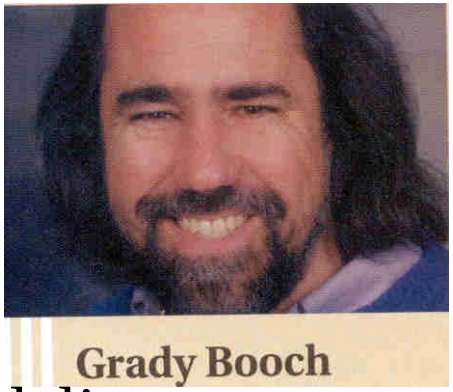
Object-Oriented Analysis (OOA) (Schach Ch 13)

- OOA: *Semi-formal* Specification Techniques:
  - With OO, *Data* and *Action* Treated as Equal Partners
  - A Class models *all* needed aspects of *one* physical entity
- Initially, Many Different "Methods" Emerged (Booch, OMT, Shlaer-Mellor, Coad-Yourdon) — all essentially doing the same thing, but in different ways.
- So, just what *is* a "Method" in this context?

What are "Methods?"

- A "Method" defines a reproducible path for obtaining reliable results. Methods vary in sophistication/formality.
  - Cooks refer to recipes, Architects use blueprints,
  - Aircrew use checklists before takeoff, landing
  - Musicians follow rules of composition.

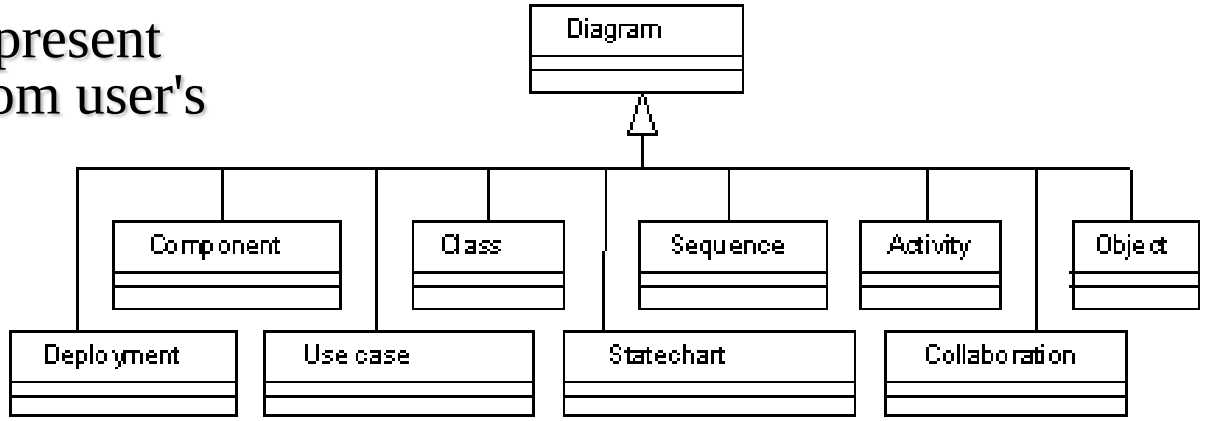

Grady Booch

- Similarly, a soft. dev. Method describes modeling software systems in a *reliable* and *reproducible* way. Facilitates comm between the various parties involved.
  - In 1994, Booch, Rumbaugh and Jacobson combined their Methods into **UML** (Unified Modeling Language):
  - UML Emerged as a *defacto* standard; uses a Common Notation for representing OOA & OOD.

