Chapter : Analysis Modeling

Requirements Analysis

- Requirements analysis
  - Specifies software’s operational characteristics
  - Indicates software's interface with other system elements
  - Establishes constraints that software must meet
- Requirements analysis allows the software engineer (called an *analyst* or *modeler* in this role) 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, and the flow of data as it is transformed.
- Throughout analysis modeling, the SE’s primary focus is on what not on how.
- Analysis model and the requirements specification provide the developer and the customer with means to assess quality once software is built.

Analysis Modeling Principles

- Analysis methods are related by a set of operational principles:

1. *The information domain of a problem must be represented and understood.*
2. *The functions that are software performs must be defined.*
3. *The behavior of the software must be represented.*
4. *The models that depict information, function and behavior must be partitioned in a manner that uncovers detail in a layered fashion.*
5. *The analysis task should move from essential information toward implementation detail.*

Analysis Modeling Principles

1. *Information domain* encompasses that the data flow into the system, out of the system and data stored.
2. *Functions* provide direct benefit to end-users and also provide internal support for those features that are user visible.
3. *Behavior* driven by its interaction with the external environment.

   E.g. Input provided by end-users, control data provided by an external system, or monitoring data.

Analysis Modeling Principles

4. Key strategy of analysis model, divide complex problem into sub-problem until each sub-problem is relatively easy to understood. This concept is called *partitioning*.
5. The "essence" of the problem is described without any consideration of how a solution will be implemented.

   E.g. Video game requires that player "instruct"

Implementation detail (design model) indicates how the essence will be implemented

E.g. Keyboard command might be typed or a joystick used.

Analysis Model Objectives

Three Primary Objectives:

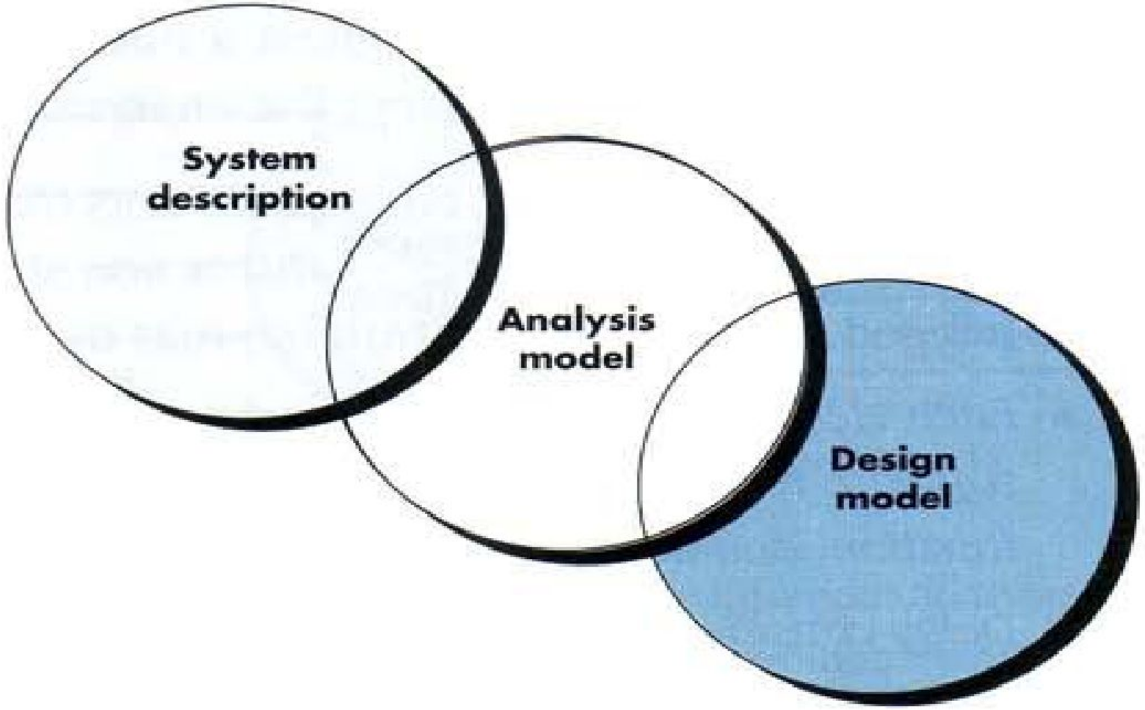
- Describe what the customer requires.
- Establish a basis for the creation of a software design.
- Devise a set of requirements that can be validated once the software is built.

- Its bridges the gap between a system-level description that describes overall system functionality and a software design.

Guidelines :

- Graphics should be used whenever possible.
- Differentiate between the logical (essential) and physical (implementation) considerations.
- Develop a way to track and evaluate user interfaces.

