Examples: *compute taxes, determine area, format report, display graph*

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

Data Flow
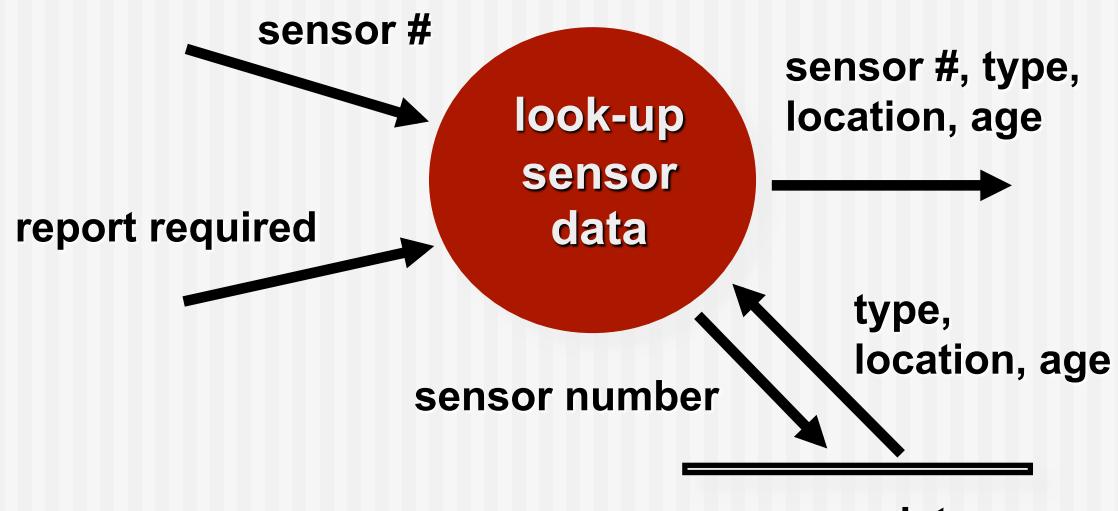
Data flows through a system, beginning as input and transformed into output.

base

compute triangle area

height

area

Data Stores

Data is often stored for later use.

sensor #

report required

look-up sensor data

sensor #, type, location, age

type, location, age

sensor number

sensor data
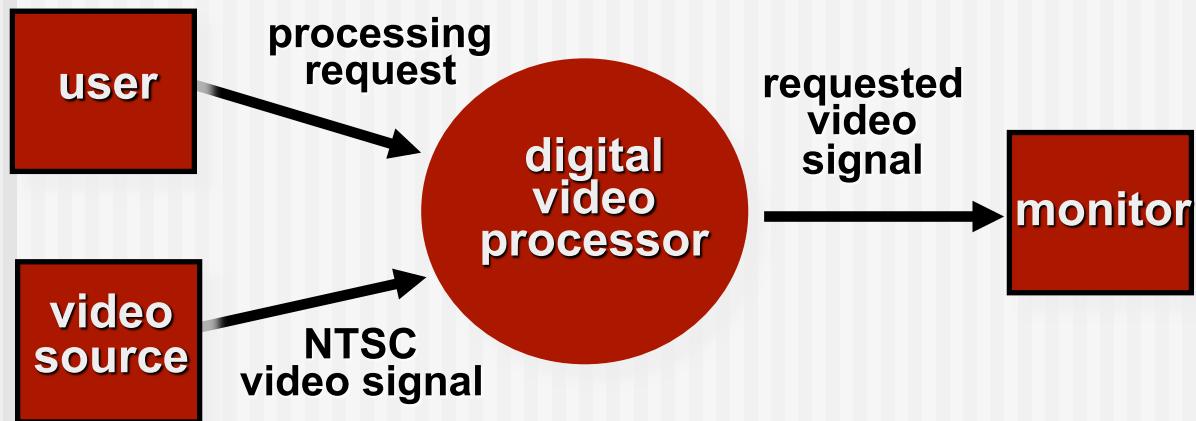
Data Flow Diagramming: Guidelines

- **all icons must be 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**

Constructing a DFD—I

- review user scenarios and/or the data model to isolate data objects and use a grammatical parse to determine "operations"

- determine external entities (producers and consumers of data)

