CSE4006 Software Engineering

06. Analysis Modeling

Scott Uk-Jin Lee

Department of Computer Science and Engineering
Hanyang University ERICA Campus

1^{st} Semester 2015

lab(se);

Overview of Analysis Modeling

1. Requirement Analysis
2. Analysis Modeling Approaches
3. Data Modeling Concepts
4. Object-Oriented Analysis
5. Scenario-based Modeling
6. Class-based Modeling
7. Flow-oriented Modeling
8. Behavioral Modeling

Requirements Analysis

- At a technical level, SE begins with a building an analysis model of a target system
- Requirements analysis
  - specifies software's **operational** characteristics
  - indicates software's **interface** with other system elements
  - establishes **constraints** that software must meet
- Objectives
  1. to describe what the customer requires
  2. establish a basis for the creation of a SW design
  3. to define **requirements** that **validated** a set of can be once the software is built

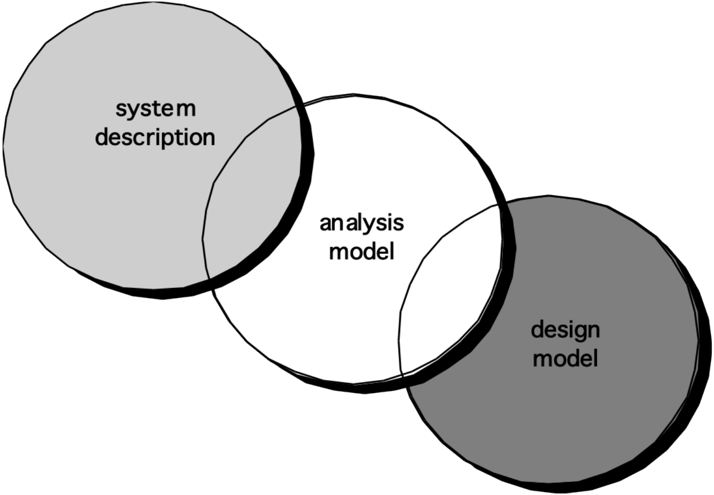
lab(se);

Requirements Analysis

- Requirements analysis allows the software engineer to:
  - elaborate on basic requirements established during earlier requirement engineering tasks
  - build models that depict
    - user scenarios
    - functional activities
    - problem classes and their relationships
    - system and class behavior
    - flow the of data as it is transformed

lab(se);

A Bridge



Elements of Analysis Model

Analysis Model

Scenario-based Elements
- Use Cases
- Activity diagrams
- Swimlane diagrams

Flow-Oriented Elements
- Data flow diagrams
- Control flow diagrams
- Processing narratives

Class-based Elements
- Class diagrams
- CRC models
- Collaboration diagrams

Behavioral Elements
- State diagrams
- Sequence diagrams

Rules of Thumb

1. The model should focus on requirements that are **visible** within the problem or business domain
   - the level of abstraction should be **relatively high**
2. Each element of the analysis model should
   - add to an overall understanding of software requirements
   - provide insight into the
     - information domain
     - function of the system
     - behavior of the system
3. Delay consideration of infrastructure and other non-functional models until design
4. Minimize coupling throughout the system
5. Be certain that the analysis model provides value to **all stakeholders**
6. Keep the model as simple as it can be

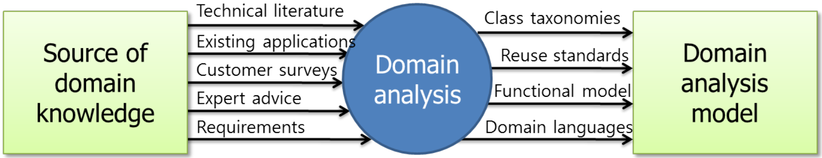
Domain Analysis

- Software Domain Analysis (Structured Analysis)
  - the identification, analysis, and specification of **common requirements** from a specific application domain, typically for **reuse** on multiple projects within that application domain ...
- Object-Oriented Domain Analysis
  - the identification, analysis, and specification of common, reusable capabilities within a specific application domain, in terms of common objects, classes, subassemblies, and frameworks . . .

by Donald Firesmith

- Define the domain to be investigated
- Collect a representative **sample** of applications in the domain
- Analyze each application in the sample
- Develop an analysis model for the objects.

Domain Analysis

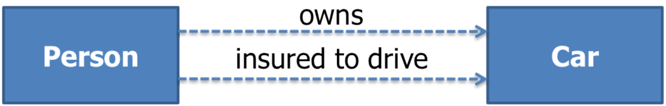


Data Modeling

- Analysis modeling often begins with data modeling
  - examines data objects independently of processing
  - focuses attention on the data domain
  - Indicates how data objects relate to one another
- **Relationship** among data objects can be expressed in UML very well
- Typical data objects
  - External entities: printer, user, sensor
  - Things: reports, displays, signals
  - Occurrences or events: interrupt, alarm
  - Roles: manager, engineer, salesperson
  - Organizational units: division, team
  - Places: manufacturing floor
  - Structure: employee record

Data Modeling

- Data objects, data attributes, relationships
  - Data objects are the representation of composite information that are process by software
  - A data object encapsulates data only
- Entity Relationship Diagram (ER Diagram)