Different Types of Diagrams Defined by UML

- **Use case diagrams** represent functions of system from user's viewpoint

  Provide a starting point for analysis efforts

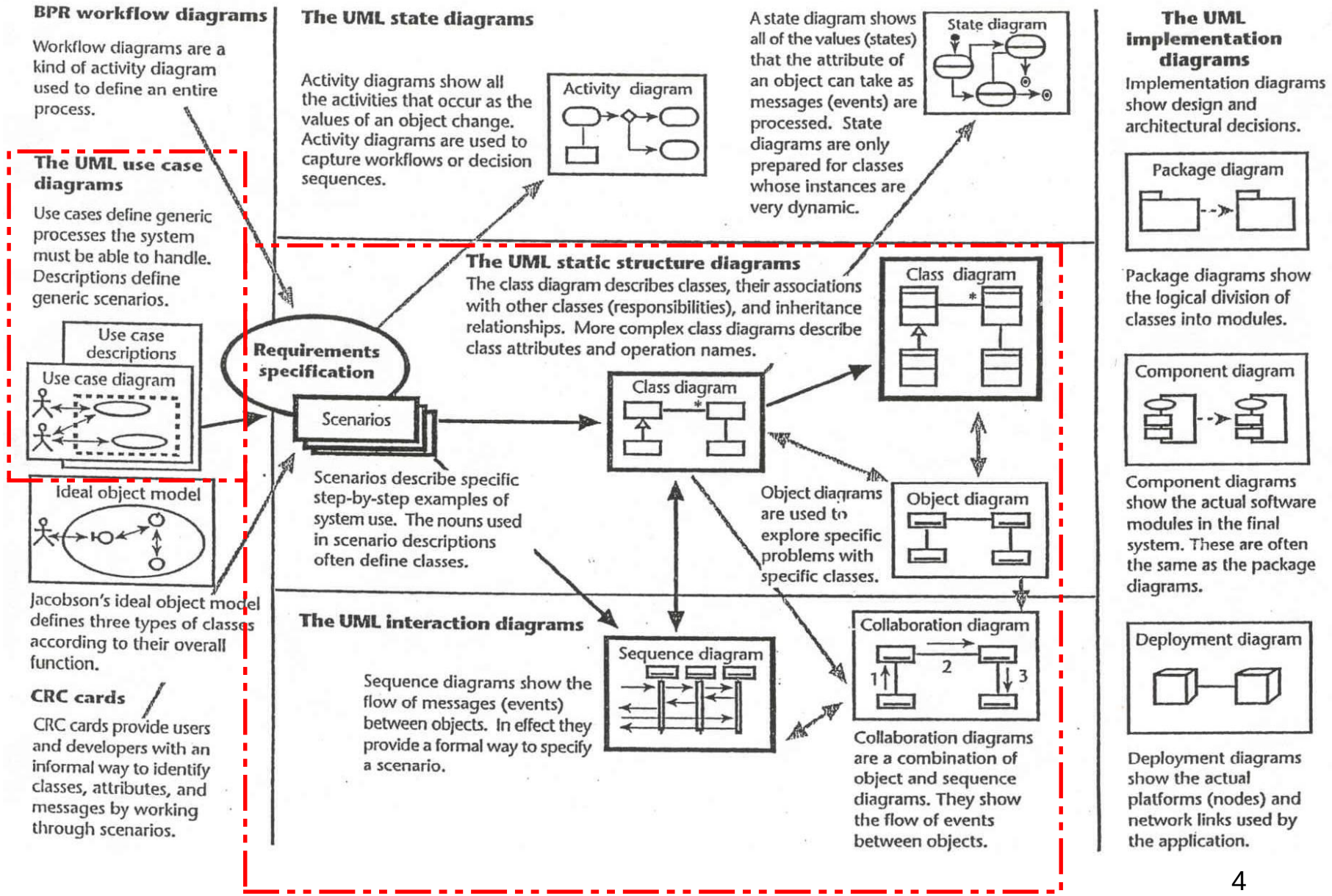


- **Class diagrams** represent the static structure of the system in terms of *classes* and *relationships*

- **Interaction diagrams** (realization of specific scenario of the use case):
  - **Sequence diagrams:** *temporal* rep of interactions between objects.
  - **Communication diagrams** *spatial* rep of interactions between objects

- Additional UML Diagrams (we won't use these in this course):
  - **Deployment diagrams** represent the deployment of components on particular pieces of hardware
  - **Object diagrams** a simplified collaboration diagram w/o message broadcasts
  - **Activity diagrams** represent the behavior of an operation as a set of actions
  - **Component diagrams** represent the physical components of an application
  - **Statechart diagrams:** Represent class behavior in terms of state

Relationships Between UML Diagrams



OOA Consists of Two Basic Steps:

1. **Use-case modeling** (Mostly Action-Oriented, behavior of system from the user/external entity perspective)
   - How Results are Computed by Product (w/o rt Sequencing)
   - Uses *Scenarios* And *Use Cases*
2. **Class Modeling** ("Object Modeling") (Purely Data Oriented)
   - Determine Classes, Attributes
   - Relationships Between Objects
   - Deduce Classes From: Use Cases, Noun Extraction

- Note: OOA is Iterative, above Steps Repeatedly Revisited

Use Cases (Step 1 of OOA)

- Use Cases Model *Intended* Behavior of System, without concern for how the Behavior will be *Implemented*.
- A Use Case Carries out Tangible Work of Value from the Perspective of an *Actor*. Examples:
  - Calculate a Result,
  - Generate a New Object, or
  - Change the State of another Object

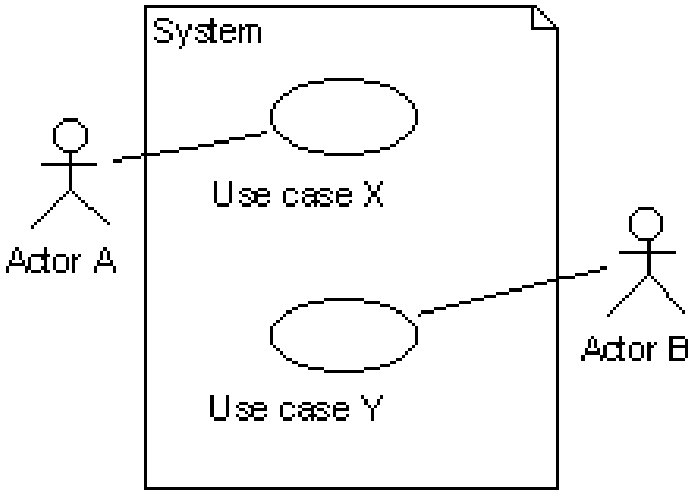
- UML Notation Allows Visualization of a Use Case Apart from its Realization and in Context with other Use Cases.
- An Actor's role is to trigger (communicate with) a use case.

Use Cases => Describe System behavior from user's standpoint.
Definition of System boundary/relationships with Environment.