- create a level 0 DFD
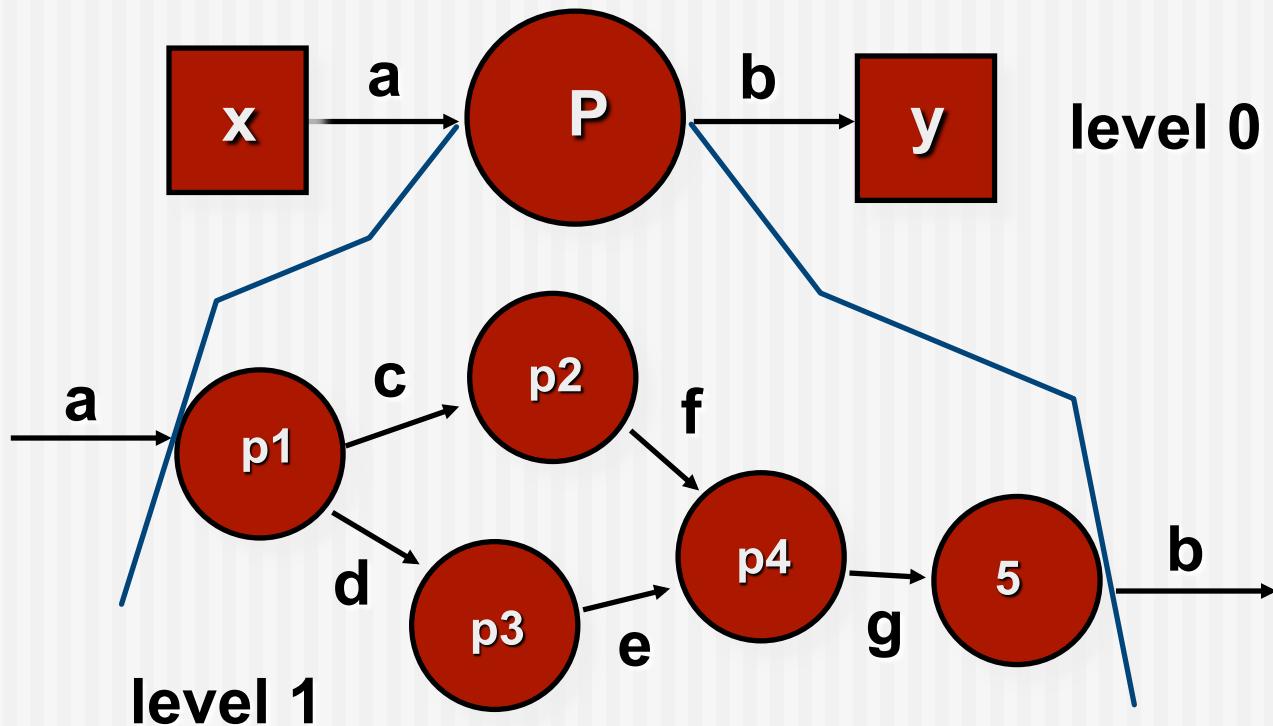
Level 0 DFD Example

Constructing a DFD—II

- write a narrative describing the transform
- parse to determine next level transforms
- "balance" the flow to maintain data flow continuity
- develop a level 1 DFD
- use a 1:5 (approx.) expansion ratio
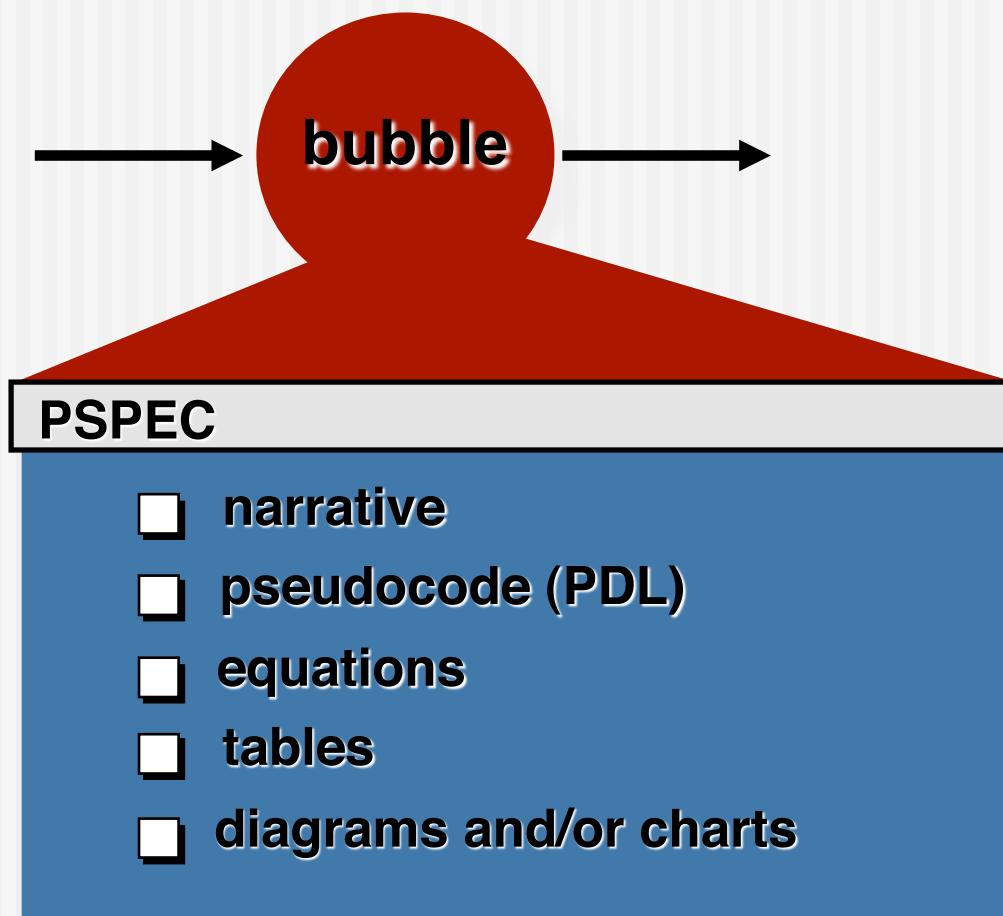
The Data Flow Hierarchy

