시스템분석

한국항공대학교 소프트웨어학과 지승도교수
R.S. Pressman

요구공학
(Requirements Engineering)

- Understanding the requirements of a problem
- Most difficult tasks that face a software engineer
- Requirement engineering (RE)
  - From Communication activity to Modeling activity
  - Builds a bridge to design and construction

Requirements Engineering Tasks

- Requirement engineering

  Provide the appropriate mechanism for;

  - ✓ Understanding what the customer wants,
  - ✓ Analyzing need,
  - ✓ Assessing feasibility,
  - ✓ Negotiation a reasonable approach,
  - ✓ Specifying the problem unambiguously,
  - ✓ Validating the specification,
  - ✓ Managing the requirements.

Requirements Engineering Process

1. Inception (착수)
2. Elicitation (추출)
3. Elaboration (상세)
4. Negotiation (협상)
5. Specification (명세)
6. Validation (검증)
7. Management (관리)

1. Inception

- How does a software project get started? Is there a single event that becomes the catalyst for a new computer-based system or product, or does the need evolve over times? There are no definitive answers to these questions.
- In some cases, a casual conversation is all that is needed to precipitate a major software engineering effort. But in general, most projects begin when a business need is identified or a potential new market or service is discovered.
- Inception
  - A set of context free question
  - E.g., "Autonomous Drone", "Unmanned Surface Vehicle", "Character-based AI"

Inception: Asking the first Questions (1/3)

- The first set of context-free questions
  - Who is behind the request for this work?
  - Who will use the solution?
  - What will be the economic benefit of a successful solution?
  - Is there another source for the solution that you need?

Inception: Asking the first Questions (2/3)

- Next set of questions – better understanding of the problem
  - How would you characterize "good" output that would be generated by a successful solution?
  - What problem will this solution address?
  - Can you show me the business environment in which the solution will be used?
  - Will special performance issue or constraints affect the way the solution is approached?

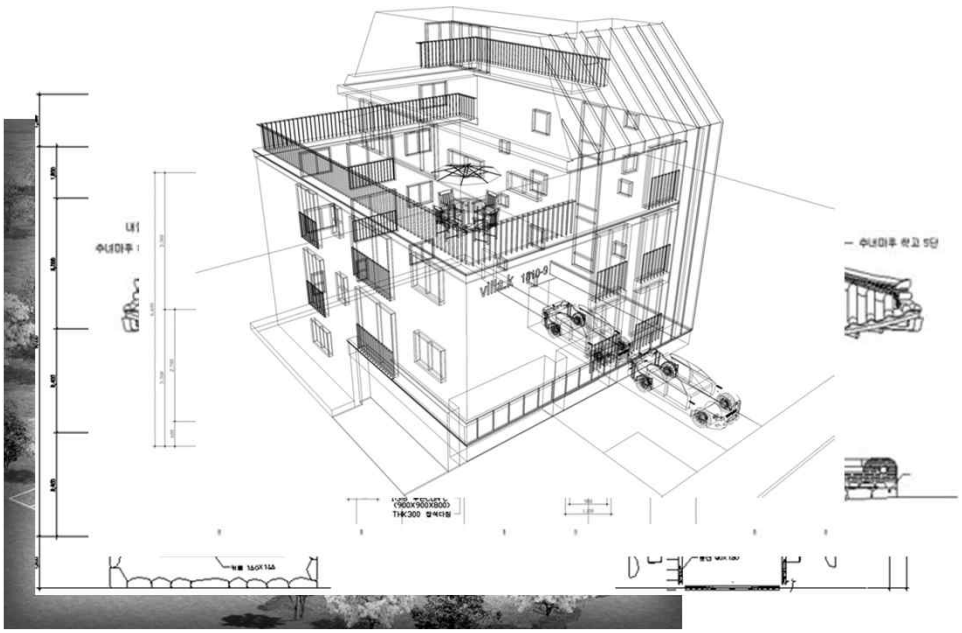
Inception: Asking the first Questions (3/3)

- The final set of questions – effectiveness of communication activity
  - Are you the right person to answer these questions? Are your answers "official"?
  - Are my questions relevant to the problem that you have?
  - Am I asking too many question?
  - Can anyone else provide additional information?
  - Should I be asking you anything else?

2. Elicitation

- It certainly seems simple enough – ask the customer, the users, and others what the objectives for the system or product are, what is to be accomplished, how the system or product fits into the needs of the business, and finally, how the system or product is to be used on a day-to-day basis. But it isn't simple–it's very hard.
- Difficulties of Elicitation
  - Problems of scope: boundary of the system is ill-defined
  - Problems of understanding: the customers/users are not completely sure of what is needed
  - Problem of volatility: the requirements change over time

3. Elaboration



3. Elaboration

- Information with customer is expanded and refined during elaboration
- Elaboration is an analysis modeling action

Scenario-based Models

Use-cases- text
Use-cases diagrams
Activity diagrams
Swimlane diagrams

Flow-oriented Models

Date flow diagrams
Control-flow diagrams
Processing narratives

Behavioral Models

State diagrams
Sequence diagrams

Class-based Models

Class diagrams
Analysis diagrams
CRC models
Collaboration diagrams

Data Models

ERD
Data object table

4. Negotiation (1/2)

- It is also relatively common for different customers or users to propose conflicting requirements, arguing that their version is "essential for our special needs"
- The requirements engineer must reconcile these conflicts through a process of negotiation
- Customers, users, and other stakeholders are asked to rank requirement and then discuss conflicts in priority
- Assess the impact of each requirement in project cost and delivery time

4. Negotiation (2/2)

- Identification of the system or subsystem's key stakeholders.
- Determination of the stakeholders' win condition
- Negotiation of the stakeholders' win conditions to reconcile them into a set of win-win conditions for all concerned.