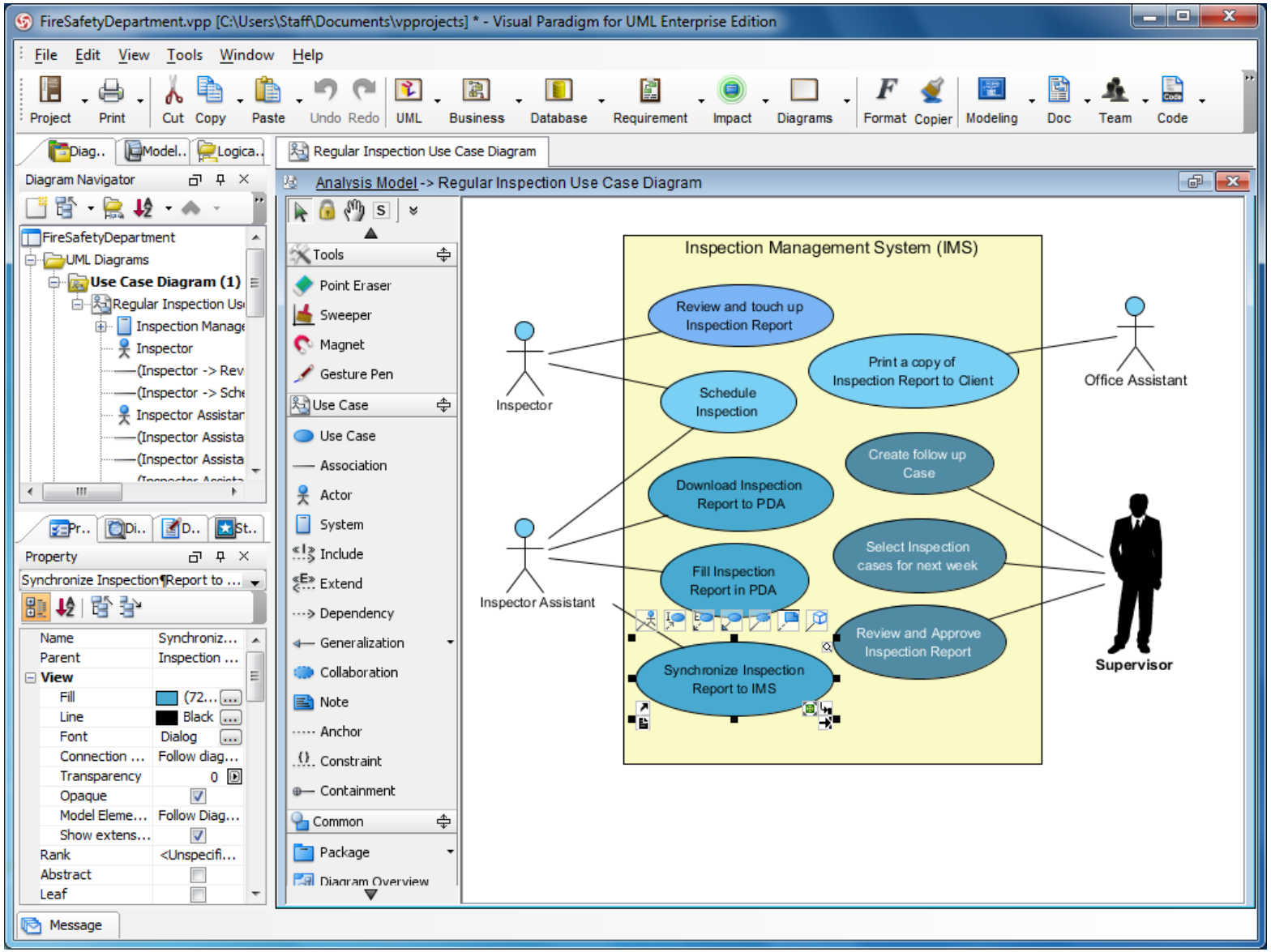
The Role of Actors in Use Cases

- **Actor:** Represents a role played by a Person or Machine that interacts with the System as part of the use-case
  - Actor typically causes system to respond by providing input to the system, and
  - Observes or otherwise uses output from the system.
  - The name of the actor describes the role played by the user

- Actors are determined by observing the direct users of the system
  - Same physical person may play the role of several actors
  - Several people may act as the same actor



Ex: Use Case Diagram

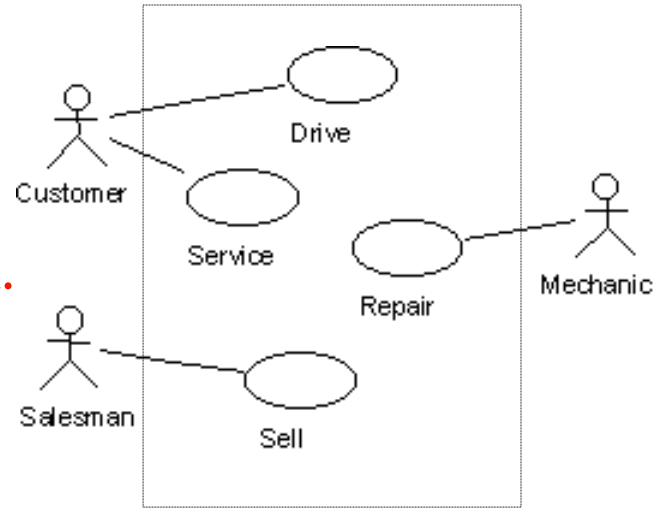


- Source: http://www.visual-paradigm.com/product/vpuml/features/screenshots.jsp

Example: A Garage Owner

- Spends most of his time acting as a mechanic, but may sometimes act as a salesman. On weekends, he plays the role of customer and services his own car

•Actors are recruited from users, customers, suppliers, and are the people and things outside a system that interact with the system.



Four Main Categories of Actors:

- ***Principal actors*:** People who use the main system functions.
- ***Secondary actors:*** People that perform admin or maintenance tasks.
- ***Integral hardware***: The unavoidable hardware devices that are part of the application domain and must be used.
- ***Other systems*:** The other systems with which system must interact.

Elevator Problem: OOA

Step I of OOA: Use-Case Modeling

- *Use Case*: Generic Description of Overall Functionality
  - *Scenario*: Instance of a Use Case
  - Consider Typical Scenarios of Activities of each Class
- *Goal:* Obtain Insight into Product Behavior

- Example: Consider an elevator control system that controls a bank of elevators in a high-rise.
  - What Actors and Use-Cases are relevant to the system?
  - As a starting point, think of various scenarios of use.

Scenarios in Terms of UML

- Scenario: an *Instance* of a Use Case, Explores "Behind the Scenes behavior." Fertile ground for acceptance test cases.
  - **Normal** scenario (intended uses of system):
    - User Wants to use Elevator to go from Floor 3 to Floor 7, Presses "Up" Button. Elevator is currently empty.
  - **Exception** (Abnormal) scenario (unintended, but possible):
    - User "A" Wants to go from Floor 3 to Floor 1, but Presses Up Button. Elevator Already Contains User "B" who Entered at Floor 1 and is Going to Floor 9.

Is the below Normal, an Exception (Abnormal), or Not Useful as a Scenario?