14
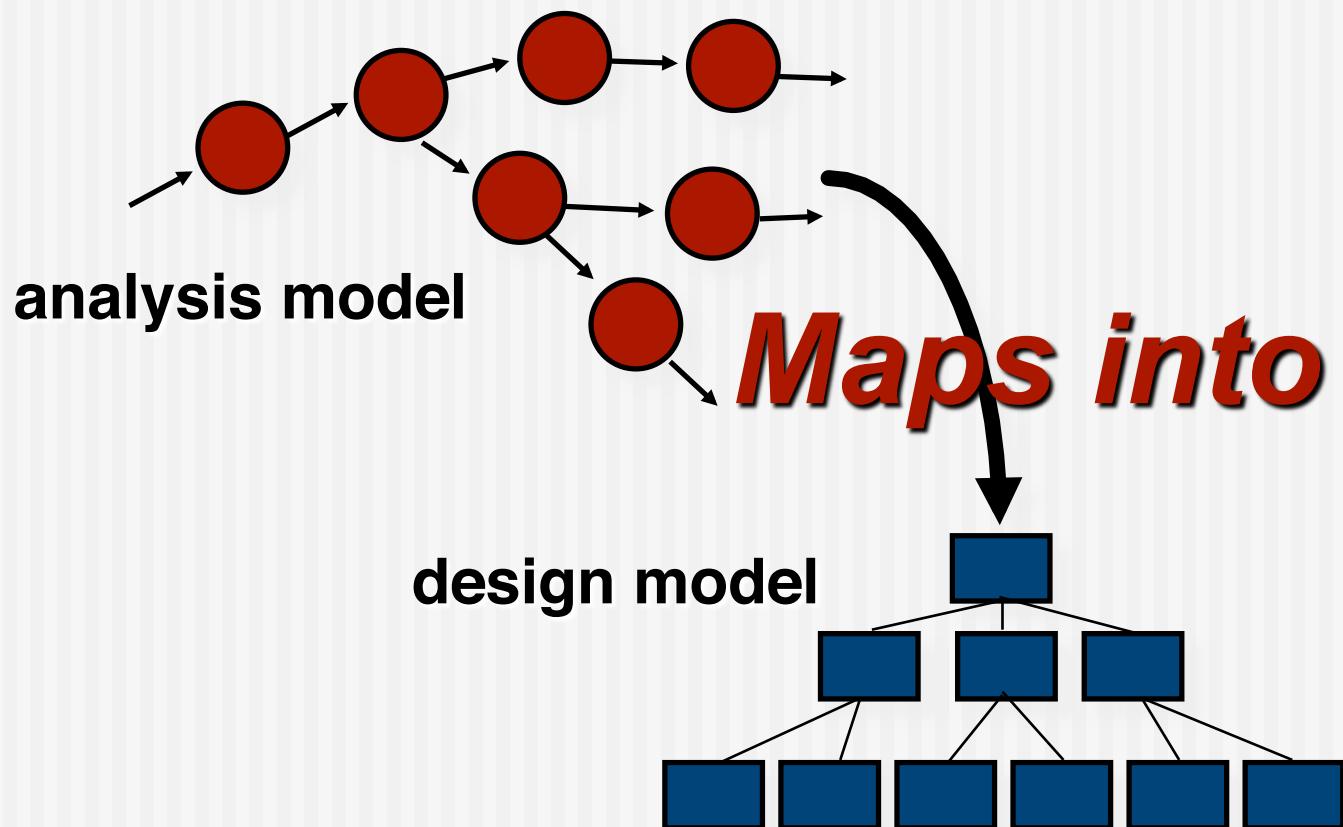
Flow Modeling Notes

- 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 (arrow) may be expanded as levels increase (data dictionary provides information)

Process Specification (PSPEC)

bubble

PSPEC

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

DFDs: A Look Ahead

analysis model

Maps into

design model

Control Flow Modeling

- Represents "events" and the processes that manage events

- An "event" is a Boolean condition that can be ascertained by:
  - 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.

Control Specification (CSPEC)

The CSPEC can be:

■ state diagram
(sequential spec)