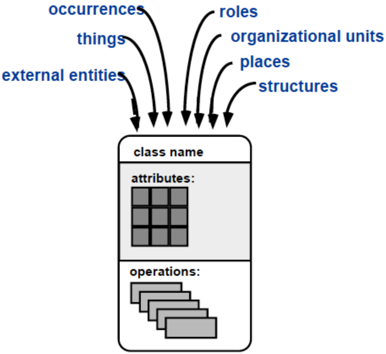
lab(se);

Object-Oriented Concepts

- Must be understood to apply class-based elements of the analysis model
- Key concepts:
  - Classes and objects
  - Attributes and operations
  - Encapsulation and instantiation
  - Inheritance

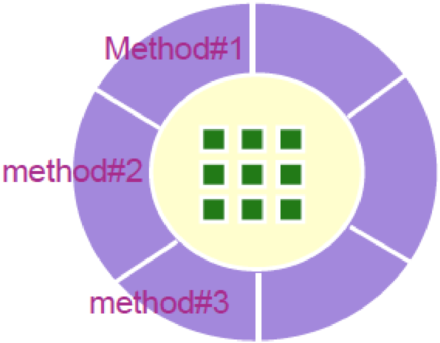
Classes

- Object-oriented thinking begins with the **definition** of a class
  - class is often defined as: templates, generalized description, "blueprint" (describing a collection of similar items)
- A superclass establishes a hierarchy of classes
- Once a class of items is defined, a specific instance of the class can be identified



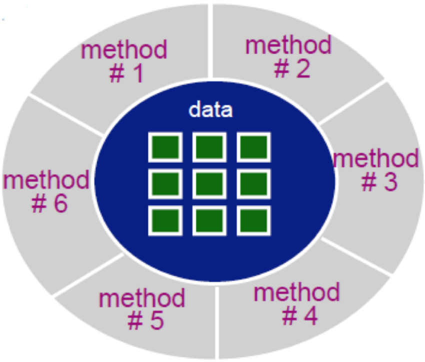
Methods (Operations, Services)

- an executable procedure that is encapsulated in a class
- designed to operate on one or more data attributes that are defined as part of the class



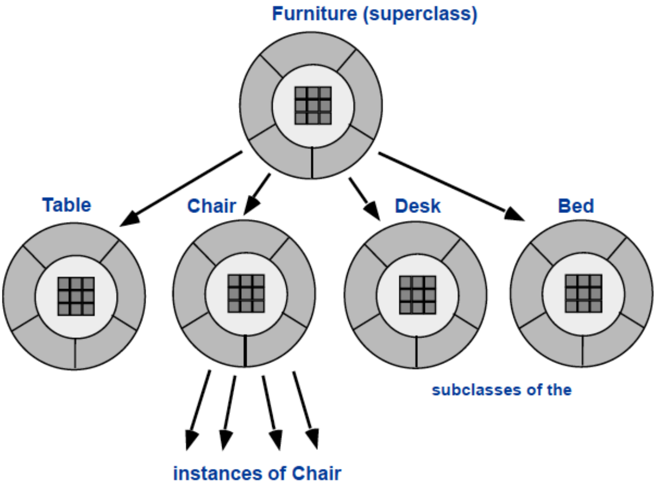
Encapsulation / Hiding

- The object **encapsulates** both data and the logical procedure required to manipulate the data ⇒ **information hiding**



lab(se);

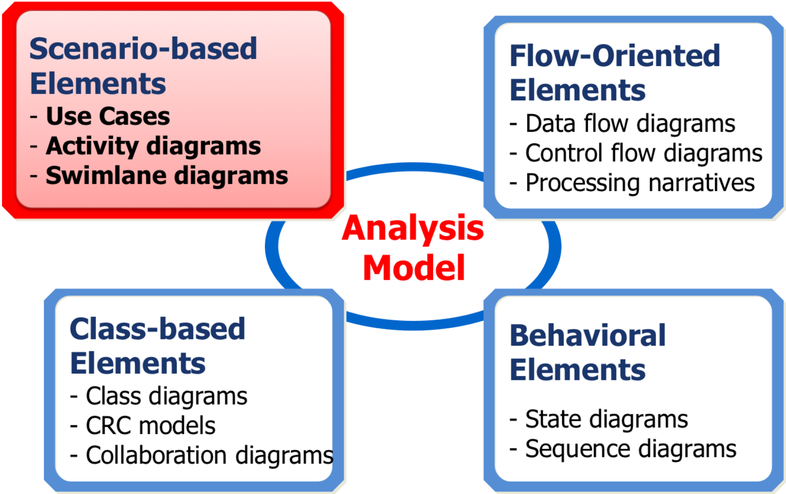
Class Hierarchy



How to Define All Classes

1. Basic **user requirements** must be communicated between the customer and the software engineer
2. Classes must be identified
   - Attributes and methods are to be defined
3. A class hierarchy is defined
4. Object-to-object relationship should be represented
5. Object behavior must be modeled
6. Tasks 1 through 6 are repeated until the model is complete