Analysis Model - A Bridge



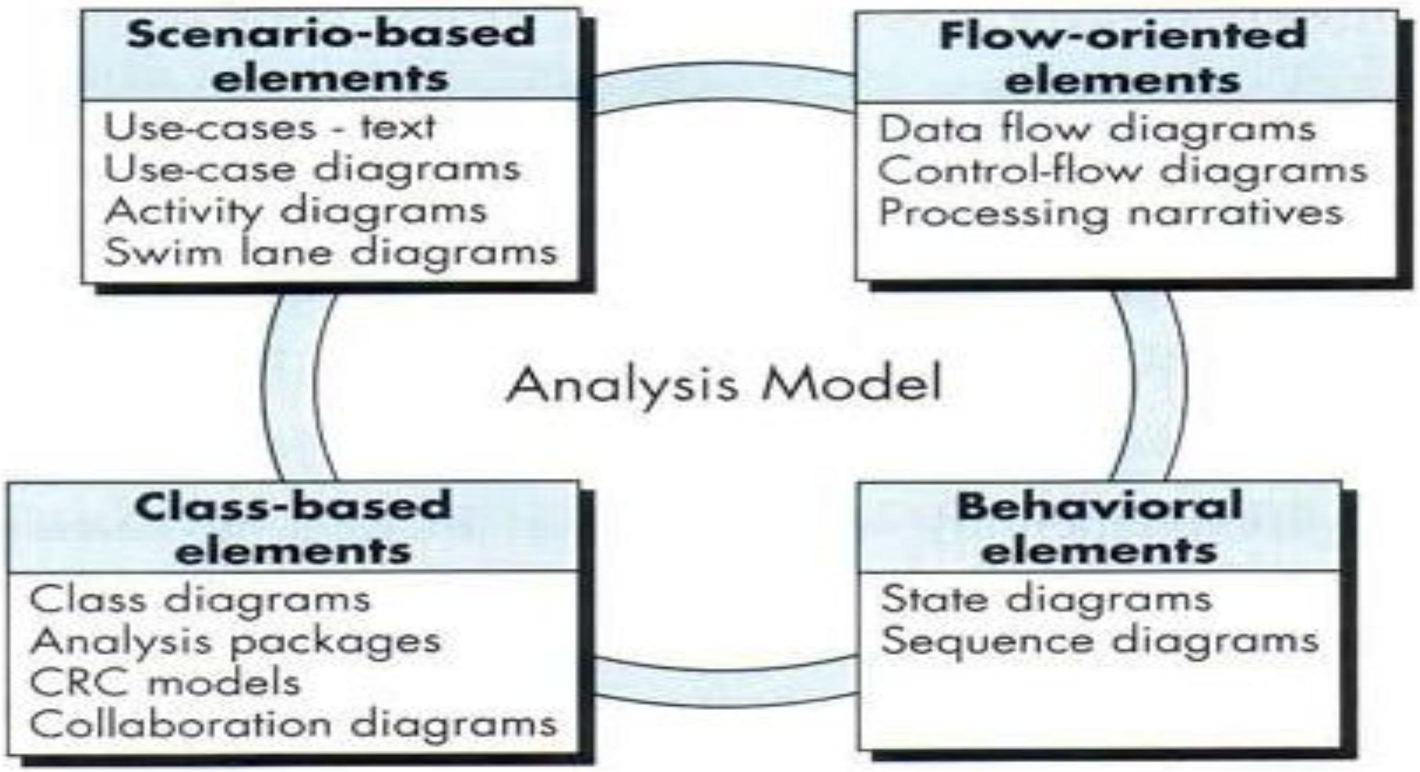
Analysis Rules of Thumb

- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high. – Don’t get bogged into the details.
- Each element of the analysis model should add to an overall understanding of software requirements and provide insight into the information domain, function and behavior of the system.
- Delay consideration of infrastructure and other non-functional models until design.
- Minimize coupling throughout the system. - If level of interconnectedness is high, efforts should be made to reduce it.
- Assured that the analysis model provides value to all stakeholders. – business stakeholder should validate requirement, Designers should use the model as a basis for design.
- Keep the model as simple as it can be. - No need to use additional diagram and use notations.

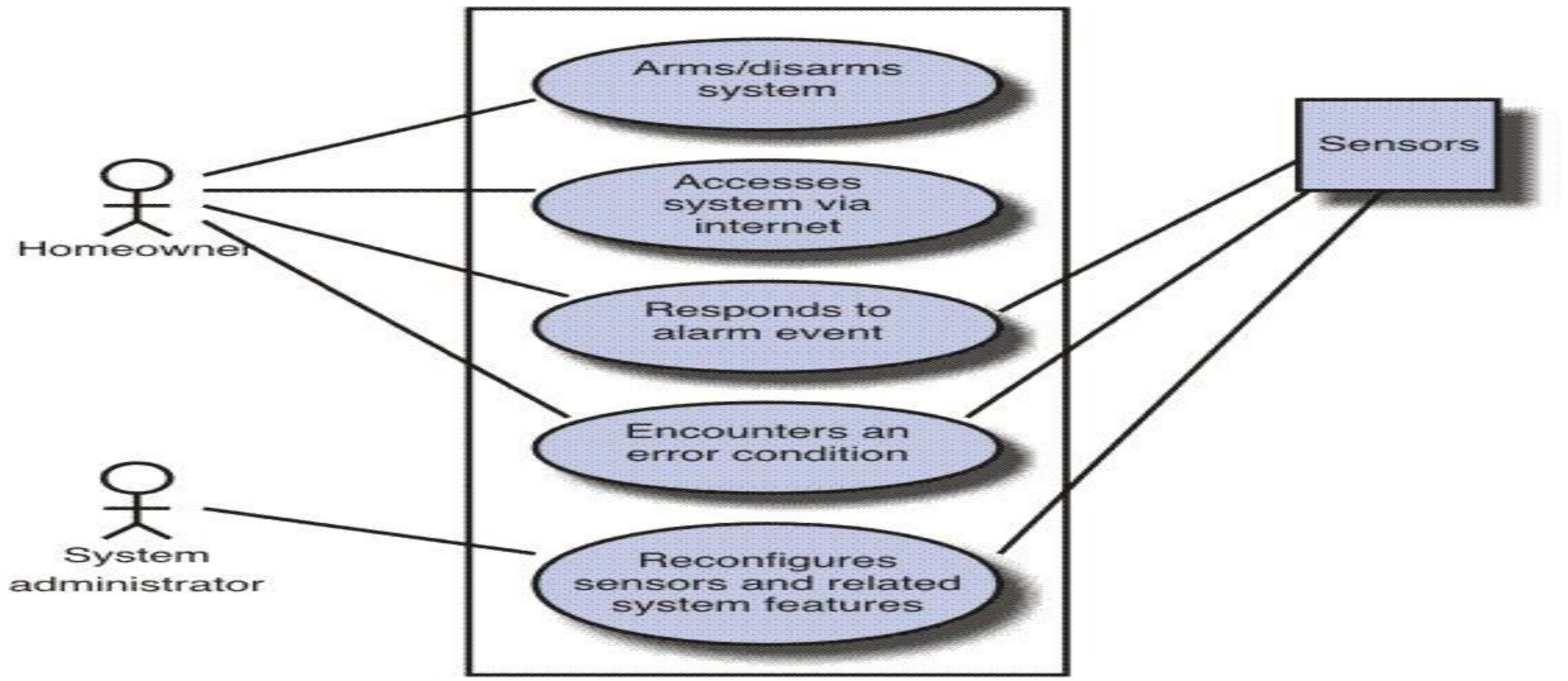
Elements of Analysis model

- There are two approaches
  1. Structured Analysis:-
     - Data objects are modeled in a way that defines their attributes and relationships.
     - Processes that manipulate data objects in a manner that shows how they transform data as data objects flow through the systems.
  2. Object Oriented Analysis :-
     - Focuses on the definition of classes and the manner in which they collaborate with one another.
     - UML is predominantly object oriented.

Elements of Analysis model

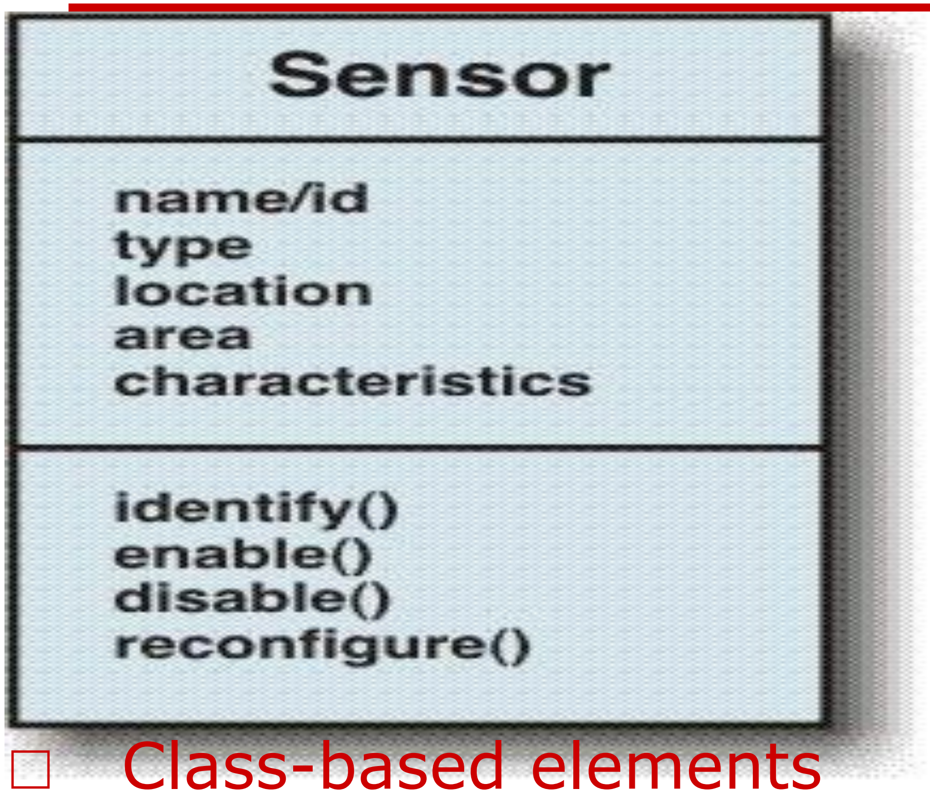


Scenario Based diagram


Arms/disarms system
Accesses system via internet
Responds to alarm event
Encounters an error condition
Reconfigures sensors and related system features
Homeowner
System administrator
Sensors