5. Specification

- Specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype or any combination of these
- "Standard template"
- Consistent, understandable

6. Validation (1/2)

- Quality
- Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected; and that the work products conform to the standards established for the process the project, and the product

6. Validation (2/2)

- Is the requirement consistent with the objectives?
- Have all requirements been specified using the level of abstraction?
- Is the requirement really necessary?
- Is the requirement bounded and unambiguous?
- Does the requirement have attribution?
- Do any requirement conflict with other requirement?
- Is each requirement achievable in the technical environment?
- Is each requirement testable once implemented?
- Does the requirement model properly reflect the information, function and behavior of the system?
- Has requirement model "partitioned"?
- Have requirement pattern used?

7. Requirement Management

- A set of activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project proceeds.

시스템분석가의 자격

- 추상적, 논리적 개념화 능력
- 애매모호하거나 모순된 것들로부터 정확한 사실을 이끌어 낼 능력
- 고객의 환경을 이해할 능력
- H/W, S/W 요소를 고객의 환경에 적용시킬 능력
- 말하고 쓰기를 통한 대화 능력
- To see the forest for the tree

Requirements Analysis

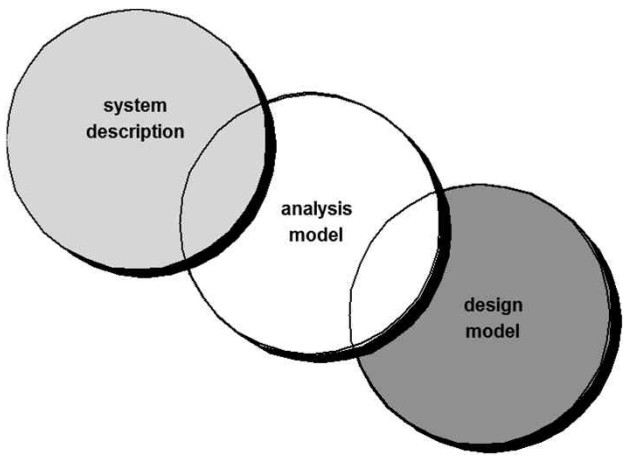
- Specification of operation characteristics, interface, constraints
- Allows the S/W engineer to elaborate on basic requirements established during earlier requirement engineering tasks and build models that depict user scenarios, function activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed.
- Provides the S/W designer with information, function, and behavior for architectural, interface, and component-level design
- Provides the developer and customer with the means to assess quality
- Focus is on "*What*" not "*how*" ("Problem" not "Solution")

Requirements Analysis

- Overall objectives and philosophy

  1. To describe what the customer requires
  2. To establish a basic for the creation of a software design
  3. To define a set of requirements that can be validated once the software is built

A Bridge



Analysis Modeling Rules

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

Domain Analysis

Software domain analysis is 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 is] 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

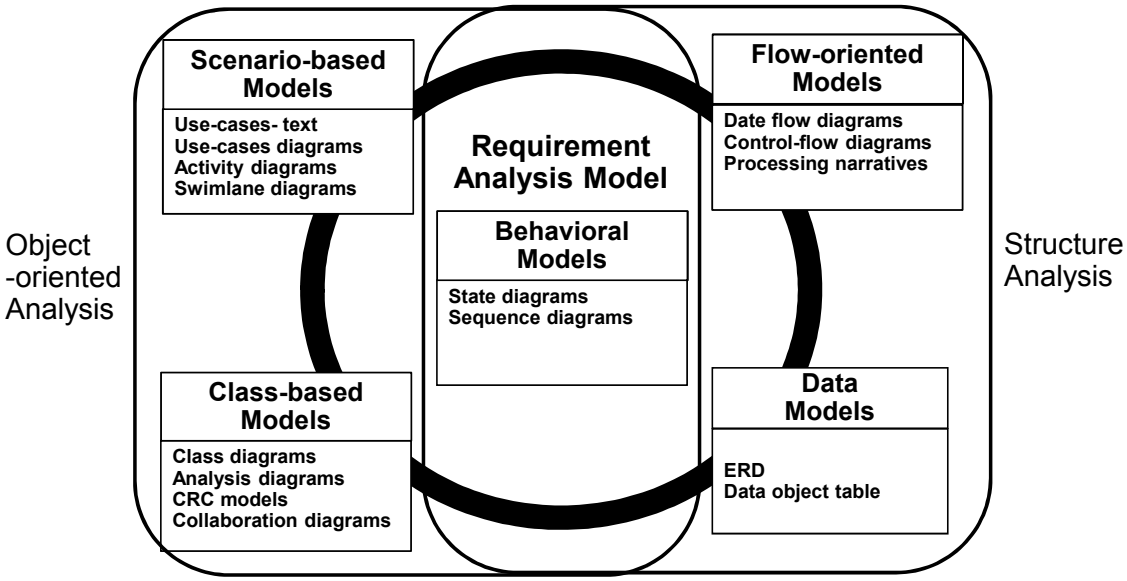
Requirement Modeling Approaches

- Structured analysis
  - Data and the processes that transform the data as separate entities.
  - Data object 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
  - Focus on the definition of classes and the manner in which they collaborate with one another to effect customer requirement.
  - UML and the Unified Process (UP)

Object-Oriented Analysis

- Object-oriented analysis (OOA) is to define all classes that are relevant to the problem to be solved
  1. Basic user requirements must be communicated between the customer and the software engineer
  2. Classes must be identified
  3. A class hierarchy is defined
  4. Object-to-object relationships
  5. Object behavior must be modeled
  6. Tasks 1 through 5 are reapplied iteratively until the model is complete

Element of the Analysis Model



Scenario-based Modeling

Scenario-based Models

Use-cases- text (Template)
Use-cases diagrams
Activity diagrams
Swimlane diagrams

Scenario-Based Modeling

"[Use-cases] are simply an aid to defining what exists outside the system (actors) and what should be performed by the system."

by Ivan Jacobson

Developing Use-Cases

- "a use-case captures a contract .... [that] describes system's behavior under various condition as the system responds to a request from one of its stakeholders."
- Use-case tells a stylized story about how an end-user interacts with the system under a specific set of circumstances

Developing Use-Cases

- Who is the primary actor, the secondary actor?
- What are the actor's goals?
- What preconditions should exist before the story begin?
- What main tasks or functions are performed by the actor?
- What exception might be considered as the story is described?
- What variations in the actor's interaction are possible?
- What system information will the actor have to inform the system about changes in the external environment?
- What information does the actor wish to be informed about unexpected changes?