Elements of Requirements Analysis



Scenario-Based Modeling

- **Use-cases** are amply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases)

by Ivar Jacobson

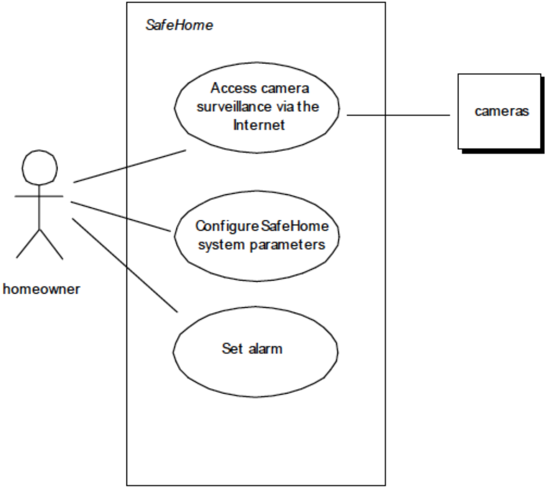
1. How should we write about?
2. How much should we write about?
3. How detailed should we make our description?
4. How should we organize the description?

lab(se);

Use-Cases

- a scenario that secibes a "thread of usage" for a system
- **actors** represent roles people or devices play as the system functions
- **users** can play a number of different roles for a given scenario
- Developing a use case
  - What are the main tasks or functions that are perfumed by the actor?
  - What system information will the actor acquire, produce or change?
  - What information does the actor desire form the system?

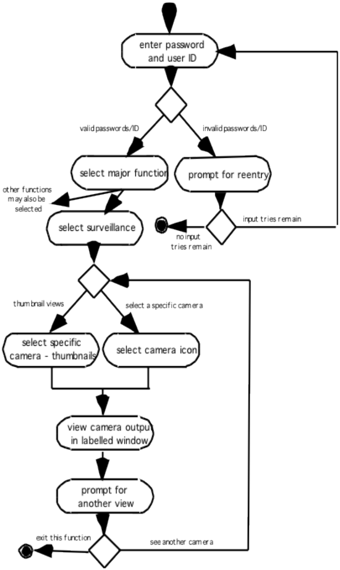
Use-Case Diagram

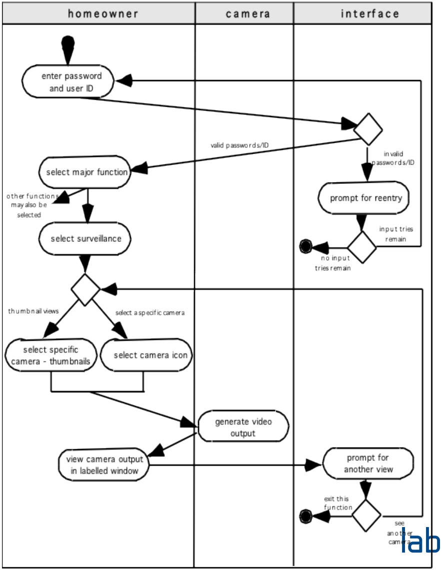


Activity & Swimlane Diagram

- Activity Diagram
  - Supplements the use-case by providing a diagrammatic representation of **procedural flow**
    - used as business modeling notation along with Business Process Modeling Notation (BPMN)
- Swimlane Diagrams
  - Allows the modeler to represent the flow of activities described by the use-case
  - This diagram indicates which actor or analysis class has responsibility for the action described by an activity rectangle

Activity & Swimlane Diagram





Elements of Requirements Analysis

Analysis Model

Scenario-based Elements
- Use Cases
- Activity diagrams
- Swimlane diagrams

Flow-Oriented Elements
- Data flow diagrams
- Control flow diagrams
- Processing narratives

Class-based Elements
- Class diagrams
- **CRC models**
- Collaboration diagrams

Behavioral Elements
- State diagrams
- Sequence diagrams

Class-based Modeling

- Class-based modeling represents:
    - objects that the system will manipulate
    - operations (methods or services) that will be applied to the objects to effect the manipulation
    - relationships (some hierarchical) between the objects
    - collaborations that occur between the classes that are defined
- The elements of a class-based model
    - classes, objects, attributes, operations
    - CRC models
    - collaboration diagrams and packages

lab(se);

Identifying Analysis Classes

- Examining the usage scenarios developed as part of the requirements model and perform a "**grammatical parse**"
  - Identify **analysis classes** by examining the problem statement
  - Identify **attributes** of each class
  - Identify **operations** that manipulate the attributes

Analysis Classes

- External entities: produce or consume information
  - other systems, devices, people
- Things: part of the information domain for the problem
  - reports, displays, letters, signals
- Occurrences or events: occur within the context of system operation
  - a property transfer or the completion of a series of robot movements
- Roles: played by people who interact with the system
  - manager, engineer, salesperson
- Organizational units that are relevant to an application
  - division, group, team
- Places: establish the context of the problem and the overall function
  - manufacturing floor or loading dock
- Structures: define a class of objects or related classes of objects
  - sensors, four-wheeled vehicles, or computers

Selecting Classes - Criteria

- criteria for selecting analysis classes from potential classes
  1. Retained information
  2. Needed services
  3. Multiple attributes
  4. Common attributes
  5. Common operations
  6. Essential requirements