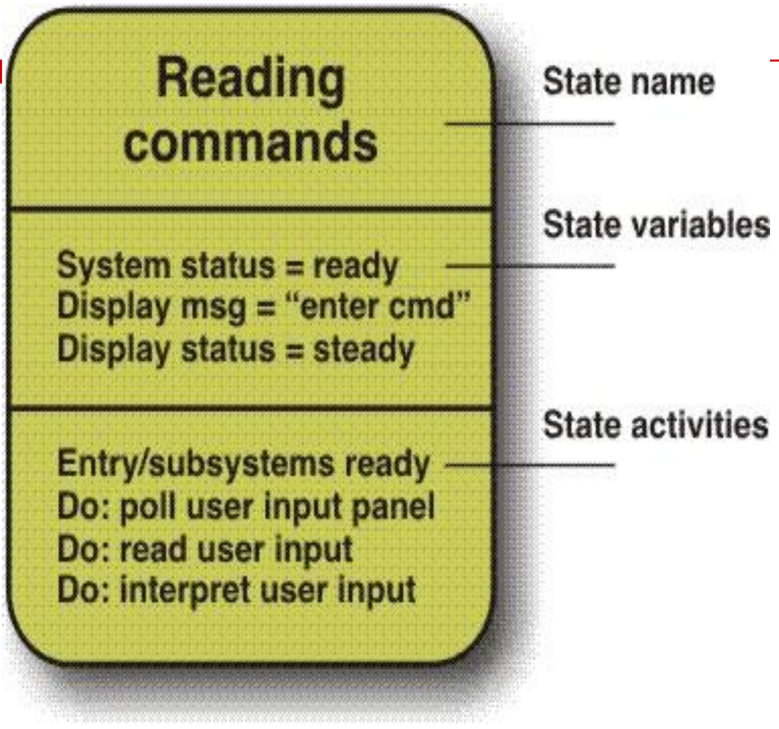
- Scenario-based elements
  - Use-case—How external actors interact with the system (use-case diagrams; detailed templates)
  - Functional—How software functions are processed in the system (flow charts; activity diagrams)
  - Activity – can be represented at many diff. level of abstraction.

Class diagram for sensor



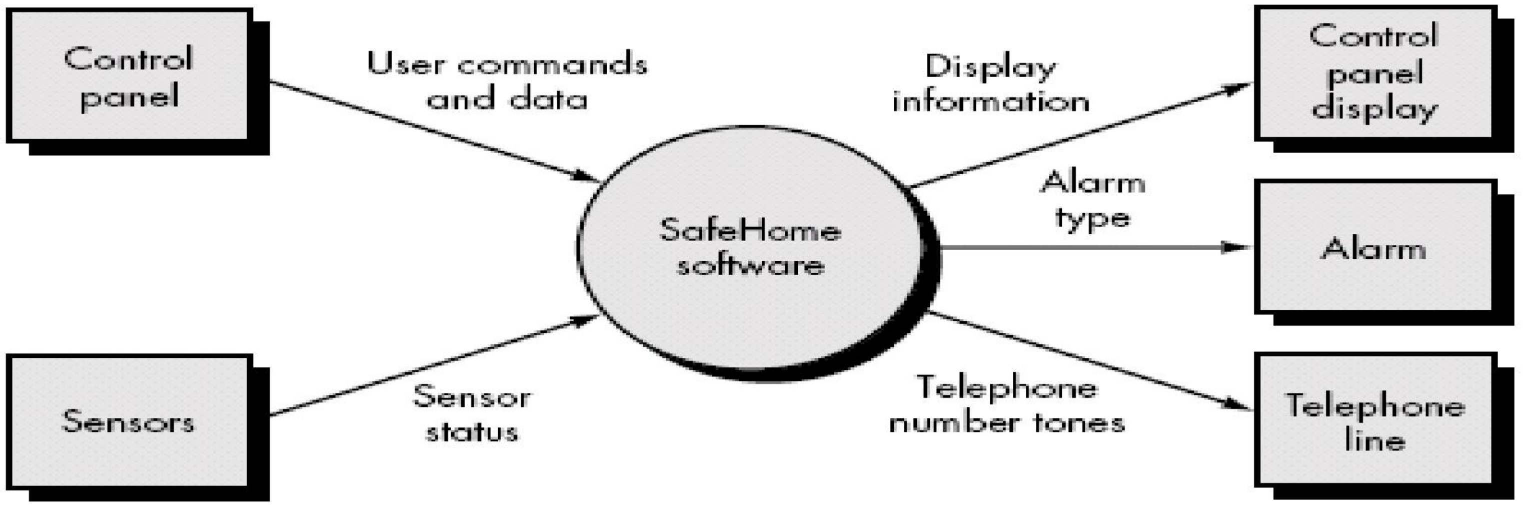
- Class-based elements
  - The various system objects (obtained from scenarios) including their attributes and functions (class diagram)

Behavioral Element – State Diagram



- Behavioral elements
  - How the system behaves in response to different events (state diagram)

Flow-oriented elements



- Flow-oriented elements
  - How information is transformed as if flows through the system (data flow diagram)
  - System accepts input in a variety forms; applies functions to transform it; and produces output in variety forms.

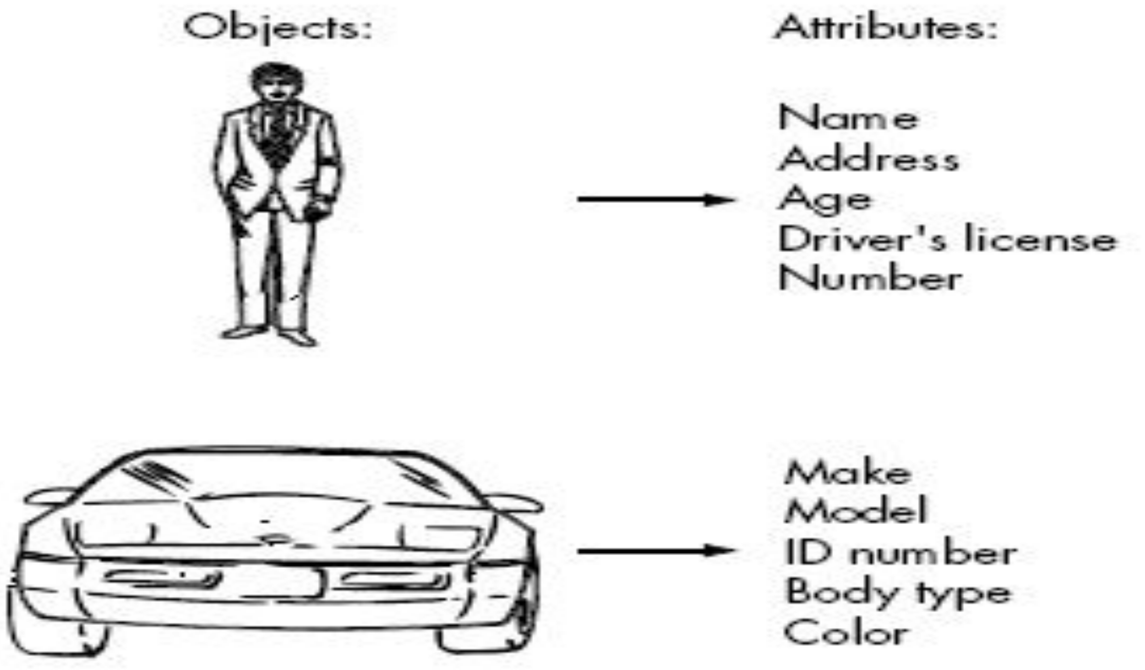
Data modeling

- Analysis model often begin with data modeling.
- Data model consists of three interrelated pieces of information:
  - The data object,
  - The attributes that describe the data object, and
  - The relationships that connect data objects to one another.