Home Security Example

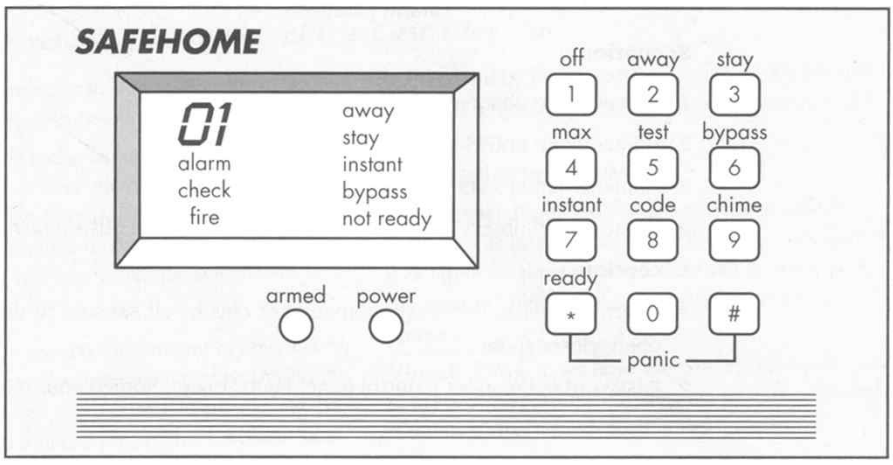
Home IOT management systems market is growing at a rate of 40 percent per year. The first Safehome function we bring to market should be the home security function. Most people are familiar with "alarm system" so this would be an easy sell.

The home security function would protect against and/or recognize a variety of undesirable "situation" such as illegal entry, fire, flooding, carbon monoxide levels, and other. It'll use our wireless sensors to detect each situation, can be programmed by the homeowner, and will automatically telephone a monitoring agency when a situation is detected.

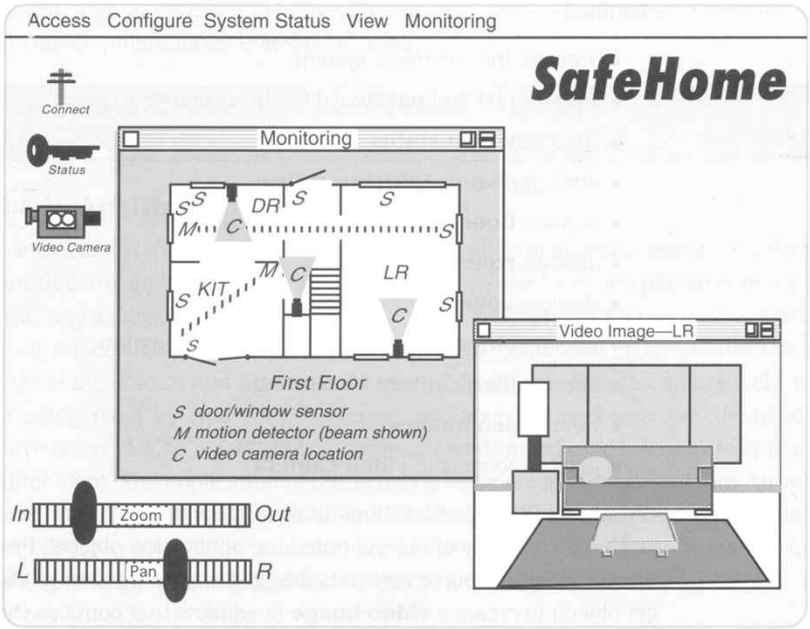
Mini-spec of Control Panel

The control panel is a wall-mounted unit that is approximately 9 X5 inches in size. The control panel has wireless connectively to sensors and a PC. User interaction occurs through a keypad containing 12 keys. A 2 X2 inch LCD display provides user feedback. Software provides interactive prompts, echo, and similar function.

SafeHome Example



SafeHome Example



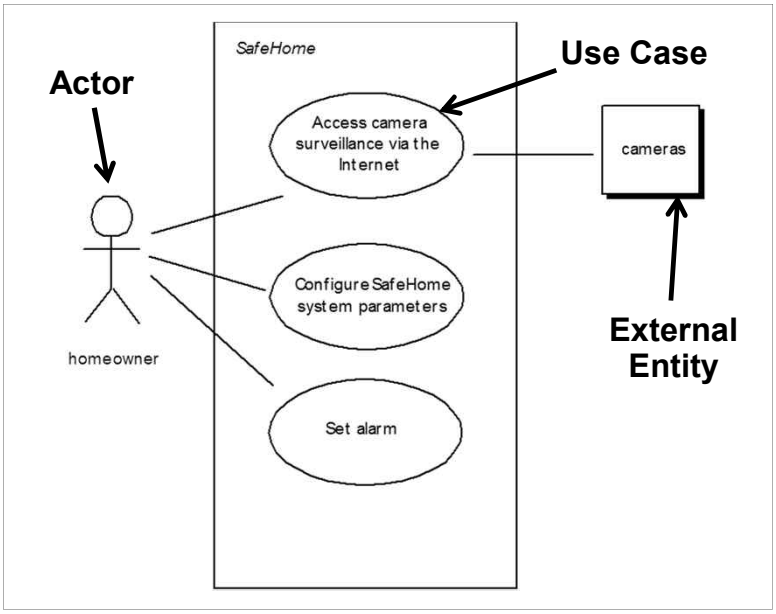
Scenario-Based Modeling

- Writing Use-Cases
  - Captures the interactions that occur between producers and consumers of information and the system itself
  - The SafeHome example:
    - Access camera surveillance via the Internet
    - Select camera to view
    - Request thumbnails from all cameras
    - Display camera views in a PC window
    - Selectively record camera output
    - Replay camera output
  - Sequential presentation does not consider any alternative interactions
  - Use-cases of this type are sometimes referred to as primary scenarios
    → so we also need secondly scenario (alternative behavior)