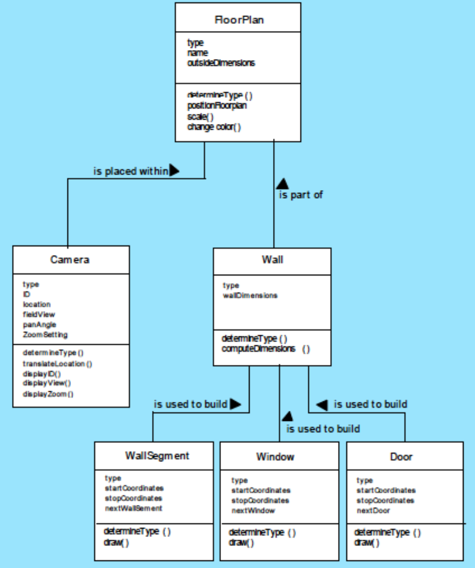
$\Rightarrow$ potential class must satisfy all (most of) the criteria above

lab(se);

Example Potential SafeHome Classes

Potential class	Applicable criteria No.
Homeowner	rejected: 1, 2 fail even though 6 applies
Sensor	accepted: all apply
Control panel	accepted: all apply
Installation	rejected
Alias security function	accepted: all apply
Number, type	rejected: 3 fails, attributes of sensor
Master password	rejected: 3 fails
Telephone number	rejected: 3 fails
Sensor event	accepted: all apply
Audible alarm	accepted: 2,3,4,5,6 apply
Monitoring service	rejected: 1,2 fail even though 6 applies

Class Diagram



CRC Models

- Class-Responsibility-Collaborator (CRC) Modeling
- Analysis classes have "responsibilities"
  - **Responsibilities** are the attributes and operations encapsulated by the class
- Analysis classes collaborate with one another
  - a property transfer or the completion of a series of robot movements
- **Collaborators** are those classes that are required to provide a class with the information needed to complete a responsibility
- In general, a collaboration implies either a request for information or a request for some action

- CRC card

Class: FloorPlan

Description:

Responsibility:	Collaborator:
defines floor plan name/type	
manages floor plan positioning	
scales floor plan for display	
scales floor plan for display	
incorporates walls, doors and windows	Wall
shows position of video cameras	Camera

Class Types in CRC Modeling

- Entity classes (model/business classes)
  - extracted directly from the statement of the problem
    e.g. FloorPlan and Sensor
- Boundary classes
  - used to create the interface
  - e.g. interactive screen or printed reports, CameraWindow
- Controller classes
  - designed to manage:
    - the creation or update of entity objects
    - instantiation of boundary objects as they obtain information from entity objects
    - complex communication between sets of objects
    - validation of data communicated between objects or between the user and the application

Responsibilities in CRC modeling

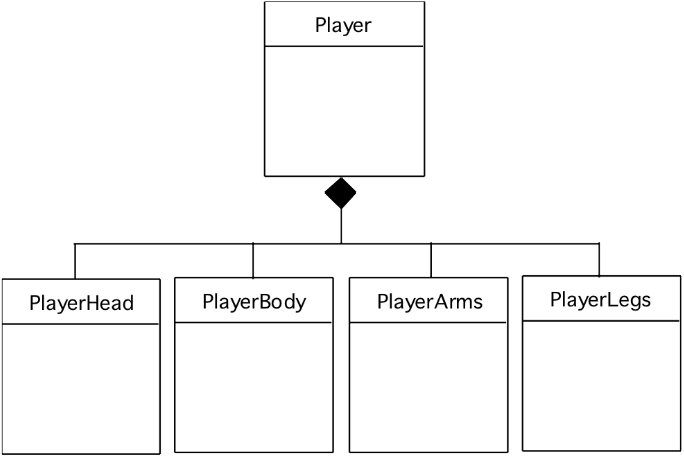
- System intelligence should be **distributed** across classes to best address the needs of the problem
  - smart classes vs. dumb classes
- Each responsibility should be stated as **generally** as possible (for higher reuse)
- Information and the behavior related to it should reside within the same class $\Rightarrow$ encapsulation
- Information about one thing should be localized with a single class
  - should not be distributed across multiple classes $\rightarrow$ difficult to maintain and test
- Responsibilities should be shared among related classes, when appropriate

Collaborations in CRC modeling

- Classes fulfill their responsibilities in one of two ways:
  - use its own operations to manipulate its own attributes
  - collaborate with other classes
- Collaborations identify relationships between classes
- Collaborations are identified by determining whether a class can fulfill each responsibility itself
- 3 different generic relationships between classes
  - the **is-part-of** relationship
  - the **has-knowledge-of** relationship
  - the **depends-upon** relationship

Composite Aggregate Class

- The "is-part-of" relationship = "aggregation" in UML



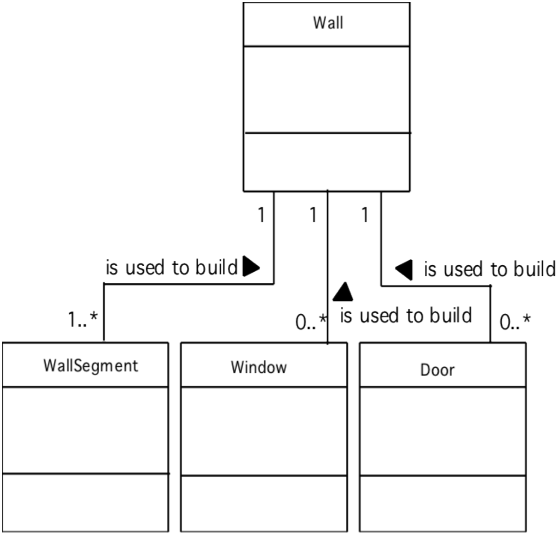
Associations and Dependencies

In UML :