Example: User presses for Floor 3, but is instead taken to Floor 8 (with no other users of the system involved).

Normal Scenario Go from Floor 3 to Floor 7, Press "Up" Button

1. User presses Up floor button at floor 3 to request elevator. User wishes to go to floor 7.
2. Up floor button is turned on.
3. Elevator arrives at floor 3, Up floor button is turned off.
4. Elevator doors open. User enters elevator.
5. User presses elevator button for floor 7.
6. Floor 7 elevator button is turned on.
7. Elevator doors close.
8. Elevator travels to floor 7.
9. Floor 7 elevator button is turned off. What's next?

ICE: Exception (Abnormal) Scenario

Scenario: User "A" Wants to go from Floor 3 to Floor 1, but Presses Up Button. Elevator Already Contains User "B" who Entered at Floor 1 and is Going to Floor 9.

Why exception (abnormal)?

- Develop the Scenario step-wise.

The Role of Use-Cases and Scenarios

- The scope of use cases and scenarios goes far beyond solely defining system requirements; allow:
  - navigation first towards the classes and objects that collaborate to satisfy a **requirement**, then
  - towards the **tests** that verify the system performs its duties correctly (i.e., validation).
- Use-Cases and Scenarios are used during various phases of object-oriented software development:

Specification/Analysis	**Design**	**Integration**
•Spell out <u>what</u> system is supposed to do via Uses-Cases and Scenarios.	• Show <u>how</u> each specific part of a Scenario is met by classes/methods.	•Acceptance Testing. Demo that each Scenario is indeed met by the system.

Aside: Walkthroughs

Technique Used to Uncover Application's Desired Behavior

- Pretend You Already Have a Working Application, Walk Through the Various Uses of the System
- Walkthroughs Help To Uncover All *Intended* Uses of a System
- **Question:** When do you stop walking through scenarios?

Day 2 of OOA, but first…a review

Step I of OOA: Use-Case Modeling

- *Use Case*: Generic Description of Overall Functionality
  - *Scenario*: Instance of a Use Case
  - Consider Typical Scenarios of Activities of each Class
- *Goal:* Obtain Insight into Product Behavior

Example: Much-o Fantastic-o Kitchen Assistant

Capabilities:

- Browse recipes stored in a database
- Add a new recipe to the database
- Edit an existing recipe
- Plan a meal consisting of several recipes
- Scale a recipe/meal for a number or people
- Plan meals for a period of time (# days)
- Generate grocery list, includes all items in period's menu

Critical: Keep these separate from each other in your designs:

- Any database,
- the info stored in the DB,
- and rest of the system.

ICE: Develop uses-cases and scenarios for the MFKA

Scenario 1: Plan meals and generate a grocery list for a week.

Scenario 2: Edit Chicken Soup Recipe

Take good notes for the MFKA! (you might see this again)

Step II of OOA: Class Modeling

Goal: Extract Classes &Attributes, represent Relationships (including Inheritance) *between* Classes.

- Various Approaches:
  - Deduce Classes from Use Cases and their Scenarios
    - Often many Scenarios
    - Danger of inferring too many Candidate Classes
  - **Noun Extraction**
    - 'Always' Works
      (i.e. Gives You Something to Start With)

Noun Extraction Approach to Class Modeling

- For Developers Without Domain Experience
- Consists of Three Stages from highly to less Abstract:
  - Stage 1: Concise Definition
  - Stage 2: Informal Strategy
  - Stage 3: Formalize the Strategy

Stage 1 of Noun Extraction: **Concise** Definition

- Define Product as Concisely as Possible (in Single Sentence if possible!)

```
Buttons in elevators and on floors are
used to control motion of n elevators
in building with m floors
```

Stage 2 of Noun Extraction: **Informal Strategy**

- Incorporate Constraints into Stage 1
- Express Result (preferably) in a Single Paragraph

```
Elevators have call buttons and floor
buttons that control movement of n
elevators in building with m floors.
Buttons illuminate when pressed by user
to request elevator to stop at specific
floor; illumination is canceled when
request has been satisfied.  If
elevator has no requests, it remains at
its current floor with its doors
closed.
```

Stage 3 of Noun Extraction: **Formalize the Strategy**

- Identify Nouns in Informal Strategy for use as Candidate Classes:
  - First, what are the nouns from stage 2?
  - Then, exclude those nouns that are outside problem boundary, and identify abstract nouns (abstract nouns may become attributes). The nouns that remain become Candidate Classes for your design.

Stage 3 of Noun Extraction: **Formalize the Strategy**

- Identify Nouns in Informal Strategy for use as Candidate Classes:
  - First, what are the nouns from stage 2?

```
Elevators have call buttons and floor
buttons that control movement of n
elevators in building with m floors.
Buttons illuminate when pressed by user
to request elevator to stop at specific
floor; illumination is canceled when
request has been satisfied.  If elevator
has no requests, it remains at its
current floor with its doors closed.
```

Stage 3 of Noun Extraction: **Formalize the Strategy**

- Identify Nouns in Informal Strategy for use as Candidate Classes:
  - First, what are the nouns from stage 2?

Elevators have call **buttons** and **floor buttons** that control **movement** of n **elevators** in **building** with m **floors**. **Buttons** illuminate when pressed by **user** to request **elevator** to stop at specific **floor**; **illumination** is canceled when **request** has been satisfied. If **elevator** has no **requests**, it remains at its current **floor** with its **doors** closed.

Stage 3 of Noun Extraction: **Formalize the Strategy**

- Identify Nouns in Informal Strategy for use as Candidate Classes:
  - Then, exclude those nouns that are outside problem boundary, and identify abstract nouns (abstract nouns may become attributes). The nouns that remain become Candidate Classes for your design.