Data Object

- A *data object* is a representation of almost any composite information that must be understood by software.
- *composite information* - number of different <u>properties</u> or <u>attributes.</u>
- A data object can be:-
  - An external entity (e.g., anything that produces or consumes information),
  - A thing (e.g., a report or a display),
  - An occurrence (e.g., a telephone call)
  - Event (e.g., an alarm),
  - A role (e.g., salesperson),
  - An organizational unit (e.g., accounting department),
  - A place (e.g., a warehouse),
  - A structure (e.g., a file).

For Ex.- Set of attributes can be defined for a person or a car (i.e. Data Object)



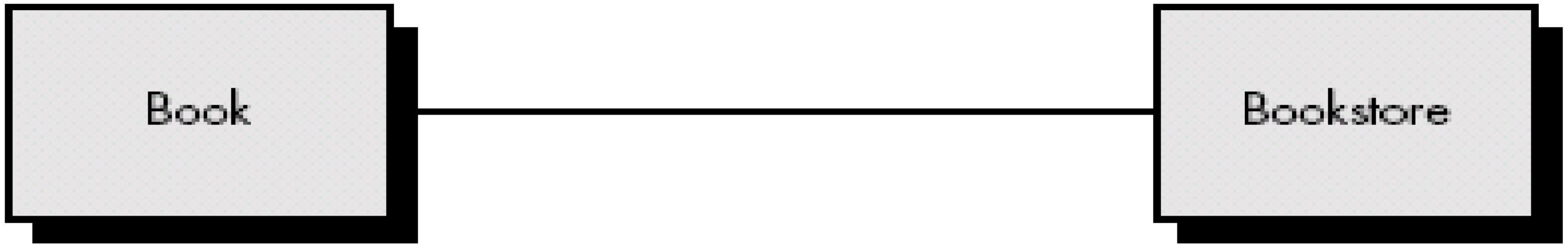
Data Attributes

- Define the properties of a data object.
- Attributes <u>name a data object</u>, <u>describe its characteristics</u>, and in some cases, <u>make reference to another object.</u>
- In addition, one or more of the attributes must be defined as an <u>*identifier(Key value or Unique value).*</u>

*Ex. Data object **Car** has Id number as identifier.*

Relationships

- Data objects are connected to one another in different ways.
- Consider two data objects
  - **Book**
  - **Bookstore**
- A connection is established between **book** and **bookstore** because the two objects are related.

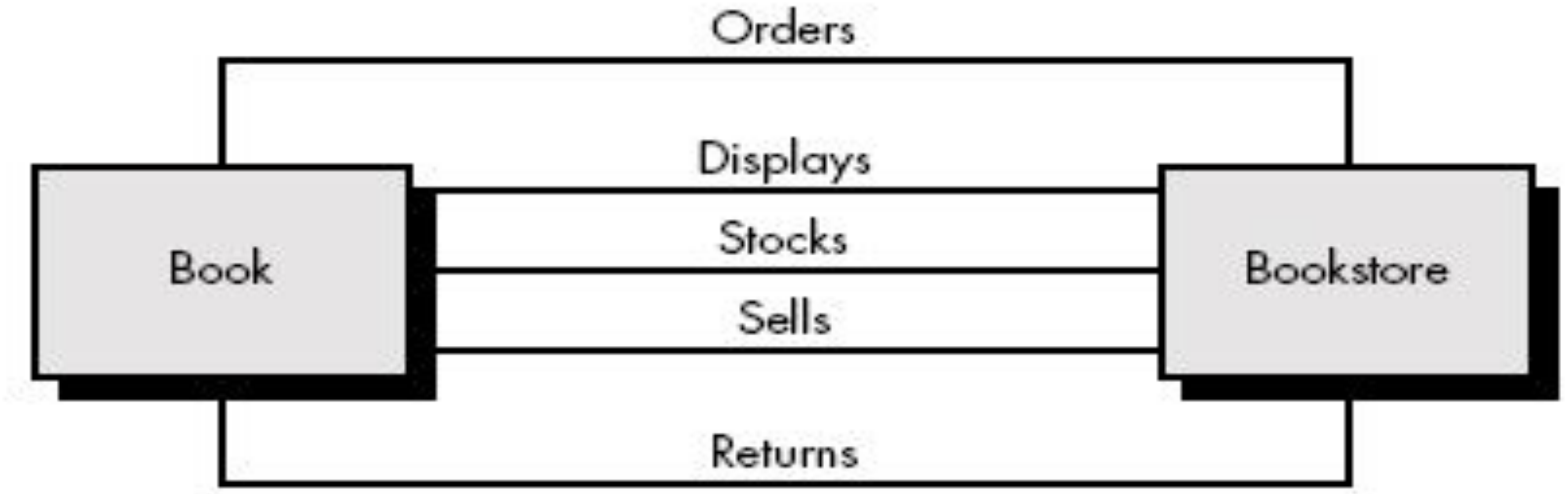


Relationship

- To determine relationship between them, must understand the role of book and bookstore.
- Can define a set of object/relationship pairs that define the relevant relationships.

For Example:

- A bookstore orders books.
- A bookstore displays books.
- A bookstore stocks books.
- A bookstore sells books.
- A bookstore returns books.



Cardinality and Modality

- Additional element of data modeling.
- Object X *relates to object Y does not provide* enough information.
- How many occurrences of object X are related to how many occurrences of object Y called **<u>cardinality.</u>**

Cardinality

- Representing the number of occurrences objects in a given relationship.
- Cardinality is the specification of the number of occurrences of one [object] that can be related to the number of occurrences of another.
- Cardinality is usually expressed as simply 'one' or 'many'.
- 1:1 – One object can relate to only one other object.
- 1:M – one object can relate to many objects.
- M:N – Some no. of occurrences of an object can relate to some other no. of occurrences of another object.

Modality

- Cardinality does not provide an indication of whether or not a particular data object must participate in the relationship.
- Modality of a relationship is 0 if there is no explicit need for the relationship to occur or the relationship is optional.
- The modality is 1 if an occurrences of the relationship is mandatory.