- Association: two analysis classes are often related to one another in some fashion
  - multiplicity (cardinality)
- Dependency: a client-server relationship exists between two analysis classes
  - a client-class depends on the server-class

lab(se);

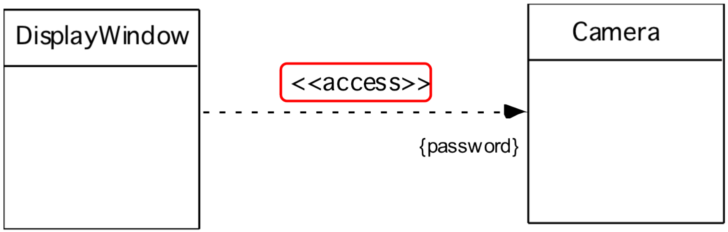
Multiplicity



lab(se);

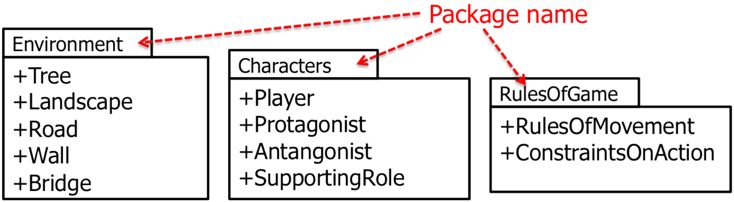
Dependencies

- Dependencies are defined by a stereotype
  - **Stereotype** is an extensibility mechanism within UML that allows user to customize the semantics of special modeling element
  - Stereotypes are represented with double angle brackets



Analysis Packages

- Various elements of the analysis model (Use-cases, analysis classes) are categorized in a manner that packages them as a grouping
  - $+$ (plus sign): public visibility for analysis classes
  - $-$ (minus sign): hidden from all other packages
  - $\#$ (sharp symbol): accessible only to packages contained within a given package



Reviewing the CRC Model

- All participants in the review (of the CRC model) are given a subset of the CRC model index cards
  - Cards that collaborate should be separated (i.e., no reviewer should have two cards that collaborate)
- All **use-case scenarios** (and corresponding use-case diagrams) should be organized into categories
- The review leader reads the use-case deliberately
  - As the review leader comes to a named object, she passes a token to the person holding the corresponding class index card

lab(se);

Reviewing the CRC Model

- When the token is passed, the holder of the class card is asked to describe the responsibilities noted on the card
  - The group determines whether one (or more) of the responsibilities satisfies the use-case requirement
- If the responsibilities and collaborations noted on the index cards cannot accommodate the use-case, modifications are made to the cards
  - This may include the definition of new classes (and corresponding CRC index cards) or the specification of new or revised responsibilities or collaborations on existing cards

Elements of Requirements Analysis

Analysis Model

Scenario-based Elements
- Use Cases
- Activity diagrams
- Swimlane diagrams

Flow-Oriented Elements
- **Data flow diagrams**
- **Control flow diagrams**
- **Processing narratives**

Class-based Elements
- Class diagrams
- CRC models
- Collaboration diagrams

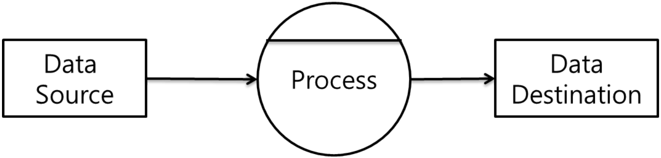
Behavioral Elements
- State diagrams
- Sequence diagrams

Flow-Oriented Modeling

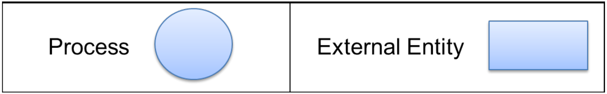
- Represents how **data** objects are **transformed** as they move through the system
- Diagrammatic notation/form: **Data Flow Diagram (DFD)**
- "Old School" approach to structured analysis/design
- continues to provide a unique view of the system
  - often used to supplement other analysis model elements

The Flow Model

- Every computer-based system = an information transform



The Flow Model Notation



- **Process**: data transformer (changes input to output)

  e.g. calculate tax, determine area, format report, display graph

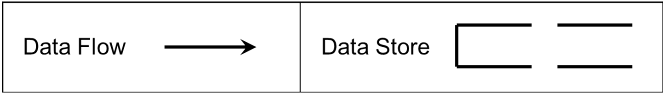
  - Data must always be **processed** in some way to achieve system function

- **External Entity**: producer(origin) or consumer(sink) of data

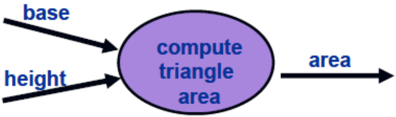
  e.g. person, device, sensor, external system