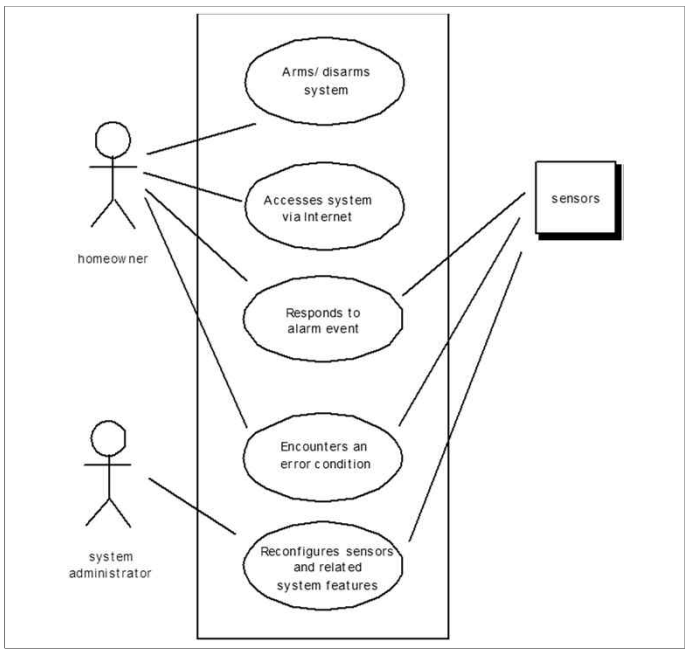
Use-case Template Example

- *Use-Case:* Access camera surveillance – display camera views (ACS-DVC)
- *Prime actor:* Homeowner
- *Goal:* To view output of camera placed throughout the house from any remote location via the Internet.
- *Precondition:* System must be fully configured; appropriate user ID and passwords must be obtained.
- *Trigger:* The homeowner decides to take a look inside the house while away.
- *Scenario:*
  1. The homeowner logs onto the SafeHome Web site
  2. The homeowner enters his or her user ID
  3. The homeowner enters two passwords
  4. The system displays all major function buttons
  5. The homeowner selects "surveillance" from the major function buttons
  6. The homeowner selects "pick a camera"
  7. …….
- *Exceptions*, *Priority*, *When available*, *Frequency of use*, etc.

Use-Case Diagram



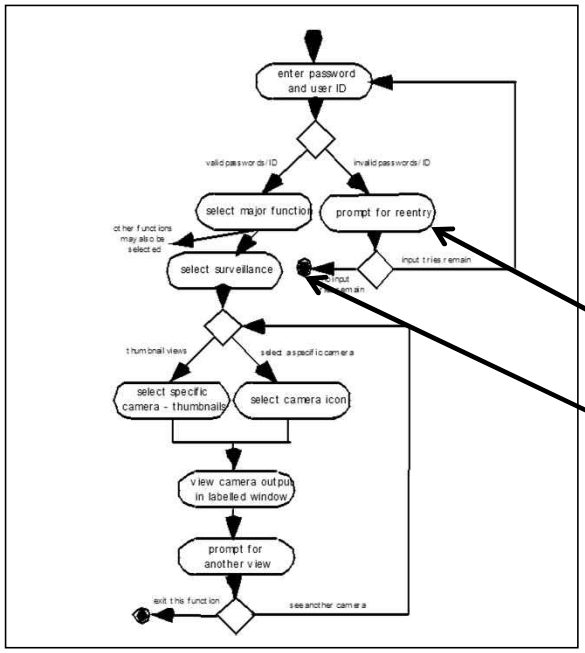
Use-Case Diagram



UML Models

- Developing an Activity Diagram
  - The UML activity diagram supplements the use-case by providing a graphical representation of interaction within a specific scenario.

Activity Diagram

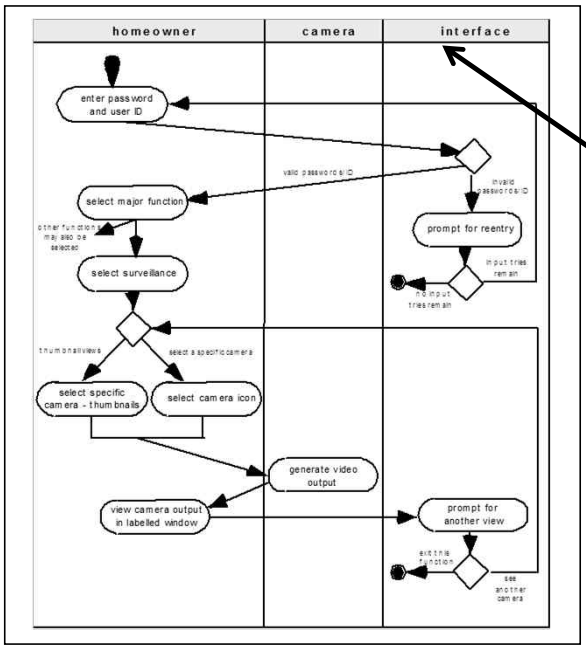


UML Models

- Swimlane Diagrams

The UML swimlane diagram is a useful variation of the activity diagram and allows the modeler to represent the flow of activity described by the use-case and at the same time indicate which actor or analysis class has responsibility for the action described by an activity rectangle.