■ state transition table

■ decision tables

combinatorial spec

■ activation tables
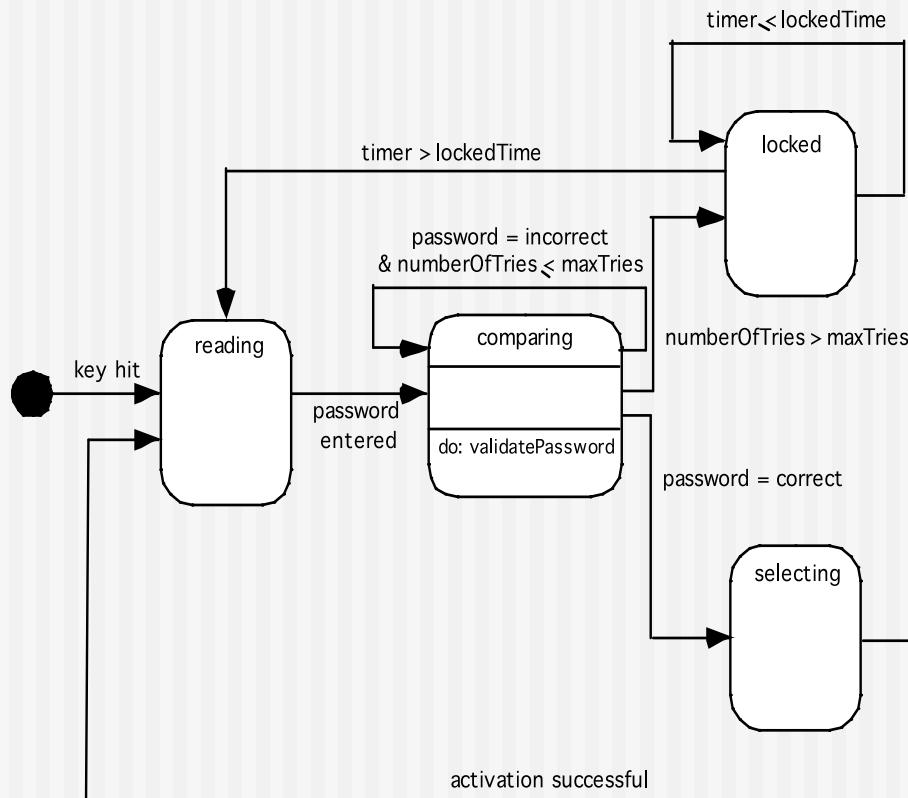
Behavioral Modeling

- The behavioral model indicates how software will respond to external events or stimuli. To create the model, the analyst must perform the following steps:

  - Evaluate all use-cases to fully understand the sequence of interaction within the system.

  - Identify events that drive the interaction sequence and understand how these events relate to specific objects.

  - Create a sequence for each use-case.

  - Build a state diagram for the system.

  - Review the behavioral model to verify accuracy and consistency.

State Representations

- In the context of behavioral modeling, two different characterizations of states must be considered:
  - the state of each class as the system performs its function and
  - the state of the system as observed from the outside as the system performs its function
- The state of a class takes on both passive and active characteristics [CHA93].
  - A *passive state* is simply the current status of all of an object's attributes.
  - The *active state* of an object indicates the current status of the object as it undergoes a continuing transformation or processing.

# State Diagram for the ControlPanel Class
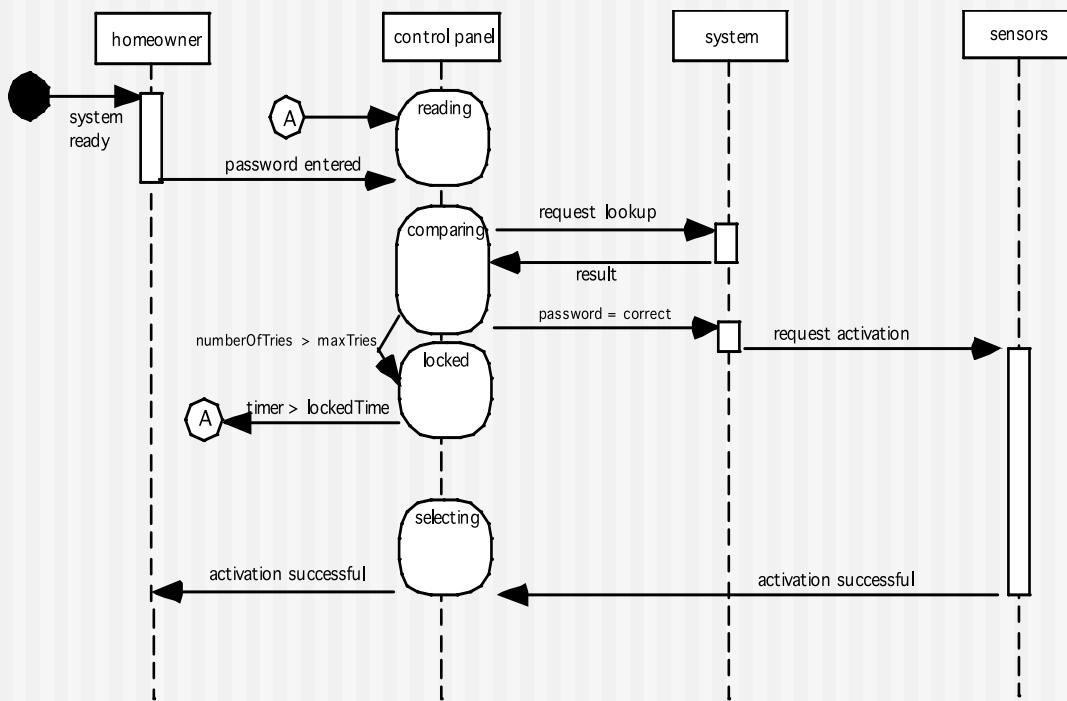
The States of a System

- **state**—a set of observable circum-stances that characterizes the behavior of a system at a given time
- **state transition**—the movement from one state to another
- **event**—an occurrence that causes the system to exhibit some predictable form of behavior
- **action**—process that occurs as a consequence of making a transition

Behavioral Modeling

- make a list of the different states of a system (How does the system behave?)
- indicate how the system makes a transition from one state to another (How does the system change state?)
  - indicate event
  - indicate action
- draw a state diagram or a sequence diagram

Sequence Diagram

Writing the Software Specification

Everyone knew exactly what had to be done until someone wrote it down!