  - Data must always be **originated** from somewhere and be **sent** to something

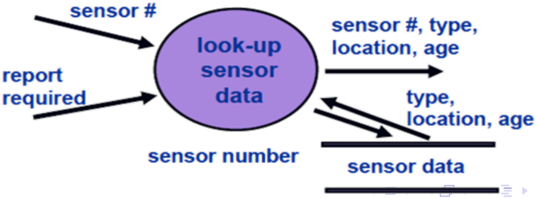
The Flow Model Notation



- **Data Flow**: Data flows through a system, beginning as input and be transformed into output

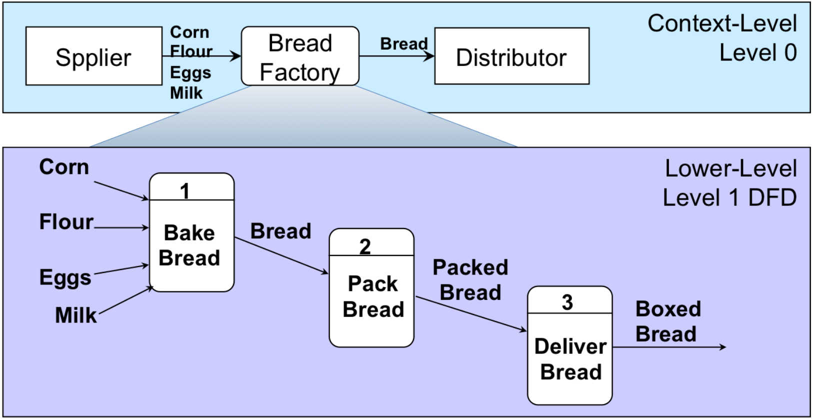


- **Data Store**: Data is often stored for later use



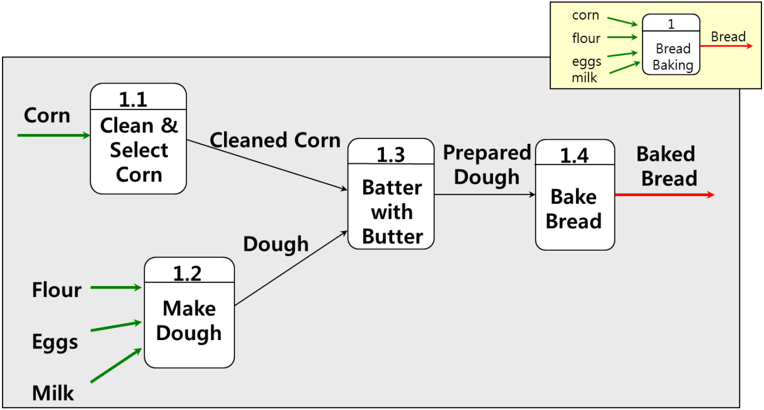
Data Flow Diagram Example: Bread Factory

- Data Flow Diagram (DFD) of Bread Factory



Data Flow Diagram Example: Bread Factory

- Bread making process refinement (Lower Level: Level 2)



Data Flow Diagramming: Guidelines

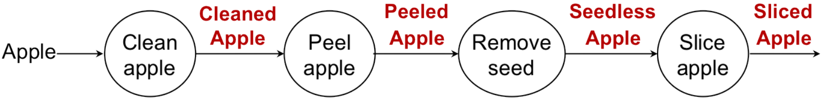
- All icons must be labeled with meaningful **names**
- The DFD evolves through a number of levels of detail
- Always begin with a context level diagram (also called level 0)
    - Top-down approach
- Always show external entities at level 0
- Always label data flow arrows
- Do **not** represent procedural logic unless DFD reaches the final level

Data Flow Diagramming: Guidelines

- Naming principles
  - use verb-noun phrase for process names
  - avoid general names that can be applied in any case

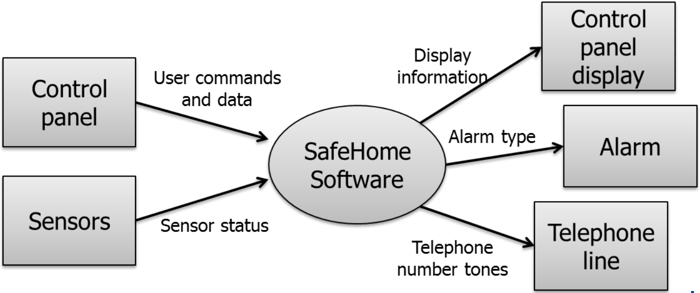
e.g. Inappropriate naming

- Naming transformed data flow
  - new name after each data flow transformation (via process)



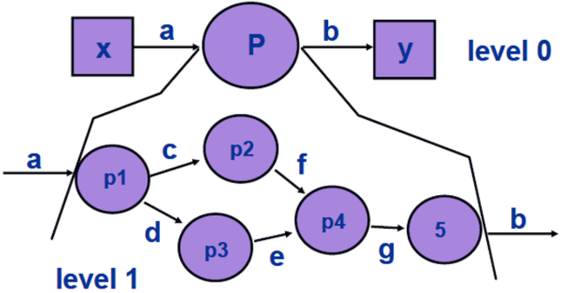
Constructing a DFD

- Review the data model
  - isolate data objects
  - use grammatical parse to determine "Operations"
- Determine external entities (producer/consumer of data)
- Create a Level 0 DFD initially