Swimlane Diagrams



Class Object

Class-based Modeling

Class-based Models

Class diagrams
Analysis diagrams
CRC models
Collaboration diagrams

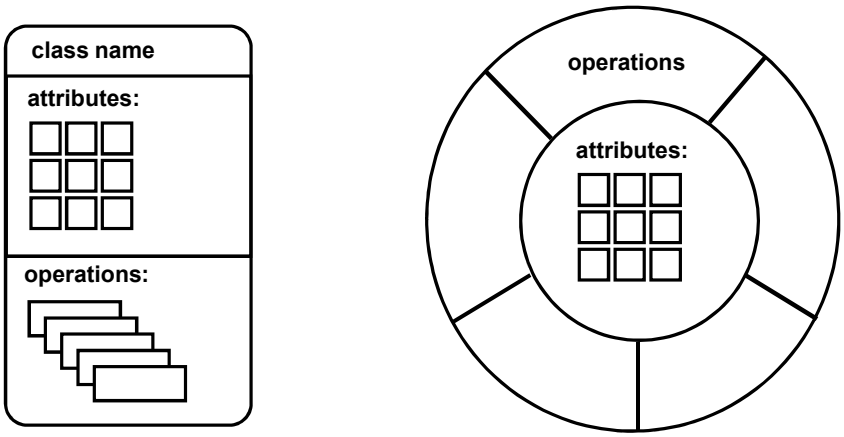
Class-Based Modeling

- Identifying analysis classes
- Specifying the attributes
- Defining operations
- Modeling CRC (Class-Responsibility-Collaborator)

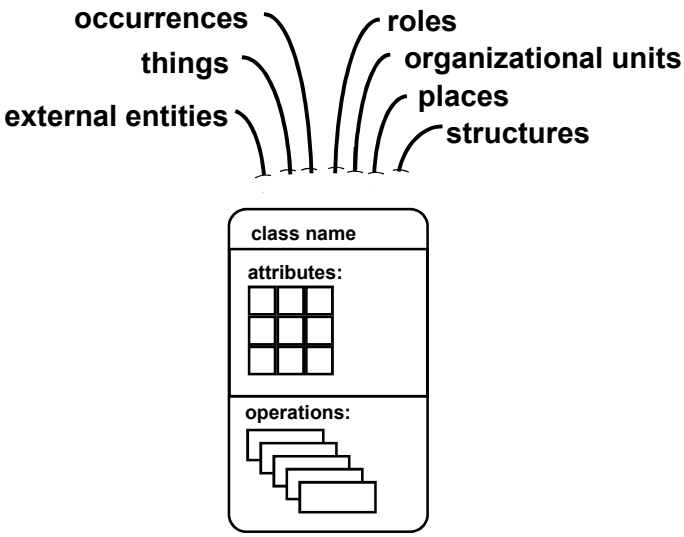
Class

- object-oriented thinking begins with the definition of a class, often defined as:
  - template
  - generalized description
  - "blueprint" ... describing a collection of similar items
- a meta-class (also called a superclass) establishes a hierarchy of classes
- once a class of items is defined, a specific instance of the class can be identified

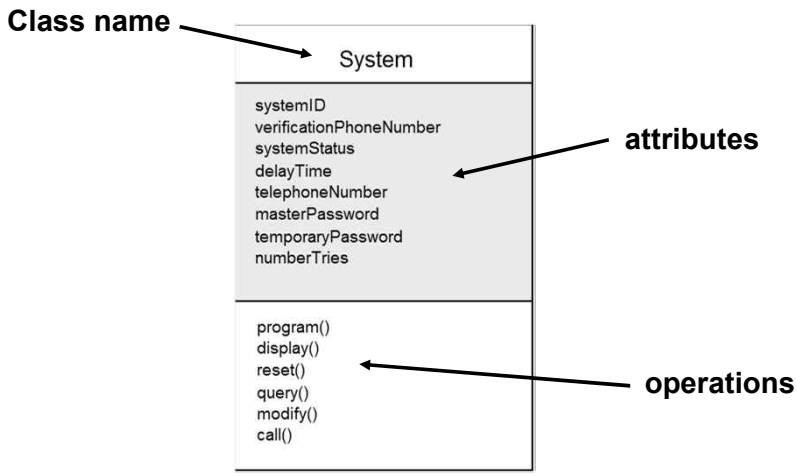
Building a Class



What is a Class?