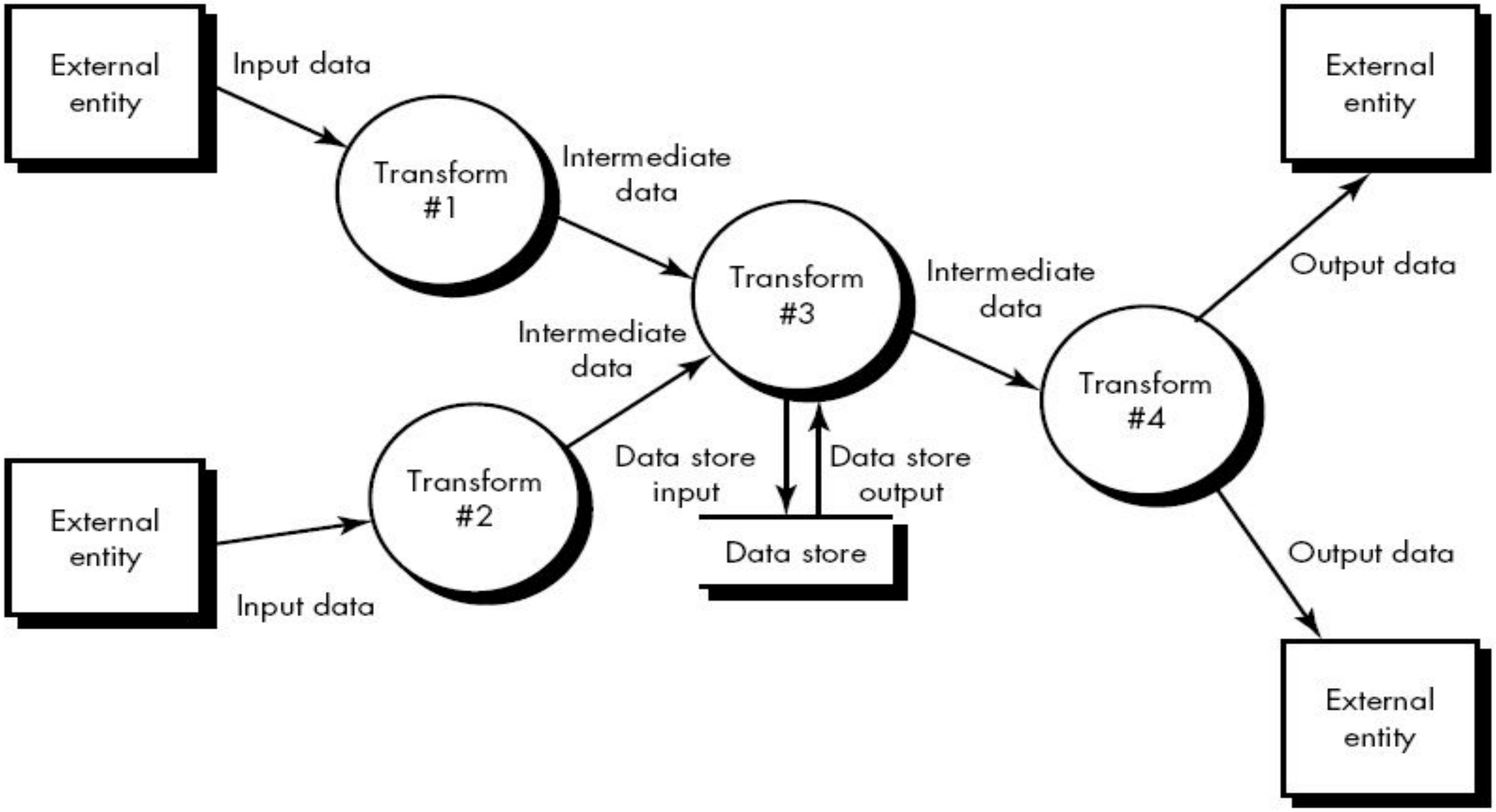
Function Modeling & Information Flow

- Information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms; applies hardware, software, and human elements to transform it; and produces output in a variety of forms
- Structured analysis began as an information flow modeling technique.
- A rectangle is used to represent an *external entity (software, hardware, a person)*
- A circle (sometimes called a *bubble*) represents a *process* or *transform* that is applied to data (or control) and changes it in some way.

Function Modeling & Information Flow

- An arrow represents one or more *data items* (data objects) and it should be labeled.
- The double line represents a data store—stored information that is used by the software.
- First data flow model (sometimes called a level 0 DFD or context diagram) represents the entire system.
- It provides incremental detail with each subsequent level.

Information Flow model



Creating a Data Flow Model

- It enables software engineer to develop models of the information domain and functional domain at the same time.
- Data flow diagram may be used to represent a system or software at any level of abstraction
- As DFD is refined into greater levels of detail, the analyst performs an implicit functional decomposition of the system.
- As DFD refinement results in corresponding refinement of data as it moves through the processes that represent the application

DFD Guidelines

- Depict the system as single bubble in level 0.
- Primary input and output should be carefully noted
- Refine by isolating candidate processes and their associated, data objects and data stores
- All arrows and bubbles should be labeled with meaningful names.
- Information flow continuity must be maintained from level to level.
- One bubble at a time should be refined.

Data flow models

- A level 0 DFD, also called a *fundamental system model* or a *context model,* represents the entire software element as a single bubble with input and output data indicated by incoming and outgoing arrows.
- Level 0 DFD refinement into level 1 DFD with all relevant processes to the system.
- Level 1 DFD each processes can be refined into level 2 DFD.
- Refinement of DFD continues until each bubble performs a simple function.

Control flow model

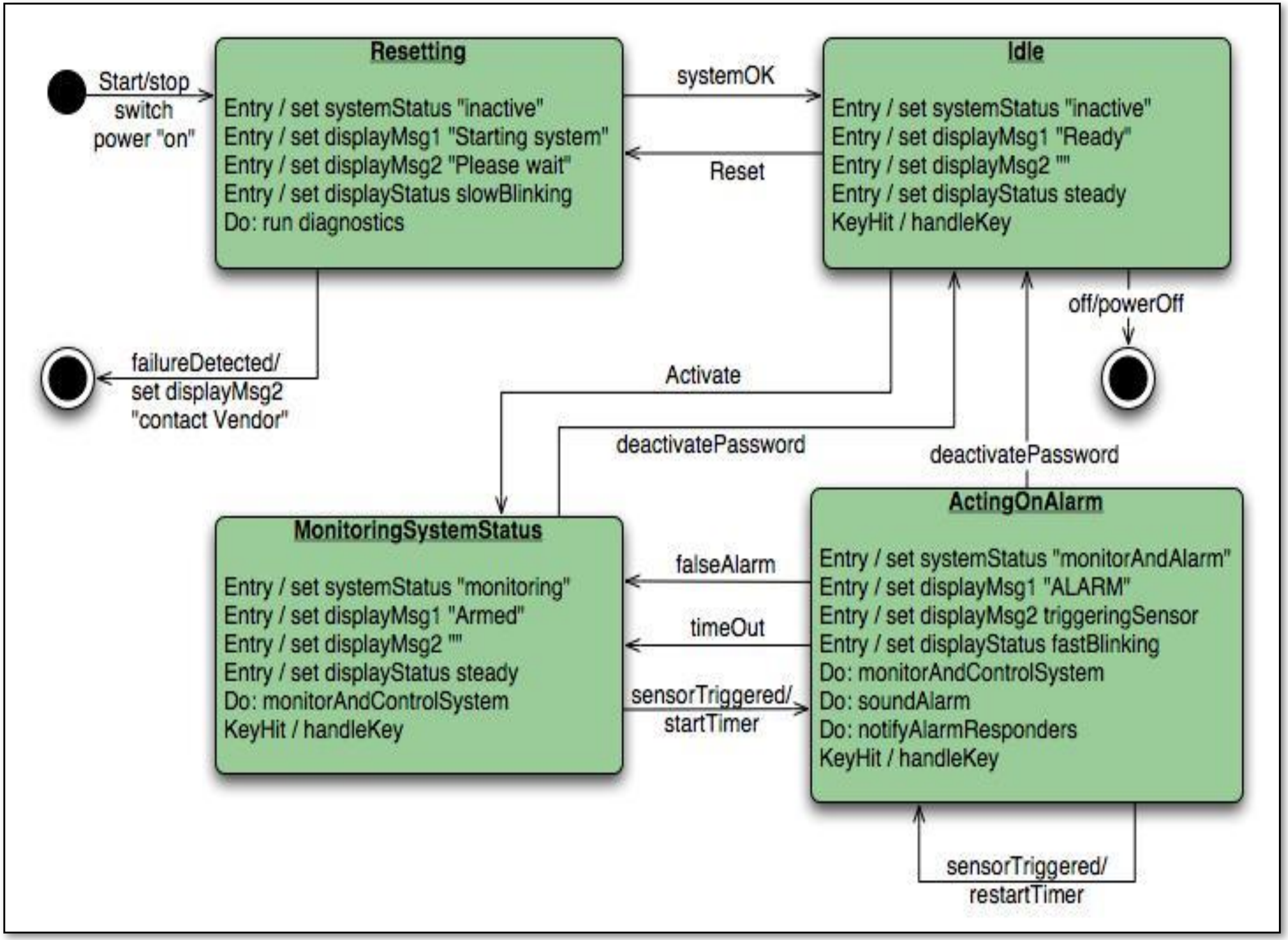
- Application which contains collection of classes are dependent on event rather than data, produce control information rather than reports or displays.
- Such application require the use of control flow modeling in addition to data flow modeling.