Constructing a DFD

- Write **narrative** describing the transformation
- Parse (grammatical) to determine next level transformation
  - nouns (noun phrases) and verbs (verb phrases)
- "**Balance**" the flow to maintain data flow continuity
  - flow of input/output data in different levels must be consistent
- Develop a Level 1 DFD (use a 1:5(approx.) expansion ratio)



lab(se);

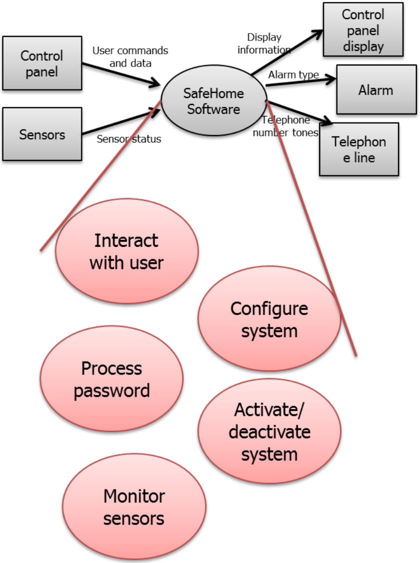
Constructing a DFD Example: SafeHome

- SafeHome processing narrative

  The SafeHome security function enables the homeowner to **configure** the security system when it its installed, **monitors** all sensors connected to the security system, and **interacts** with the homeowner through the Internet, a PC, or a control panel.

  During installation, the SafeHome PC is used to program and configure the system. Each sensor is assigned a number and type, a master password is programmed for **arming** and **disarming** the system, and telephone number are input for dialing when a sensor event occurs.

Constructing a DFD Example: SafeHome

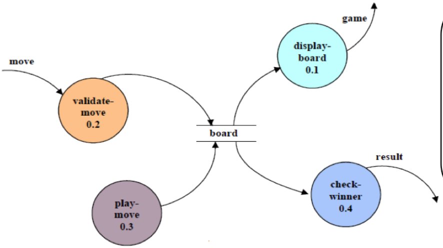


Flow Modeling Notes

- Each bubble (process) is refined until it does just **one** thing
- Expansion ratio decreases as the number of levels increase
- Most systems require 3 - 7 levels for an adequate flow model
- A single data flow item (arrow) may be expanded as levels increase
  - data dictionary provides information

Flow Model Components

Data Flow Diagram



Data Dictionary

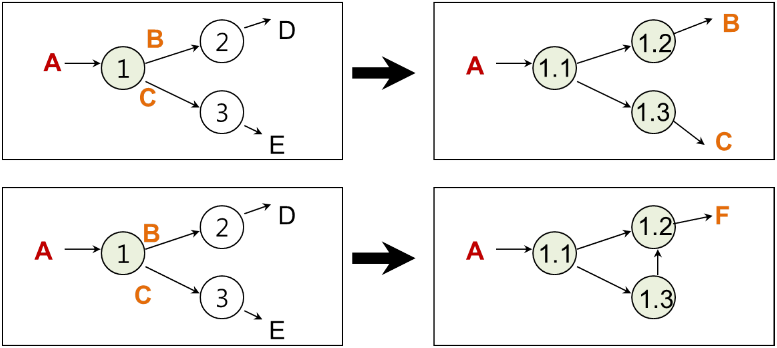
move: integer /*number between 1 and 9 */
display: game+result
game: board
board: {integer}9
result: ["computer won", "human won" "draw"]
....

Process Specification (Mini-Spec)

If the amount of the invoice exceeds $500,
If the account has any invoice more than
....
Else
....

Data Dictionary

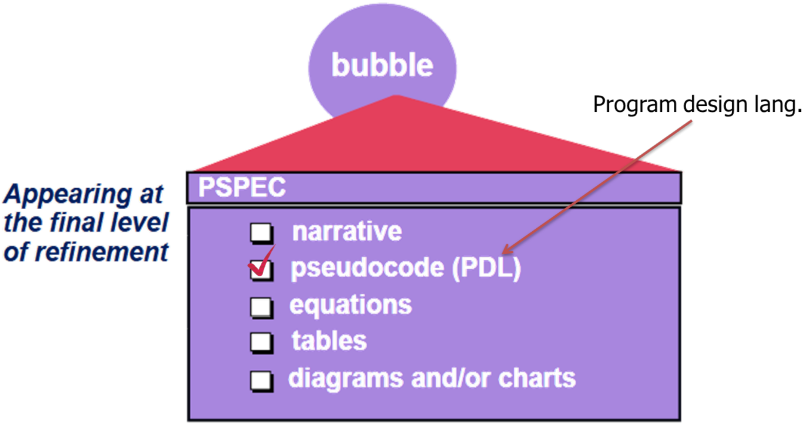
- Definition of all data items appearing in data flow diagram
- Data item = equation representing the configuration of data item