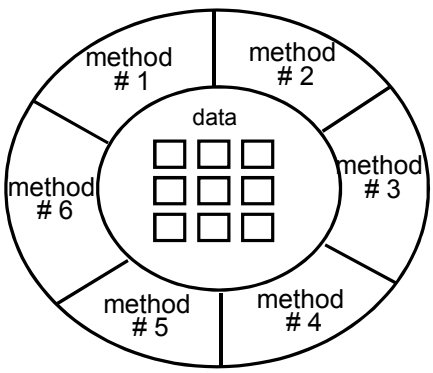


Class Diagram

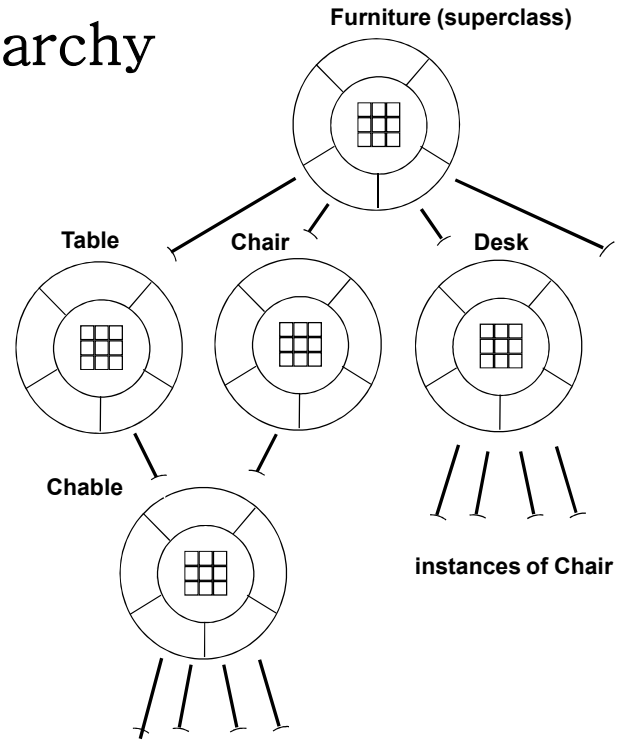


Encapsulation / Information Hiding

- The object encapsulates both data and the logical procedures required to manipulate the data.

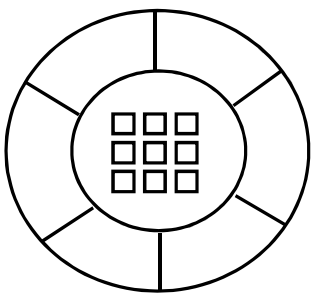


Class Hierarchy

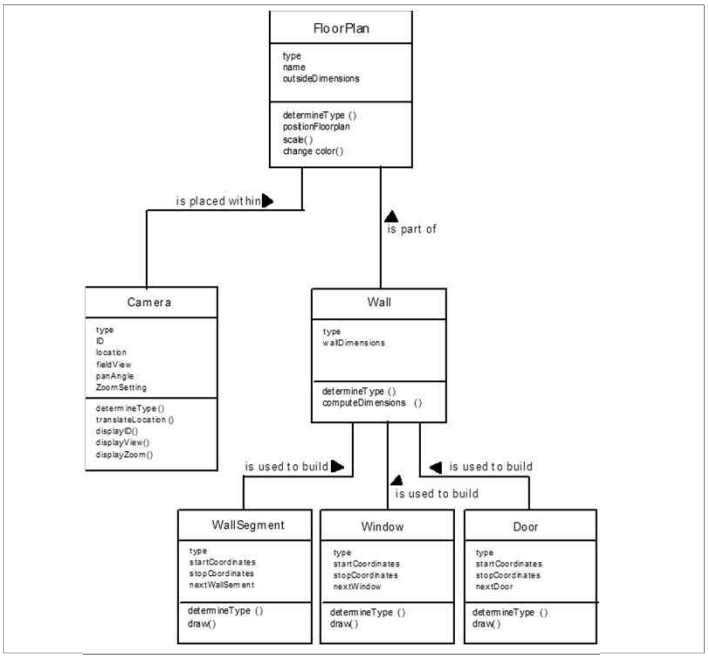


Methods (Operations, Services)

- An executable procedure encapsulated in a class and designed to operate on one or more data attributes.
- A method is invoked via message passing.



Class Diagram



CRC Modeling

- Class-Responsibility-Collaborator (CRC) Modeling provide a simple means for identifying and organizing the classes that are relevant to system or product requirement.

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

Classes

- *Entity classes*, also called *model* or *business* classes, are extracted directly from the statement of the problem (e.g., FloorPlan and Sensor).
- *Boundary classes* are used to create the interface (e.g., interactive screen or printed reports) that the user sees and interacts with as the software is used.
- *Controller classes* manage a "unit of work" from start to finish. Controller classes can be designed to manage.
  - the creation or update of entity objects;
  - the 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.

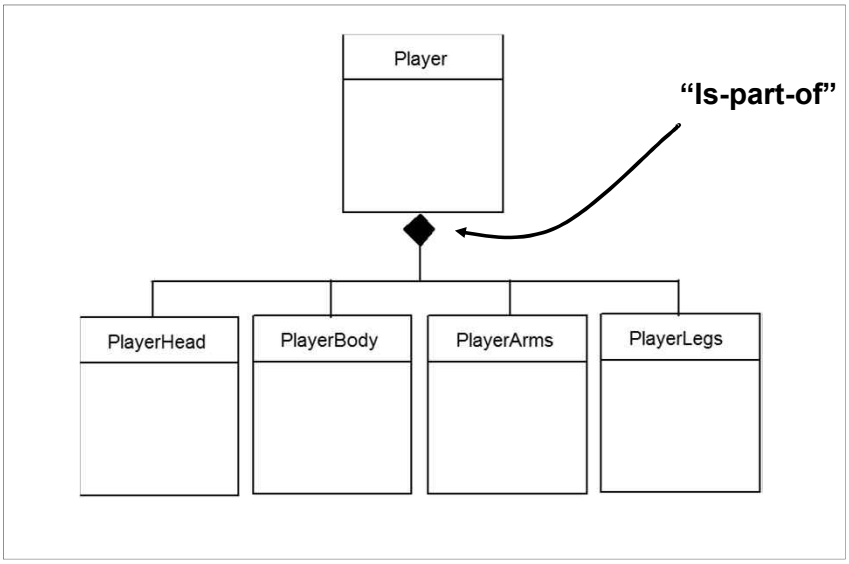
Responsibilities

- System intelligence should be distributed across classes to best address the needs of the problem
- Each responsibility should be stated as generally as possible
- Information and the behavior related to it should reside within the same class
- Information about one thing should be localized with a single class, not distributed across multiple classes.
- Responsibilities should be shared among related classes, when appropriate.