Patterns for Requirements Modeling

- Software patterns are a mechanism for capturing domain knowledge in a way that allows it to be reapplied when a new problem is encountered
  - domain knowledge can be applied to a new problem within the same application domain
  - the domain knowledge captured by a pattern can be applied by analogy to a completely different application domain.
- The original author of an analysis pattern does not "create" the pattern, but rather, *discovers* it as requirements engineering work is being conducted.
- Once the pattern has been discovered, it is documented

Discovering Analysis Patterns

- The most basic element in the description of a requirements model is the use case.

- A coherent set of use cases may serve as the basis for discovering one or more analysis patterns.

- A *semantic analysis pattern* (SAP) "is a pattern that describes a small set of coherent use cases that together describe a basic generic application." [Fer00]

An Example

- Consider the following preliminary use case for software required to control and monitor a real-view camera and proximity sensor for an automobile:

> **Use case:** *Monitor reverse motion*
>
> **Description:** When the vehicle is placed in *reverse* gear, the control software enables a video feed from a rear-placed video camera to the dashboard display. The control software superimposes a variety of distance and orientation lines on the dashboard display so that the vehicle operator can maintain orientation as the vehicle moves in reverse. The control software also monitors a proximity sensor to determine whether an object is inside 10 feet of the rear of the vehicle. It will automatically break the vehicle if the proximity sensor indicates an object within 3 feet of the rear
>
> of the vehicle.

An Example

- This use case implies a variety of functionality that would be refined and elaborated (into a coherent set of use cases) during requirements gathering and modeling.

- Regardless of how much elaboration is accomplished, the use case(s) suggest(s) a simple, yet widely applicable SAP—the software-based monitoring and control of sensors and actuators in a physical system.

- In this case, the "sensors" provide information about proximity and video information. The "actuator" is the breaking system of the vehicle (invoked if an object is very close to the vehicle.

- But in a more general case, a widely applicable pattern is discovered --> **Actuator-Sensor**

Actuator-Sensor Pattern—I

Pattern Name: *Actuator-Sensor*

Intent: Specify various kinds of sensors and actuators in an embedded system.

Motivation: Embedded systems usually have various kinds of sensors and actuators. These sensors and actuators are all either directly or indirectly connected to a control unit. Although many of the sensors and actuators look quite different, their behavior is similar enough to structure them into a pattern. The pattern shows how to specify the sensors and actuators for a system, including attributes and operations. The *Actuator-Sensor* pattern uses a *pull* mechanism (explicit request for information) for **PassiveSensors** and a *push* mechanism (broadcast of information) for the **ActiveSensors**.

Constraints:

Each passive sensor must have some method to read sensor input and attributes that represent the sensor value.

Each active sensor must have capabilities to broadcast update messages when its value changes.

Each active sensor should send a *life tick*, a status message issued within a specified time frame, to detect malfunctions.

Each actuator must have some method to invoke the appropriate response determined by the **ComputingComponent**.

Each sensor and actuator should have a function implemented to check its own operation state.

Each sensor and actuator should be able to test the validity of the values received or sent and set its operation state if the values are outside of the specifications.
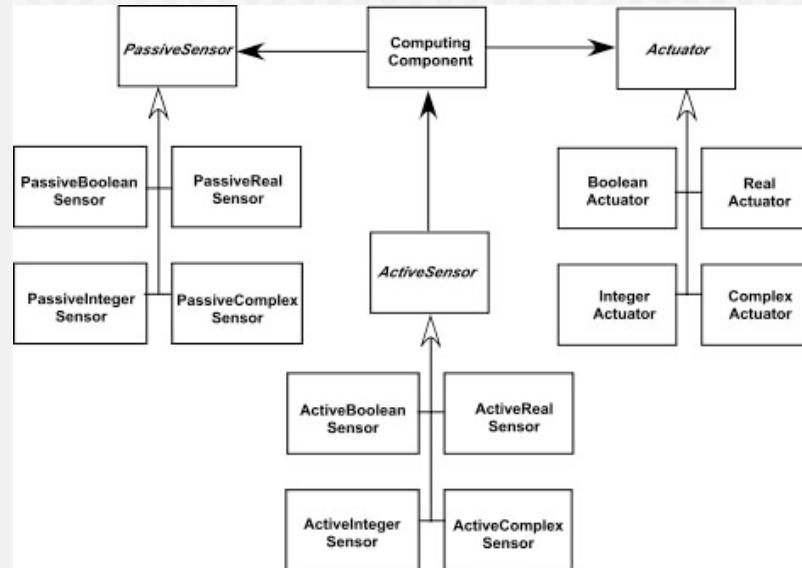
Actuator-Sensor Pattern—II

Applicability: Useful in any system in which multiple sensors and actuators are present.