All Nouns	Abstract Nouns	Candidate Classes
elevator(s) call button(s) floor button(s) movement building floor(s) user illumination request doors		

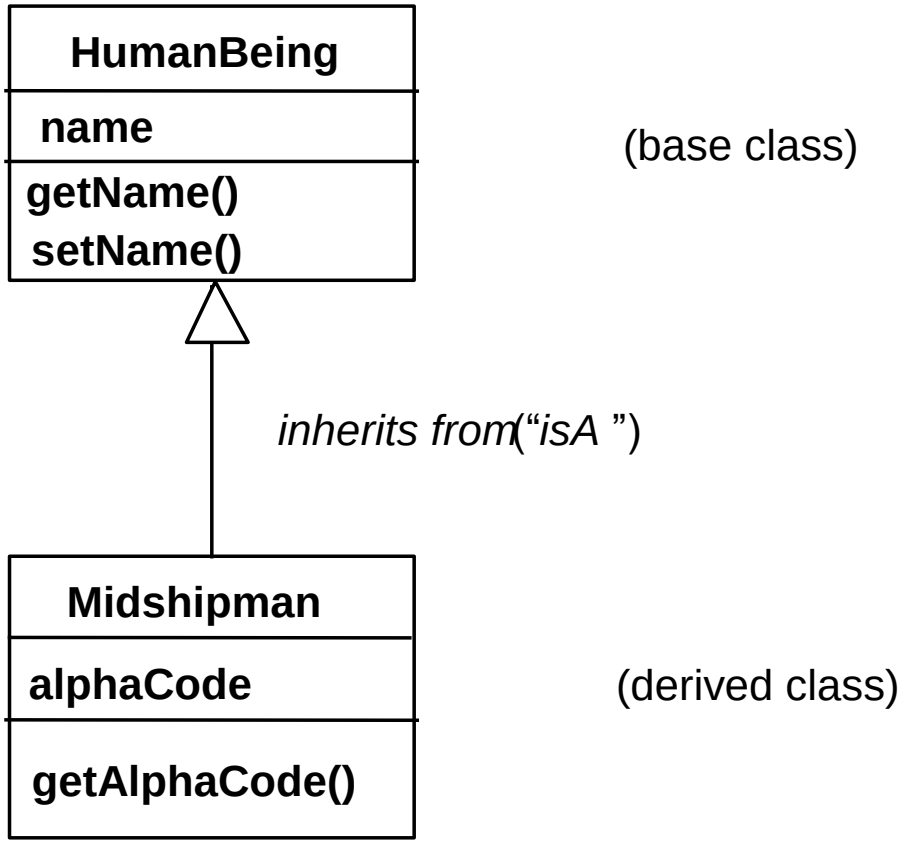
Stage 3 of Noun Extraction: **Formalize the Strategy**

- Identify Nouns in Informal Strategy for use as Candidate Classes:
  - Then, exclude those nouns that are outside problem boundary, and identify abstract nouns (abstract nouns may become attributes). The nouns that remain become Candidate Classes for your design.

All Nouns	Abstract Nouns	Candidate Classes
elevator(s)		elevator
call button(s)		call button
floor button(s)		floor button
movement	movement	
~~building~~		
~~floor(s)~~		
~~user~~		
illumination	illumination	
~~request~~		
~~doors~~		

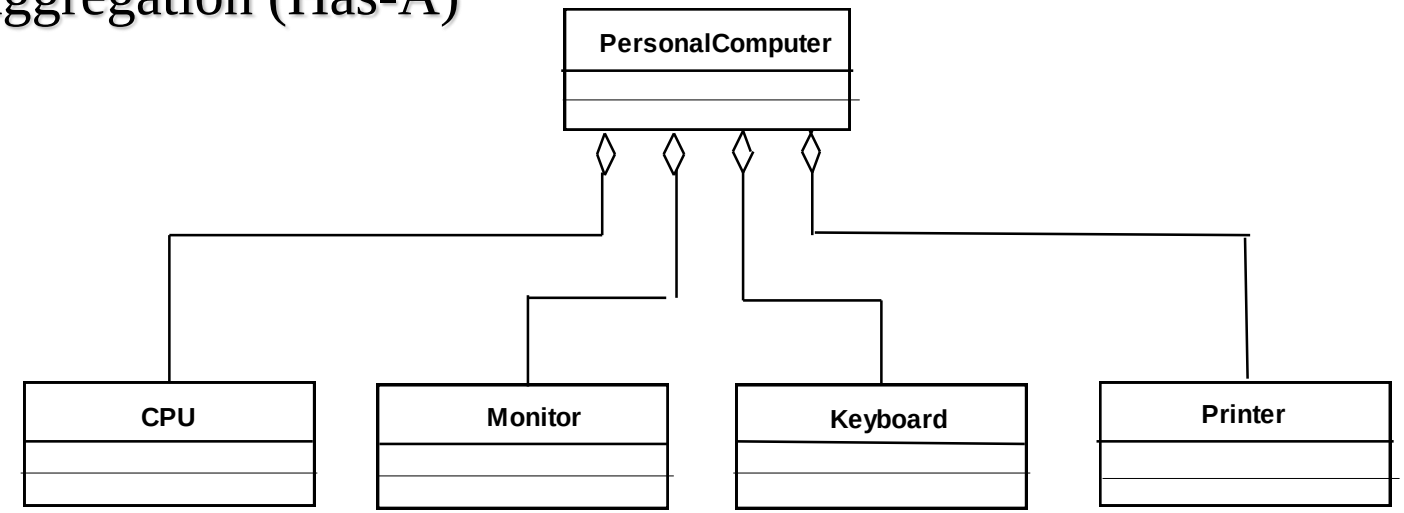
Aside: UML notation for Is-A, Has-A, Association

