So far, we've employed control flow modeling to describe the time-varying behavior of a program.

We can also model data:
structure
flow
The techniques we are using are used for both requirements analysis and design.

We are thus collecting data for two distinct reasons.

This leads to some tension -- reflected in the book.
Goal of requirements analysis: what should we do
   Role of modeling: describe what should happen with behavior.

Goal of design
How should we do it?
   Role of modeling: suggest natural boundaries between the work of different people/groups.
Maximize cohesion.
Minimize coupling.
Maximize cohesion
Minimize coupling
The most effective visualization of a thing is something that is obvious once shown, but counter-intuitive until it's shown.

Game:

Make up some diagrams with coupling and cohesion properties. Make the whole process of breaking the thing into pieces relatively obvious. Consider multiple dimensions of the problem.
Alternative modeling techniques:
○ Give alternative, **simpler pictures** of behavior
○ Suggest **modularity boundaries** based upon attributes other than just control flow
○ Enable **cost estimation** based upon behavioral descriptions.
First step in data modeling: the **data dictionary** lists the **nouns** in the problem description, codifies their attributes and relationships.
An ATM machine reads a customer's card number from a credit card. It then asks the customer to type the last four digits of the card number, and compares them with the number read from the card to make sure it read the card correctly. If the numbers match, the ATM machine asks the customer for a PIN code and compares that with a PIN code stored on its central server for the card number. If those match, the customer is given access; otherwise access is denied.
An ATM machine reads a customer's card number from a credit card. It then asks the customer to type the last four digits of the card number, and compares them with the number read from the card to make sure it read the card correctly. If the numbers match, the ATM machine asks the customer for a PIN code and compares that with a PIN code stored on its central server for the card number. If those match, the customer is given access; otherwise access is denied.
An ATM machine reads a customer's card number from a credit card. It then asks the customer to type the last four digits of the card number, and compares them with the number read from the card to make sure it read the card correctly. If the numbers match, the ATM machine asks the customer for a PIN code and compares that with a PIN code stored on its central server for the card number. If those match, the customer is given access; otherwise access is denied.

credit card contains:
   card number

customer provides:
   credit card
   last four digits "is an attribute of" credit card number
   PIN code

central server contains:
   table of
      credit card number (key)
      PIN code

Fancy word for this set of relationships is a "data ontology": set of terms and their relationships.
A data flow diagram encodes the concept of "provides" $A \rightarrow B$ means $A$ provides data to $B$. 
An ATM machine reads a customer's card number from a credit card. It then asks the customer to type the last four digits of the card number, and compares them with the number read from the card to make sure it read the card correctly. If the numbers match, the ATM machine asks the customer for a PIN code and compares that with a PIN code stored on its central server for the card number. If those match, the customer is given access; otherwise access is denied.
Purposes of data flow diagrams

"Abstract away" time-varying behavior and describe only information exchange. Make some potentially difficult problems easier to represent. Allow easy determination of the interfaces required between entities.
One real strength of data flow analysis: coping with **asynchronous** behaviors.
- sometimes things can happen *independently* of one another.
- the control flow graph gets complicated.
- the data flow graph may remain simple.
In a musical instrument factory, a 'report card' is created for each instrument that describes what has been done for it. As different inspectors inspect the instrument, the report card is updated with various attributes of the instrument. When the report card is completed, then it is printed and included in a shipping box for the instrument.
A second kind of modeling: data structure
Jackson diagrams
Structure diagrams
Class diagrams

Next part of the problem: define the interfaces necessary for data flow
An **ATM machine** reads a customer's **card number** from a **credit card**. It then asks the **customer** to type the **last four digits** of the **card number**, and compares them with the number read from the card to make sure it read the card correctly. If the numbers match, the **ATM machine** asks the **customer** for a **PIN code** and compares that with a **PIN code** stored on its **central server** for the **card number**. If those match, the **customer** is given access; otherwise access is denied.
Describe entities as objects:

Inheritance: $A \rightarrow B$ means $A$ inherits attributes and interfaces from $B$.

Correspondence:

A 1 $\rightarrow$ 1 B  
One A interacts with one B

A * $\rightarrow$ 1 B  
There are many A's that interact with one B.

A 1 $\rightarrow$ 0 B

Etc.
Object-oriented requirements modeling (ORM)
- Not the same thing as object-oriented design!
- Describes behavior, not program structure.
- Key idea: data exchange requires interfaces.

Steps in ORM:
- Express major entities as objects
- Express data interactions as interfaces.
- Draw an overall "object diagram" that shows relationships between objects and data.
A class is a limitation upon behavior
"e.g., my first step in a walk is silly"
A subclass is a limitation upon a class
"E.g., my second step is silly"

A credit card has a number, expiration, security code.
A platinum credit card also has a high credit line
Platinum credit card is a subclass of credit card.
If A is a subclass of B, then
There are more rules for what constitutes an A than for what constitutes a B.
There are less instances of A's than of B's.

If A is a subclass of B, then an instance of B is always an instance of A.

\[
\text{Instances}(B) \subseteq \text{Instances}(A) \\
\text{Rules}(A) \subseteq \text{Rules}(B)
\]
An example of ORM subclassing
Old-style Booch modeling

Wednesday, September 23, 2009
4:03 PM

Diagram:

- Object
- Subroutine
- Data objects

A pulls data \( x \) from \( B \)

A calls \( X \) and \( Y \)

B
An ATM machine reads a customer's card number from a credit card. It then asks the customer to type the last four digits of the card number, and compares them with the number read from the card to make sure it read the card correctly. If the numbers match, the ATM machine asks the customer for a PIN code and compares that with a PIN code stored on its central server for the card number. If those match, the customer is given access; otherwise access is denied.

Refinement of a data flow diagram in the above notes.
An ATM machine reads a customer's card number from a credit card. It then asks the customer to type the last four digits of the card number, and compares them with the number read from the card to make sure it read the card correctly. If the numbers match, the ATM machine asks the customer for a PIN code and compares that with a PIN code stored on its central server for the card number. If those match, the customer is given access; otherwise access is denied.
why do data modeling?
  ○ gives an alternative view into the system.
  ○ suggests alternatives for modular decomposition of the system.
  ○ encapsulates and allows one to measure cost factors.
  ○ suggests testing methodologies(!)
  ○ helps one validate (potentially complex) activity diagrams.
key issue in design: break the program into modules that have
  ○ maximal **cohesion**: do one thing
  ○ minimal **coupling**: have as little as possible a relationship to other modules.

One purpose of data modeling: provide another way to analyze cohesion and coupling.
Identify regions of **transform** and **transaction flow**
transform flow: data is getting changed
transaction flow: "ok's" are flowing from entity to entity as computation proceeds.

Identify the center of each region;
this is the node that interacts with the most other nodes in the region, or the node "in the center of the region"

 Decompose the flow diagram into pieces based upon using each "center" as a **control module**.
The structure of a thing gives a good estimate of its cost.

Cost varies with:

- # of objects
- # of interfaces per object
- # of (distinct) interactions between objects
- complexity of each interface