Collaborations

- Classes fulfill their responsibilities in one of two ways:
  1) A class can use its own operations to manipulate its own attributes.
  2) A class can collaborate with other classes.
- Collaborations identify relationships between classes
- To help in the identification of collaborators, the analysis can examine three different generic relationships between classes:
  1) the *is-part-of* relationship (aggregate)
  2) the *has-knowledge-of* relationship (associate)
  3) the *depends-upon* relationship (dependency)

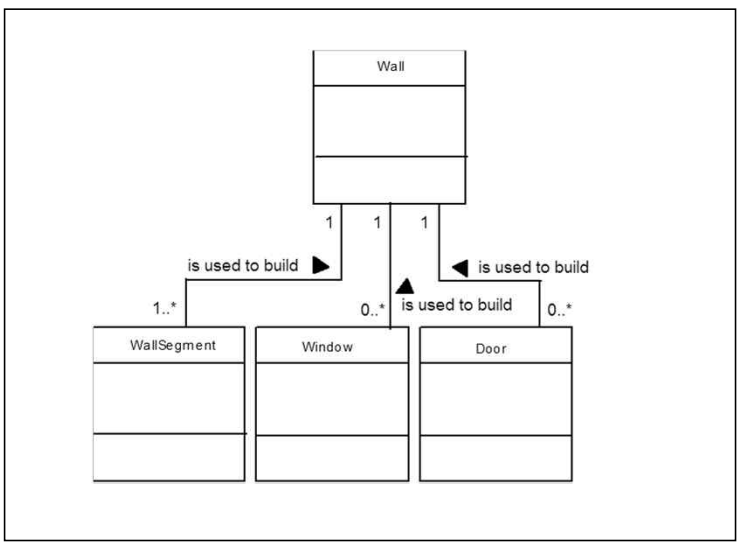
Aggregation (*is-part-of*)



Associations

- Two analysis classes are often related to one another in some fashion
  - In UML these relationships are called *associations*
  - Associations can be refined by indicating *multiplicity* (the term *cardinality* is used in data modeling)

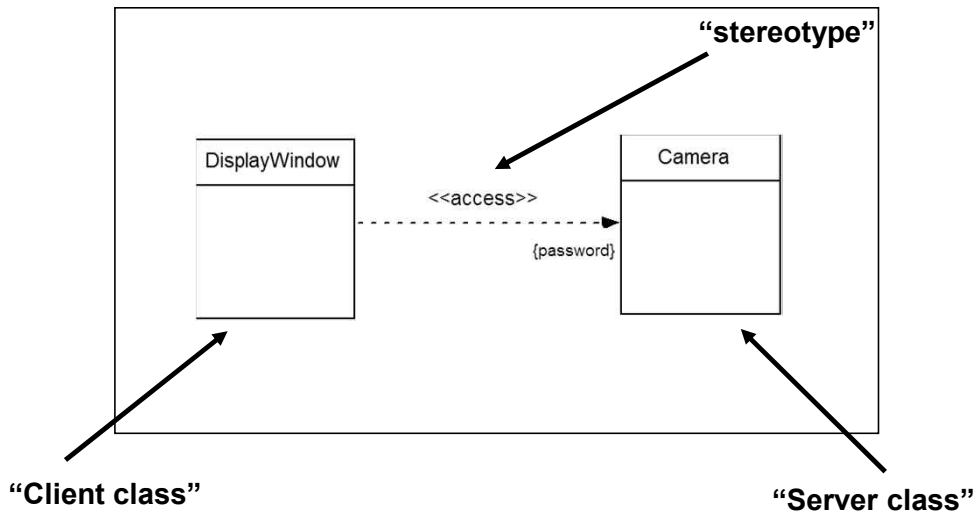
Association (Multiplicity)
has-knowledge-of



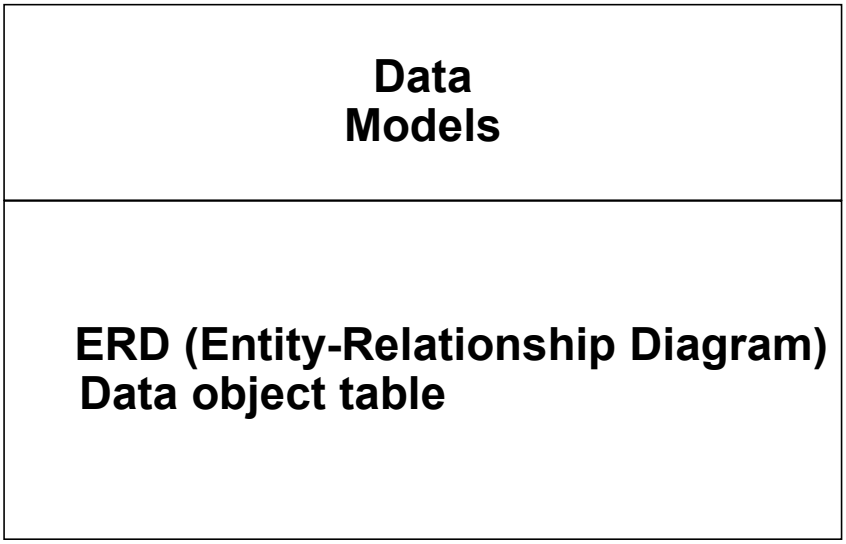
Dependencies

- In many instances, a client-server relationship exists between two analysis classes.
  - In such cases, a client-class depends on the server-class in some way and a *dependency relationship* is established

Dependencies (client-server)