Structure: A UML class diagram for the *Actuator-Sensor* Pattern is shown in Figure 7.8. **Actuator, PassiveSensor** and **ActiveSensor** are abstract classes and denoted in italics. There are four different types of sensors and actuators in this pattern. The Boolean, integer, and real classes represent the most common types of sensors and actuators. The complex classes are sensors or actuators that use values that cannot be easily represented in terms of primitive data types, such as a radar device. Nonetheless, these devices should still inherit the interface from the abstract classes since they should have basic functionalities such as querying the operation states.
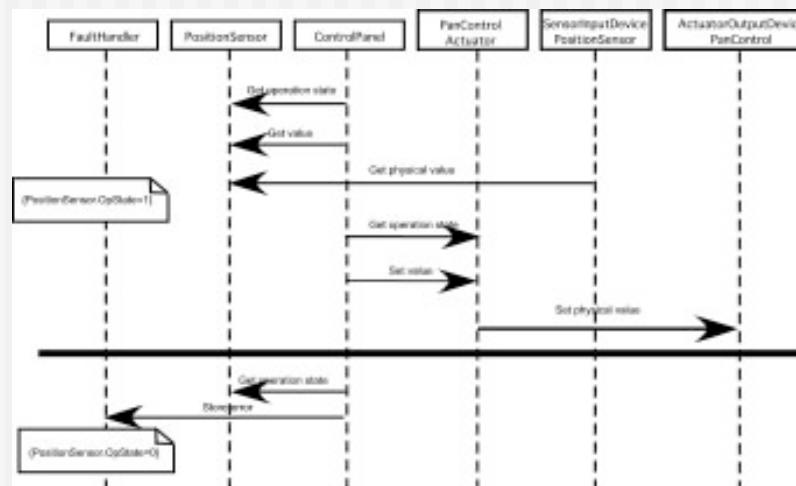
Actuator-Sensor Pattern—III

Behavior: Figure 7.9 presents a UML sequence diagram for an example of the *Actuator-Sensor* Pattern as it might be applied for the *SafeHome* function that controls the positioning (e.g., pan, zoom) of a security camera. Here, the **ControlPanel** queries a sensor (a passive position sensor) and an actuator (pan control) to check the operation state for diagnostic purposes before reading or setting a value. The messages *Set Physical Value* and *Get Physical Value* are not messages between objects. Instead, they describe the interaction between the physical devices of the system and their software counterparts. In the lower part of the diagram, below the horizontal line, the **PositionSensor** reports that the operation state is zero. The **ComputingComponent** then sends the error code for a position sensor failure to the **FaultHandler** that will decide how this error affects the system and what actions are required. it gets the data from the sensors and computes the required response for the actuators.

Actuator-Sensor Pattern—III

- See SEPA, 7/e for additional information on:
    - Participants
    - Collaborations
    - Consequences

Requirements Modeling for WebApps

Content Analysis. The full spectrum of content to be provided by the WebApp is identified, including text, graphics and images, video, and audio data. Data modeling can be used to identify and describe each of the data objects.

Interaction Analysis. The manner in which the user interacts with the WebApp is described in detail. Use-cases can be developed to provide detailed descriptions of this interaction.

Functional Analysis. The usage scenarios (use-cases) created as part of interaction analysis define the operations that will be applied to WebApp content and imply other processing functions. All operations and functions are described in detail.

Configuration Analysis. The environment and infrastructure in which the WebApp resides are described in detail.
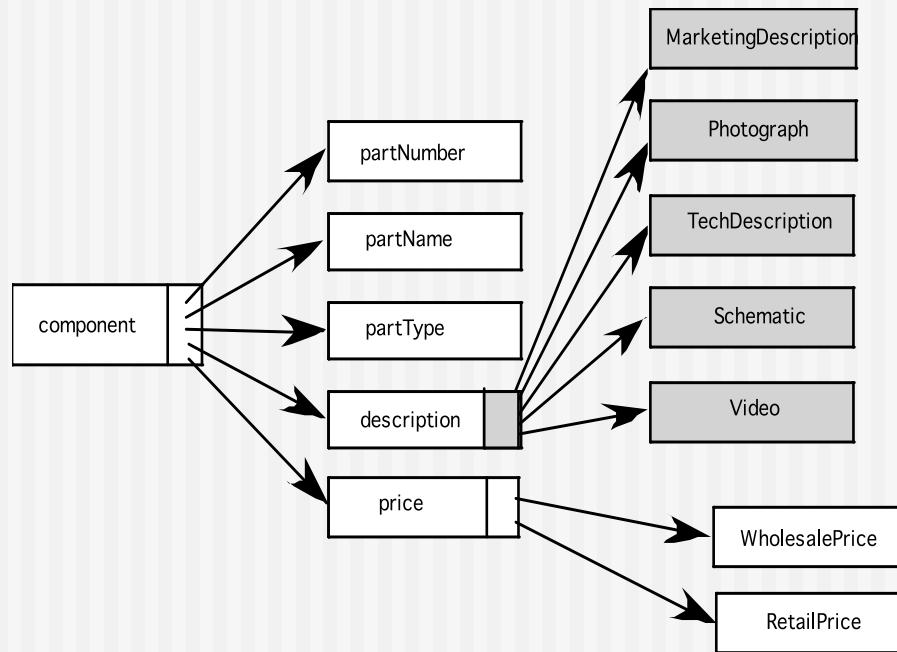
When Do We Perform Analysis?