- Inheritance (Is-A) represented by a large open triangle

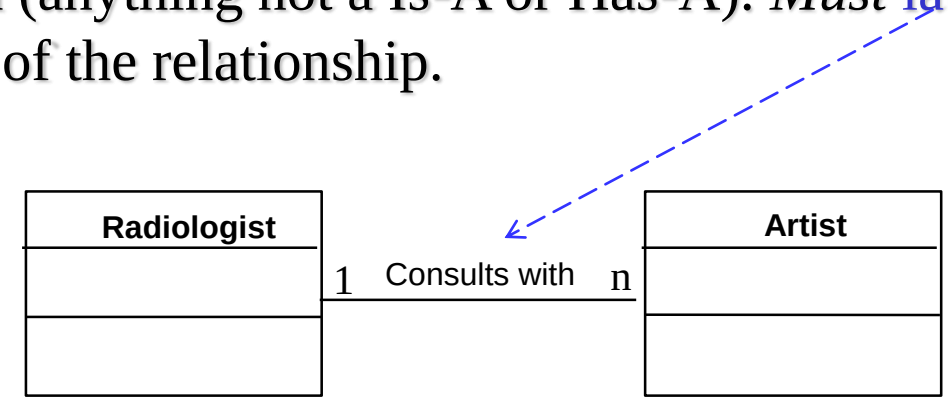


UML Notation (cont'd)

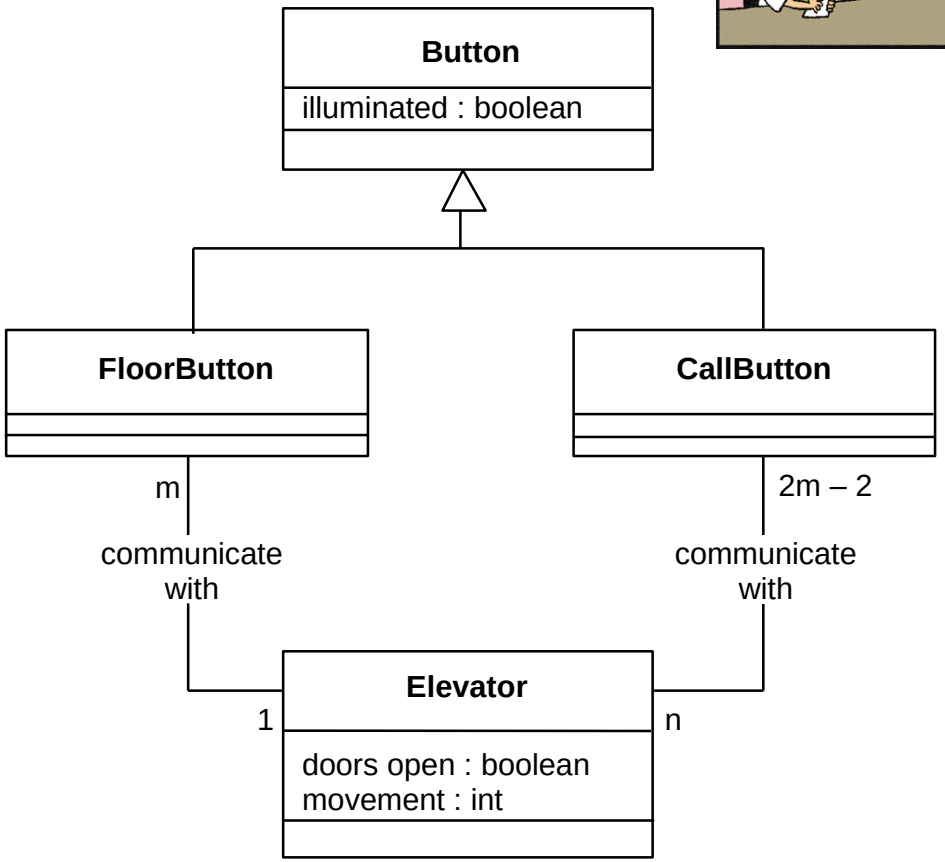
- Aggregation (Has-A)



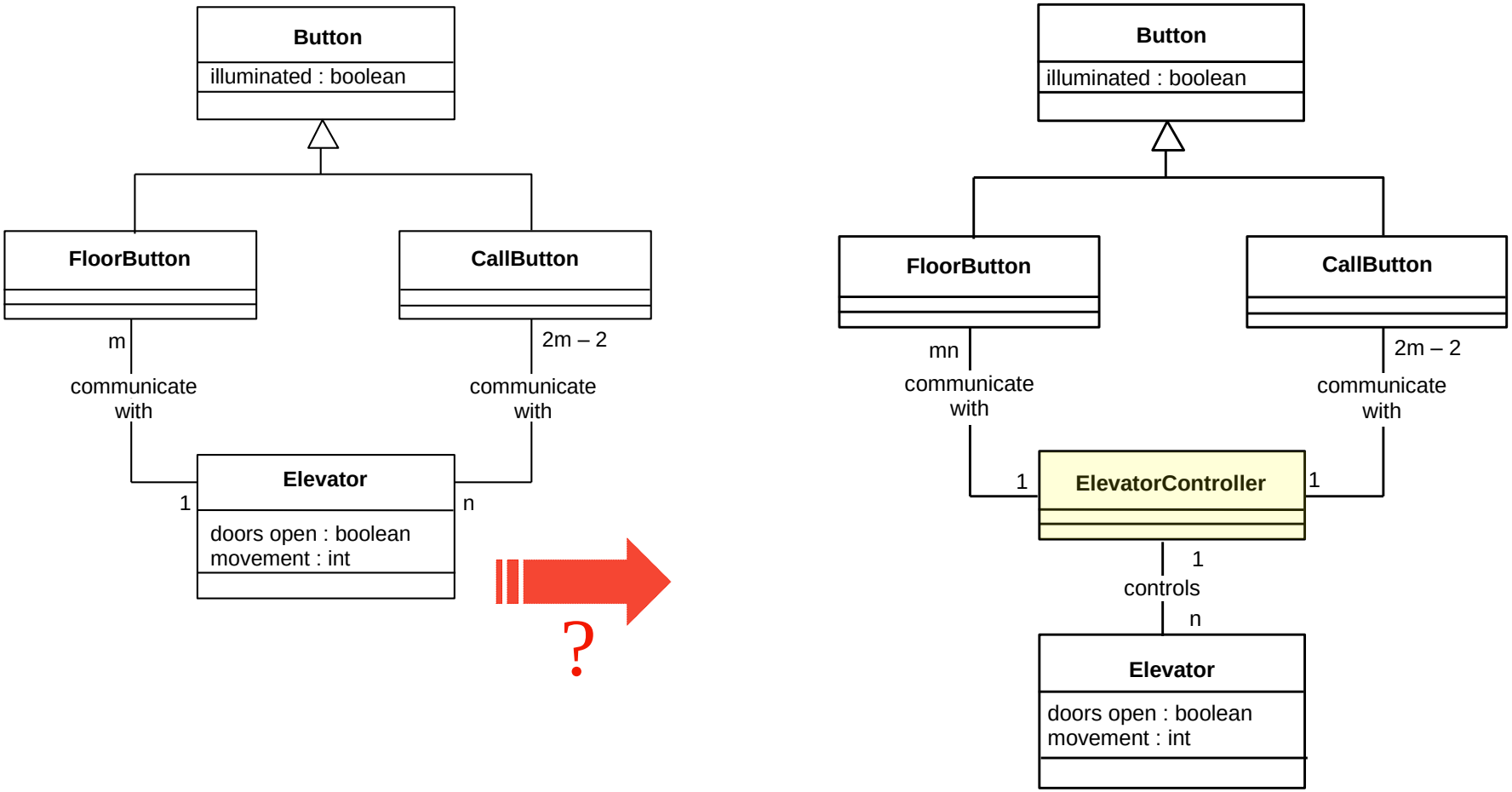
- Association (anything not a Is-A or Has-A). *Must* label the edge with description of the relationship.