Guideline for control flow

- List all processes that are performed by the software.
- List all the interrupt conditions.
- List all activities that are performed by operator or actor.
- List all data conditions.
- Review all the “Control items” as possible for control flow inputs / outputs.
- Describe the behavior of a system by identifying its states; identify how each state is reached; define the transitions between states.
- Focus on possible omission – a very common error in specifying control

Control Specification (CSPEC)

- CSPEC represent the behavior of the system in two different ways.
- It contains a state diagram – sequential specification of behavior.
- It also contain program activation table – combinatorial specification of behavior.
- By reviewing the state diagram, a software engineer can determine the behavior of the system and can discover whether there are “holes” in specified behavior.
- CSPEC describe the behavior of the system, but it gives us no information about the inner working of the processes that activated result.


Start/stop switch power "on"
Resetting
Entry / set systemStatus "inactive"
Entry / set displayMsg1 "Starting system"
Entry / set displayMsg2 "Please wait"
Entry / set displayStatus slowBlinking
Do: run diagnostics
systemOK
Reset
Idle
Entry / set systemStatus "inactive"
Entry / set displayMsg1 "Ready"
Entry / set displayMsg2 ""
Entry / set displayStatus steady
KeyHit / handleKey
off/powerOff
failureDetected/ set displayMsg2 "contact Vendor"
Activate
deactivatePassword
deactivatePassword
MonitoringSystemStatus
Entry / set systemStatus "monitoring"
Entry / set displayMsg1 "Armed"
Entry / set displayMsg2 ""
Entry / set displayStatus steady
Do: monitorAndControlSystem
KeyHit / handleKey
falseAlarm
timeOut
sensorTriggered/ startTimer
ActingOnAlarm
Entry / set systemStatus "monitorAndAlarm"
Entry / set displayMsg1 "ALARM"
Entry / set displayMsg2 triggeringSensor
Entry / set displayStatus fastBlinking
Do: monitorAndControlSystem
Do: soundAlarm
Do: notifyAlarmResponders
KeyHit / handleKey
sensorTriggered/ restartTimer

Process Specification (PSPEC)

- Used to describe all flow model processes that appear at the final level of refinement.
- It include narrative text, a program design language (PDL) description of the process algorithm, mathematical equations, tables, diagrams or charts.
- By providing a PSPEC to accompany each bubble in the flow model, the software engineer creates a “mini-spec” that can serve as guide for design of the software component that will implement the process.

Class based modeling

- Identifying Analysis Classes
- Specifying Attributes
- Defining operations
- CRC modeling

Identifying Analysis classes

- Identify classes by examining the problem statement or by performing a "Grammatical Parse" on the use-cases or processing narratives developed for the system.

How do analysis classes manifest?

- <u>External entities</u> (other system, people, devices) that produce or consume information
- <u>Things</u> (reports, display, signals) that are part of the information domain for the problem.
- <u>Occurrences or events</u> – occur within the context of system operations.

Analysis classes (cont.)

- Roles (manager, engineer, salesperson) played by people who interact with the system.
- Organizational units (Division, group, team) that are relevant to an application,
- Places – establish the context of the problem and overall function of the system.
- Structures (sensors, computers) that define a class of objects or related classes of objects.

Selecting Criteria - Classes

- Retained Information – Potential class must be remembered so that system can function.
- Needed Services – Must have set of identifiable operations that can change the value of its attributes.
- Multiple Attributes – A class with single attribute may, in fact, be useful during design, but probably better represented as an attributes of another class.
- Common Attributes – These attributes apply to all instances of the class

Specifying Attributes

- Attributes are the set of data objects that fully define the class within the context of the problem.
- To develop attributes for class, a s/w can study use-case and select those “things” that reasonably “Belong” to the class.