- In some WebE situations, analysis and design merge. However, an explicit analysis activity occurs when …
    - the WebApp to be built is large and/or complex
    - the number of stakeholders is large
    - the number of Web engineers and other contributors is large
    - the goals and objectives (determined during formulation) for the WebApp will effect the business' bottom line
    - the success of the WebApp will have a strong bearing on the success of the business

The Content Model

- **Content objects** are extracted from use-cases
  - examine the scenario description for direct and indirect references to content
- **Attributes** of each content object are identified
- The **relationships** among content objects and/ or the hierarchy of content maintained by a WebApp
  - Relationships—entity-relationship diagram or UML
 - Hierarchy—data tree or UML
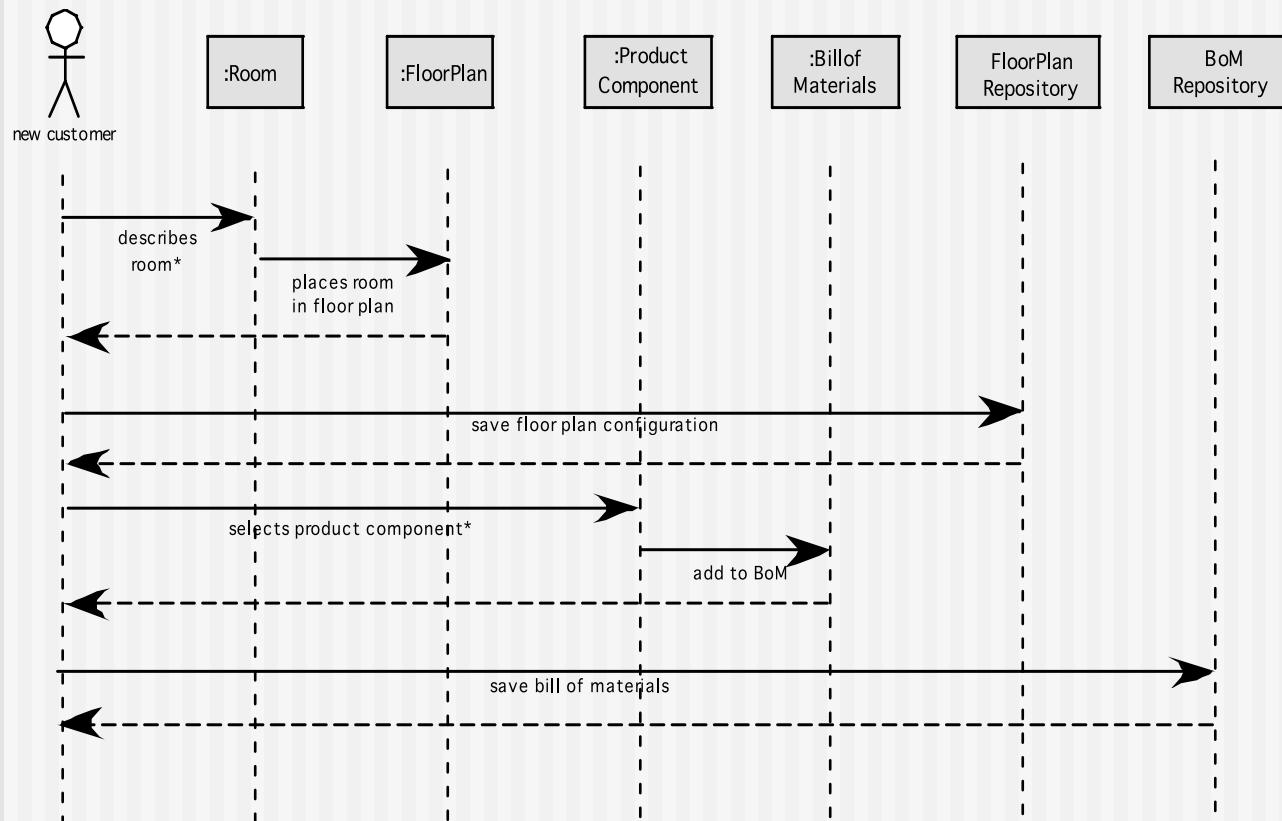
# Data Tree
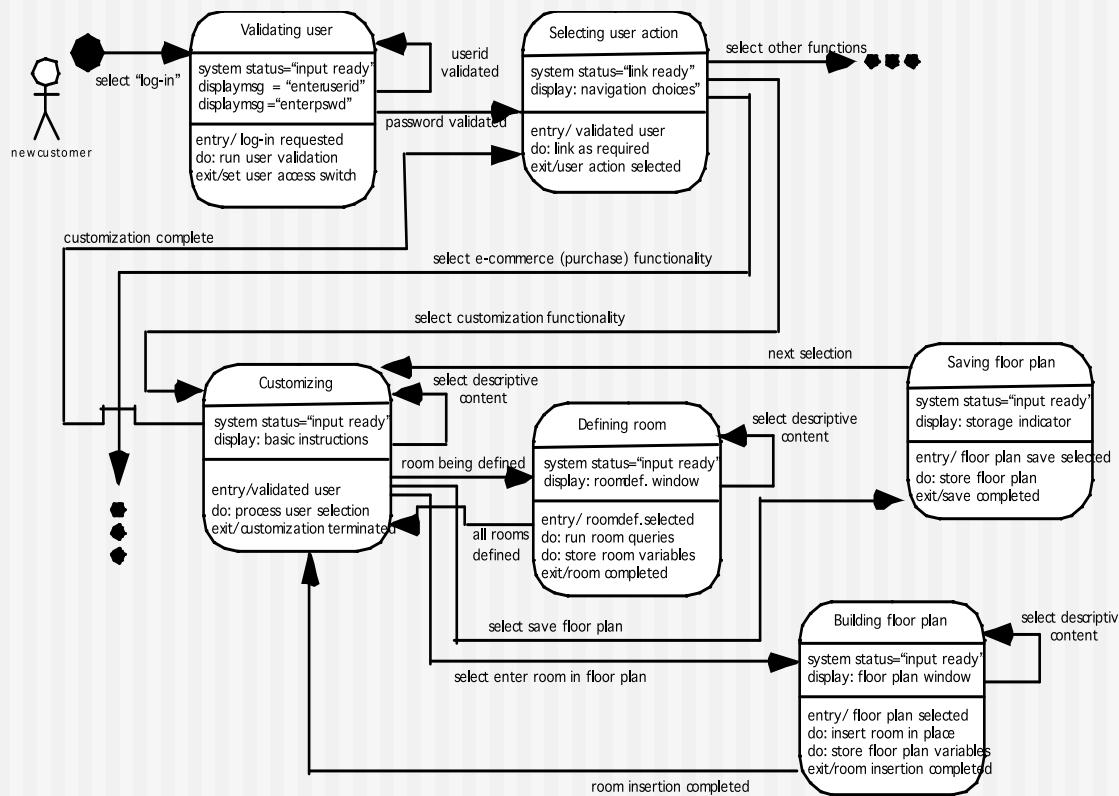
The Interaction Model