First Iteration of Elevator System Class Diagram



Second Iteration of Class Diagram



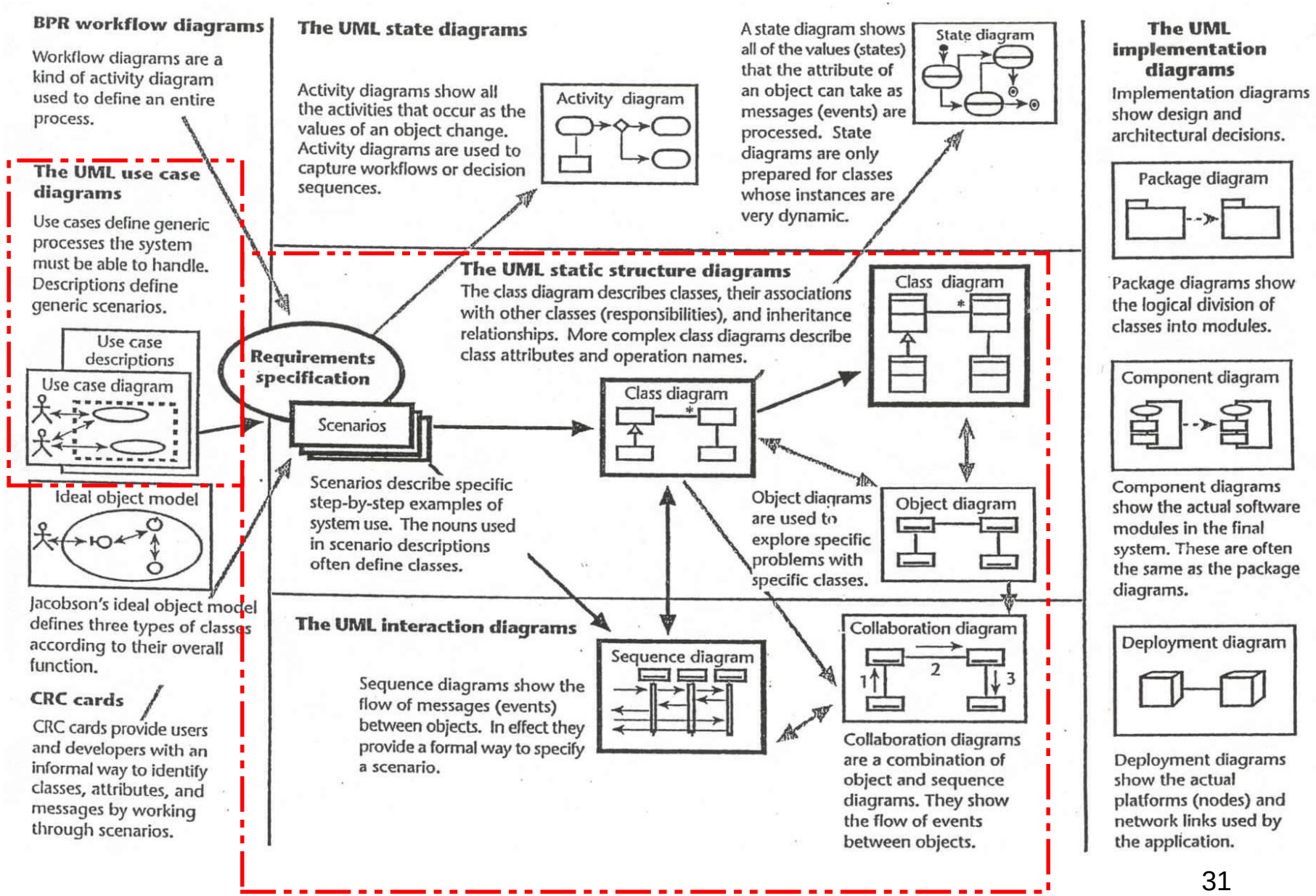
2nd Iteration: Add a Controller Class to determine which elevator is sent to a Floor Button Request.

Note: OOA is an *intentionally* iterative process.

Is All This Iteration Really Needed?