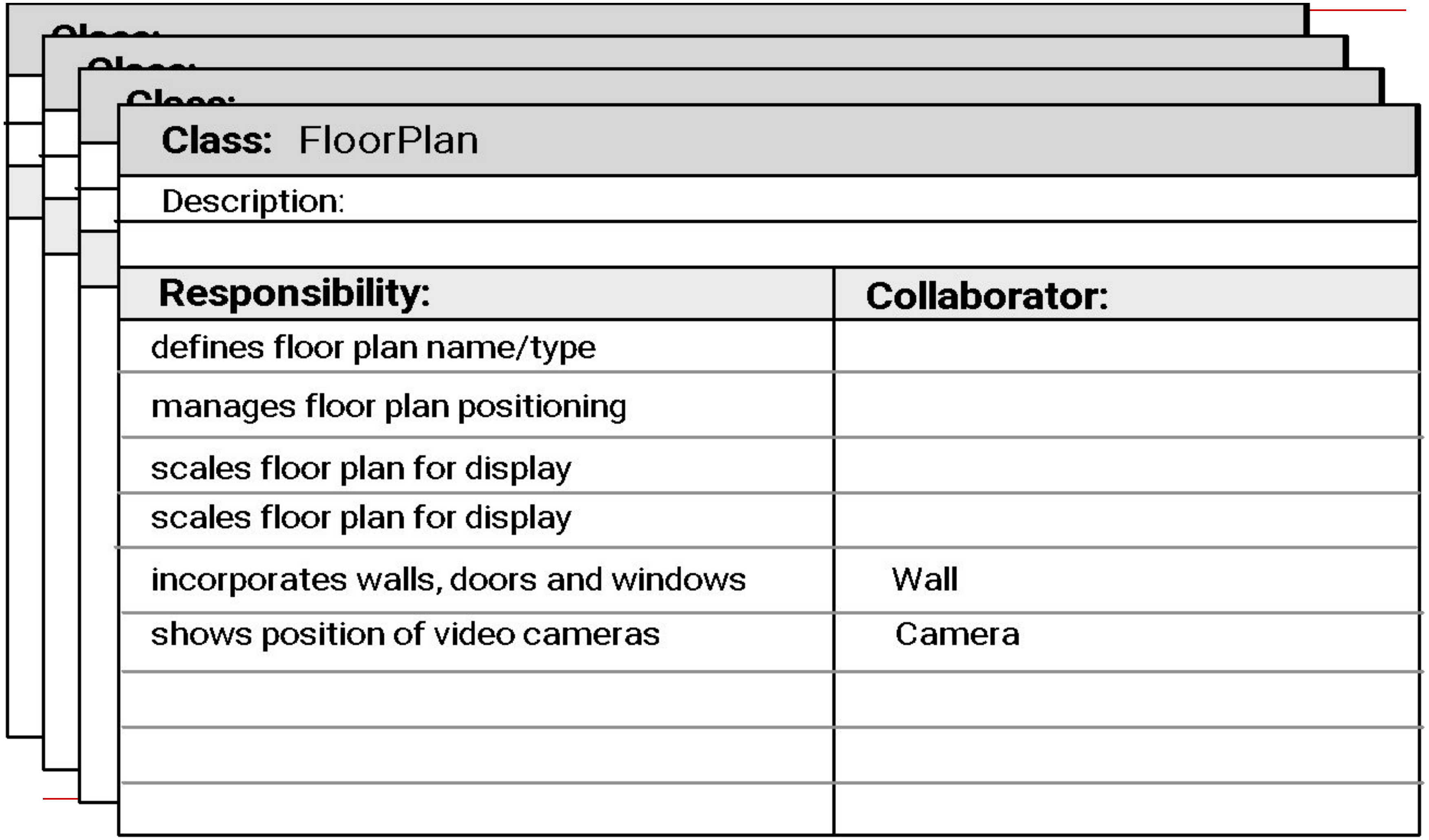
Defining operations

- Operations define the behavior of an object.
- Four broad categories
  1. Operations that manipulate data in some way.
  2. Operations that perform a computation.
  3. Operations that inquire about the state of an object
  4. Operations that monitor an object for the occurrence of the controlling event.
- To derive a set of operations, analyst study a use-case(or narrative) and select those operations that reasonably belongs.

Class-Responsibility-collaborator (CRC) modeling

- A CRC model is a collection of standard index cards that represent classes. Cards are divided into three sections.
  - Top of the cards write class name
  - In the body, list the class responsibility on left.
  - Collaborator on the right
- Simple means for identifying and organizing the classes that are relevant to system or product requirement.
- It make use of actual or virtual index cards.

CRC modeling



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

CRC – Classes Instantiation

- Three types classes:
  1. Entity Classes (model or business classes):- Represent things that are to be stored in a database and persist throughout the duration of the application.
  2. Boundary class:- used to create interface. It designed with the responsibility of managing the way entity objects are represented to users.
  3. Controller Class:- manage a “unit of work” from start to finish.
     1. Creation or update of entity objects.
     2. Representation of boundary objects as they obtain information from entity objects.
     3. Complex communication between sets of objects.
     4. Validation of data.

Responsibility – CRC modeling

- Guideline for allocating responsibility to classes:

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

Collaborations – CRC modeling

- Collaborations represent request from client to server in fulfillment of a client responsibility.

Ex. One object collaborate with another object if it needs to send some msg to other object.

- It identifying relationships between objects.
- Collaborations are identified by determining whether a class can fulfill each responsibility itself. If it cannot, then it needs to interact with another class.

Reviewing 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.

Cont.

- 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.

Behavioral Modeling

- Behavioral model indicates how software will respond to external events.
- To create model –
  - Evaluate all use cases to understand the sequence of interaction within the system.
  - Identify events
  - Create sequence for each use-case
  - Build a state diagram for the system.
  - Review the behavioral model to verify accuracy or consistency.

Identifying events with the use-cases

- Use-case represents a sequence of activities that involves actors and the system.
- An event occurs whenever the system and an actor exchange information.
- An event is not the information that has been exchanged, but rather the fact that information has been exchanged.
- An actor should be identified for each event.
- Information that is exchanged should be noted.
- Any conditions or constraints should be listed.

State Representation

- 2 diff. characteristics should be considered.
  - Passive State
  - Active State
- **Passive state** is simply the current status of all of an object’s attributes.

Ex. Player – class

current position and orientation – attributes.

- **Active State** is current state of the object as it undergoes a continuing transformation or processing.

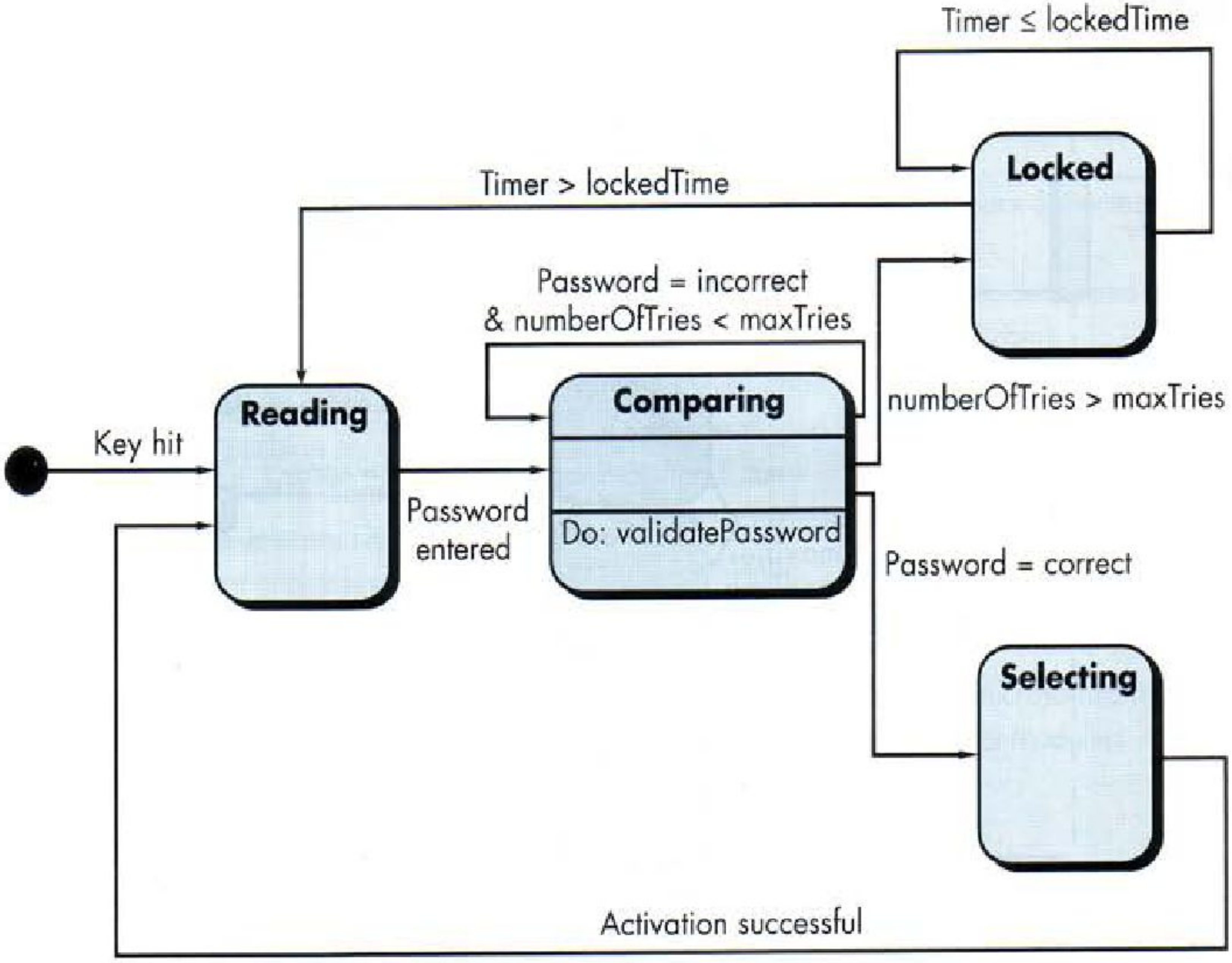
Ex. Player – class

active state – moving, injured, trapped, lost etc.