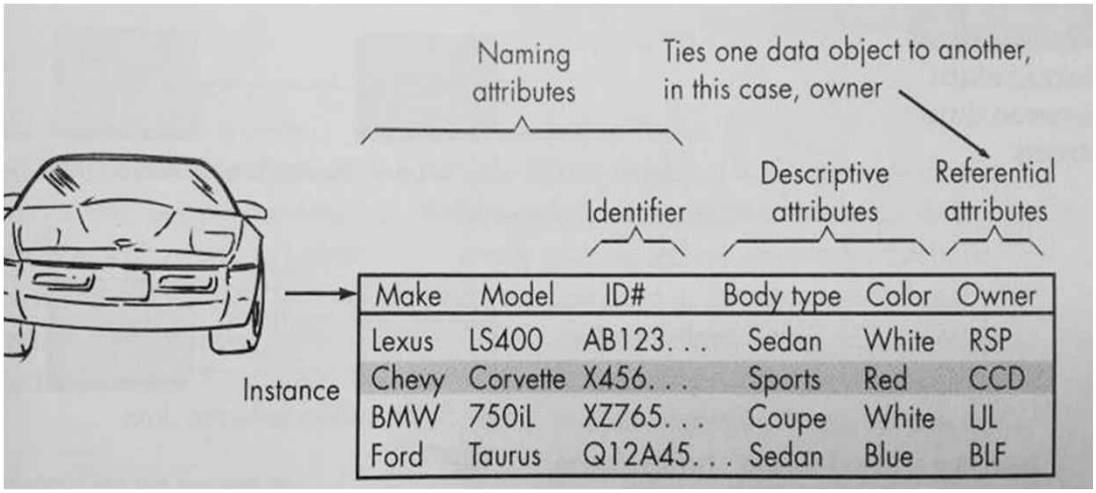
Data Modeling



Data Modeling Concept

- Data object
  - Representation of almost any composite information that must be understood by software
  - By Composite information, we mean something that has a number of different properties or attributes
  - Data object can be;
    - An Entity
    - A thing
    - An Occurrence
    - A Event
    - A Role
    - An Organizational unit
    - A Place
    - A structure
  - Data object encapsulates data only (different with OO paradigm)

Data Objects and Attributes



Make	Model	ID#	Body type	Color	Owner
Lexus	LS400	AB123. . .	Sedan	White	RSP
Chevy	Corvette	X456. . .	Sports	Red	CCD
BMW	750iL	XZ765. . .	Coupe	White	LJL
Ford	Taurus	Q12A45. . .	Sedan	Blue	BLF

Data Objects and Attributes

A data object contains a set of attributes that act as an aspect, quality, characteristic, or descriptor of the object

object: automobile

attributes:
- **make**
- **model**
- **body type**
- **price**
- **options code**

← **Data Object Table**

Data Modeling Concept

- Data attributes
  - Define the properties of data object

1. Name an instance of the data object
2. Describe the instance
3. Make reference to another instance in another table

  - Identifier – that is, the identifier attribute becomes a “key” when we want to find an instance of the data object

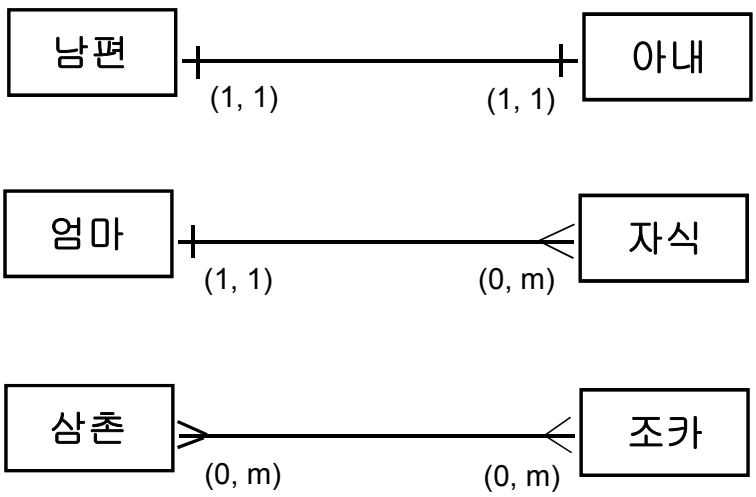
Data Modeling Concept

- Relationships
  - Data objects are connected to one another in different ways
  - Object/Relationship pairs
    - A person owns a car
    - A person is insured to drive a car

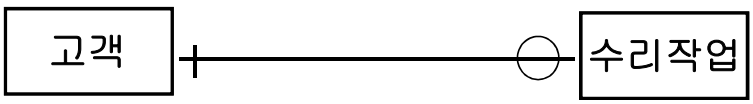
Data Modeling Concept

- Cardinality and Modality
  - Cardinality
    - The data model must be capable of representing the number of occurrences of objects in a given relationship
  - Modality
    - A relationship of a relationship is 0 if there no explicit need for the relationship to occur or the relationship is optional
    - The modality is 1 if an occurrence of the relationship is mandatory

ERD Notation: cardinality example

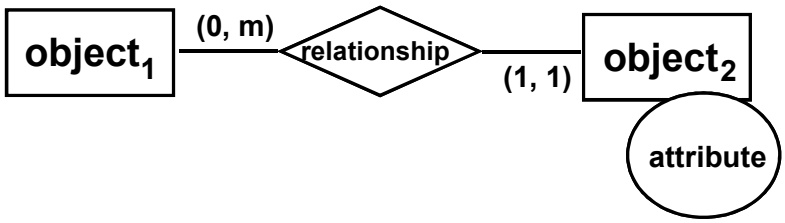


ERD Notation: modality example

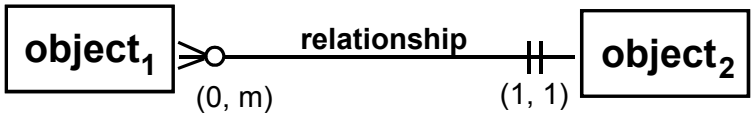


Entity-Relationship Diagram (ERD)

One common form:



Another common form:



The ERD: An Example