- Composed of four elements:
  - use-cases
  - sequence diagrams
  - state diagrams
  - a user interface prototype
- Each of these is an important UML notation and is described in Appendix I

Sequence Diagram

State Diagram
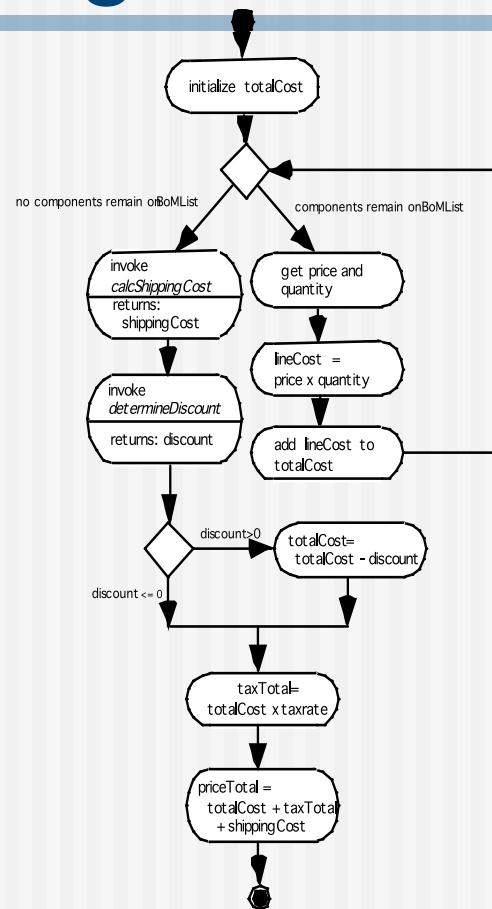
The Functional Model

- The functional model addresses two processing elements of the WebApp
  - user observable functionality that is delivered by the WebApp to end-users
  - the operations contained within analysis classes that implement behaviors associated with the class.
- An activity diagram can be used to represent processing flow

Activity Diagram

initialize totalCost

no components remain on BoMList   components remain on BoMList

invoke
calcShippingCost
returns:
shippingCost

get price and
quantity

invoke
determineDiscount

returns: discount

lineCost =
price x quantity

add lineCost to
totalCost

discount>0

totalCost=
totalCost - discount

discount <= 0

taxTotal=
totalCost x taxrate

priceTotal =
totalCost + taxTotal
+ shippingCost

The Configuration Model

- **Server-side**

    - Server hardware and operating system environment must be specified

    - Interoperability considerations on the server-side must be considered

    - Appropriate interfaces, communication protocols and related collaborative information must be specified

- **Client-side**

    - Browser configuration issues must be identified

    - Testing requirements should be defined

Navigation Modeling-I

- Should certain elements be easier to reach (require fewer navigation steps) than others? What is the priority for presentation?
- Should certain elements be emphasized to force users to navigate in their direction?
- How should navigation errors be handled?
- Should navigation to related groups of elements be given priority over navigation to a specific element.
- Should navigation be accomplished via links, via search-based access, or by some other means?
- Should certain elements be presented to users based on the context of previous navigation actions?
- Should a navigation log be maintained for users?

Navigation Modeling-II

- Should a full navigation map or menu (as opposed to a single "back" link or directed pointer) be available at every point in a user's interaction?

- Should navigation design be driven by the most commonly expected user behaviors or by the perceived importance of the defined WebApp elements?

- Can a user "store" his previous navigation through the WebApp to expedite future usage?

- For which user category should optimal navigation be designed?

- How should links external to the WebApp be handled? overlaying the existing browser window? as a new browser window? as a separate frame?