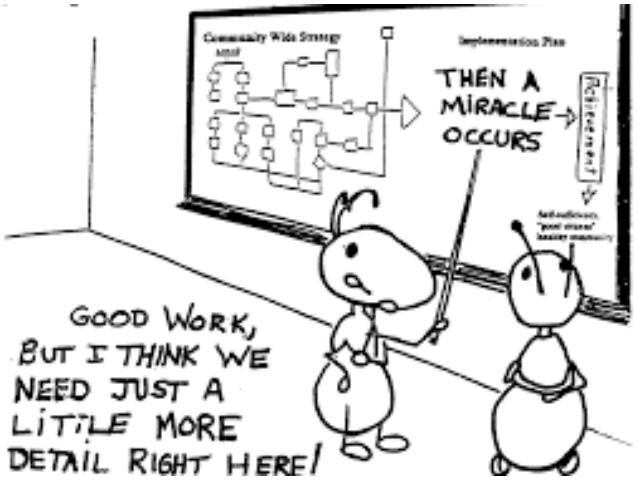
- All software development models include iteration.
  - Waterfall
  - Spiral
  - Agile
    - Latter 2 Models Explicitly Reflect Iterative Approach
- Is iteration *Intrinsic* or *Extrinsic* to the Software Development Problem?
  - Without iteration, have to get everything exactly right the first time!

Recap: Relationships Between UML Diagrams

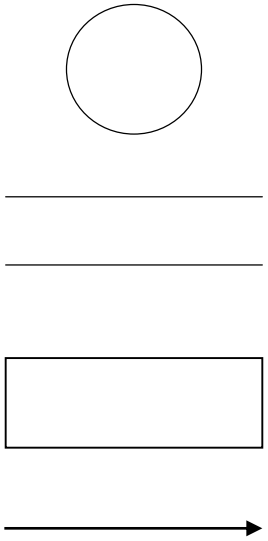


DFDs: when OO doesn't quite fit

- Data Flow Diagrams (DFDs) are used for modeling *non*-object oriented systems. (Yourdon/DeMarco)
  - Shows where data comes from,
  - where data goes,
  - where data stored, and
  - what happens to data on the way.



- These four things are the *only* things that can happen to data in a DFD.
- DFDs show the overall picture of a system, and some of the detail.



a process: alters data, computes new data

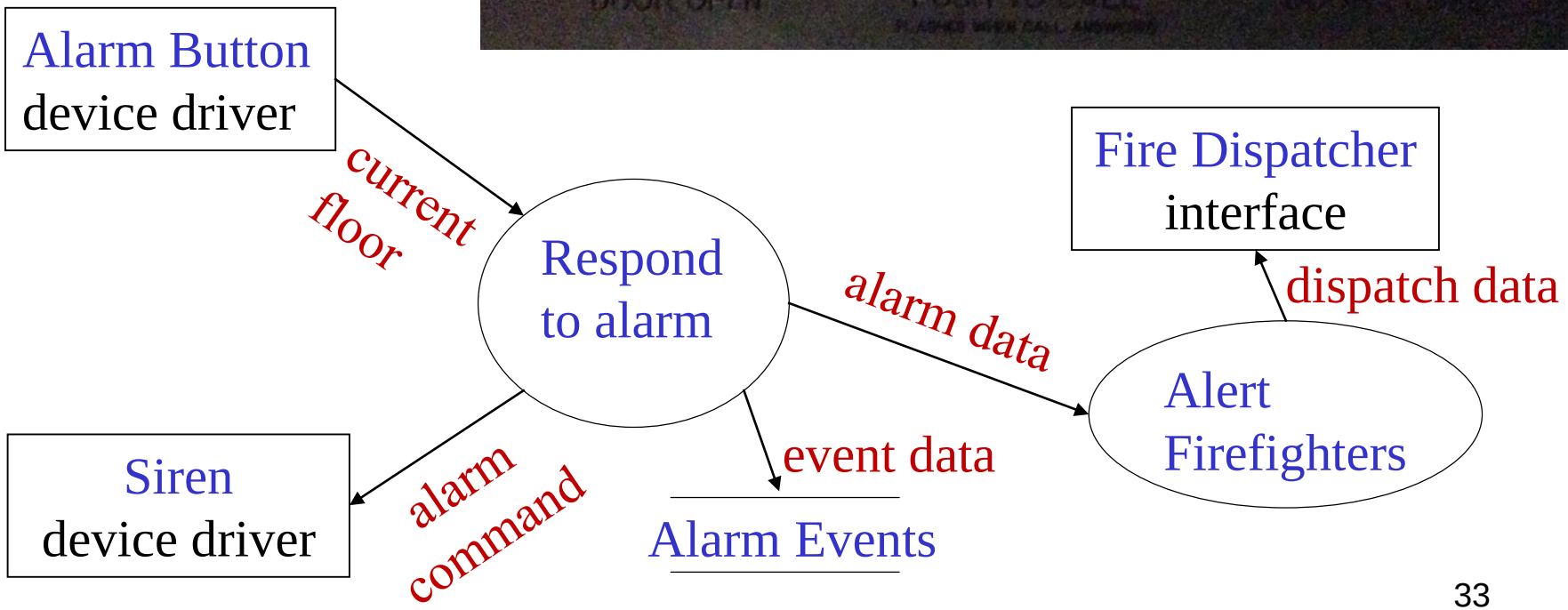
file/database: stores data

data source/sink (I/O): hardware device, user

data flow

DFD - Elevator Example (Level 1)

- A DFD modeling an elevator's alarm



Transitioning To The OOD Phase

- Once Specification Contract is Approved (Signed), the following is Delivered to the Design Team:
  - Specification Document,
  - Use Case Scenarios,
  - UML Use-Case, Class Diagrams
  - DFDs (also any ERs, FSMs, etc)

- Object-Oriented Design uses the above as the beginning of *high level* design.