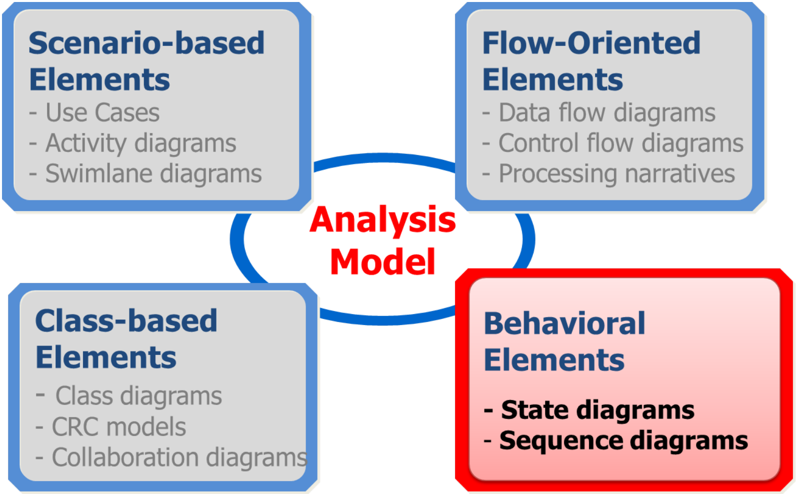
Data Dictionary ⇒ **F = B + C**

Process Specification (PSPEC)

- Specifies all the flow model process in the final level



Elements of Requirements Analysis



Behavioral Modeling

- Behavioral model indicates how software will respond to external **events** or stimuli
- Behavioral model creation 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

What are Events?

Event = a type of observable occurrence

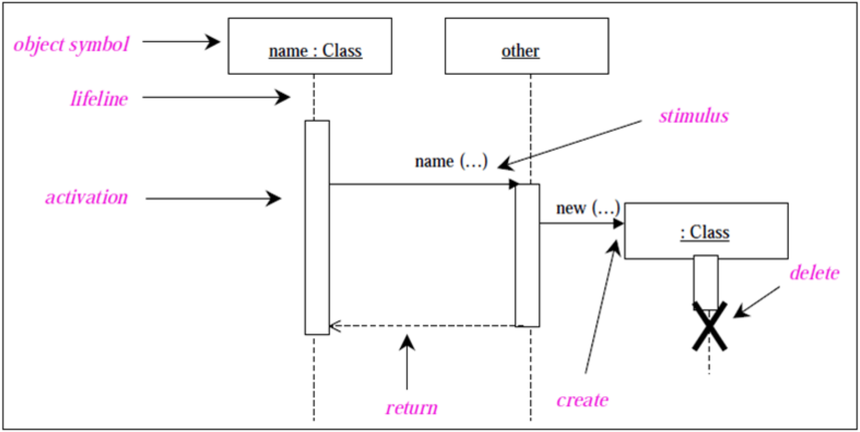
- **interactions** are set of communication between instances
  - synchronous object operation invocation (**call event**)
  - asynchronous signal reception (**signal event**)
  - **creation** and **destruction** of instances
- occurrence of time instants (**time event**)
  - interval expiry
  - calendar/clock time
- change in value of some entity (**change event**)

Behavioral Modeling

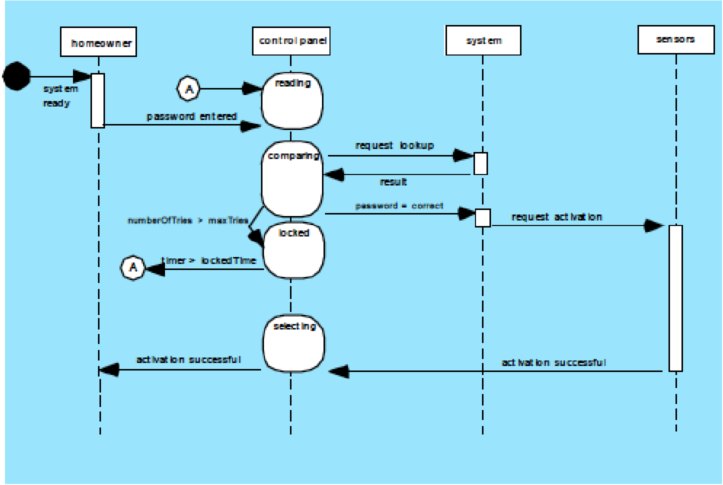
- Two different characterizations of states:
  - each class has states representing the behaviors that changes as the system performs functions
  - system has states to represent behaviors that are observable from the outside
- indicate how the system makes a **transition** from one state to another
  - system state change: indicate event and action
- represent state
  - State diagram
  - Sequence diagram

Sequence Diagram

- illustrates how an event causes a transition from an object to another object

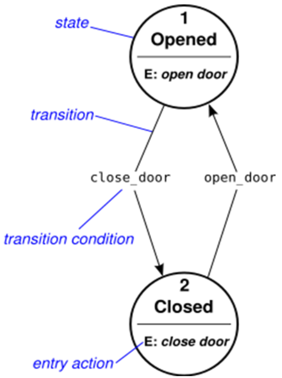


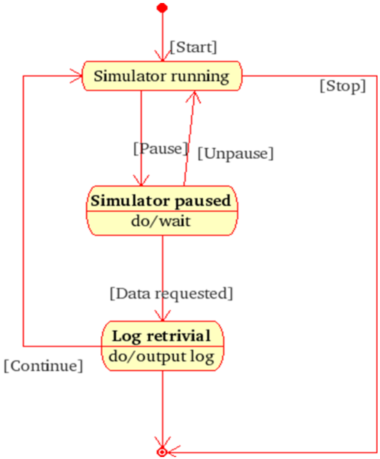
Sequence Diagram Example: SafeHome



State Diagram

- represents events that causes changes between active states in each class





State Diagram Example: SafeHome Control Panel Class