An event must occur to force an object to make a transition from one active state to another.

State diagram for analysis classes

- UML state diagram that represents active states for each class and events that causes changes between these active state.


Timer ≤ lockedTime
Locked
Timer > lockedTime
Password = incorrect & numberOfTries < maxTries
Comparing
Do: validatePassword
numberOfTries > maxTries
Key hit
Reading
Password entered
Password = correct
Selecting
Activation successful

- An action occurs concurrently with the state transition or as a sequence of it and generally involves one or more operations of the object.

Sequence diagram

- It indicates how events cause transitions from object to object .
- Once event have identified by examining a use-cases, the modeler creates a sequence diagram.
- It representation of how events cause flow from one objects to another as function of time.
- Its shorthand version of use-case diagram that represent key classes and the events that cause behavior to flow from class to class.
- System objects and events will help in creation of effective design.

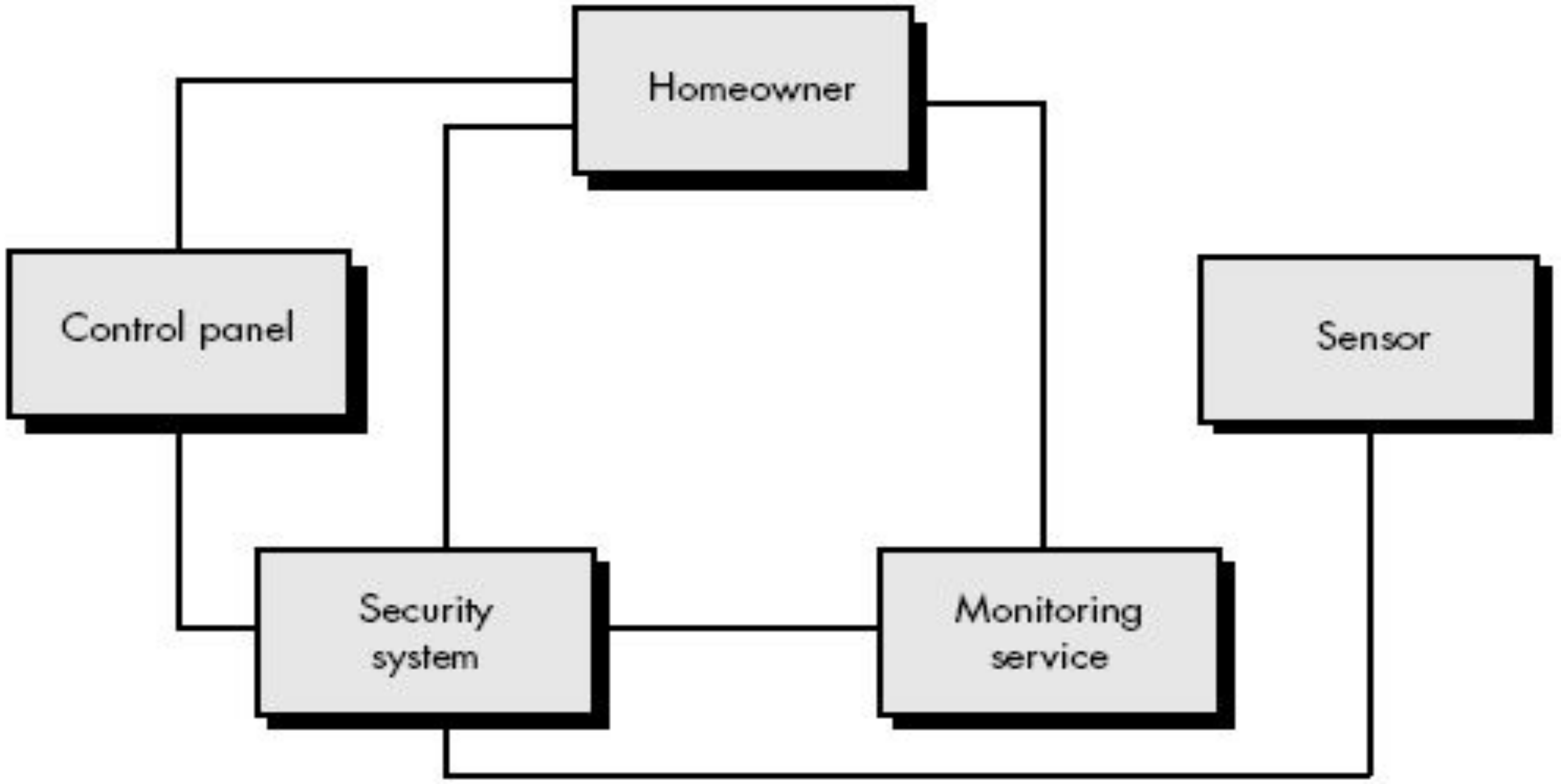
Mechanics of Structured Analysis

- It all about
  - Entity relationship diagram (ERD)
  - Data flow diagram (DFD)
  - State transition diagram (STD)

ERD

- Creating ERD diagram:
- Customers are asked to list the “things” that the application or business process addresses
- The analyst and customer defined connection exist between data object and other objects (if any)
- Wherever a connection exists, the analyst and the customer create one or more object/relationship pairs.
- For each object/relationship pair, cardinality and modality are explored.
- Steps 2 through 4 are continued iteratively until all object/relationships have been defined.
- The attributes of each entity are defined.
- An entity relationship diagram is formalized and reviewed.
- Steps 1 through 7 are repeated until data modeling is complete.

Example


Homeowner
Control panel
Sensor
Security system
Monitoring service

Safe-Home Security System:

- Step 1: Identified things
  - homeowner
  - control panel
  - sensors
  - security system
  - monitoring service
- Step 2: one or more object/relationship pairs are identified for each connection.
- Step 3: **security system** *monitors* **sensor**
  - **security system** *enables/disables* **sensor**
  - **security system** *tests* **sensor**
  - **security system** *programs* **sensor**
- Step 4: Cardinality and modality
  - The cardinality between **security system** and **sensor** is one to many.
  - Modality of **security system** (mandatory) and **sensor**(mandatory)
- Step 5: repeat 2 to 4 for all objects.
- Step 6: Each object is studied to determine its attributes.
  - For example: **sensor** -- sensor type, internal identification number, zone location, and alarm level.

Data dictionary

- ***Data dictionary*** is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of stores and [even] intermediate calculations.
- Data dictionary is always implemented as part of a CASE "structured analysis and design tool."
- Most contain following information :
- **Name** —the primary name of the data or control item, the data store or an external entity.
- **Alias** —other names used for the first entry.
- **Where-used/how-used** —a listing of the processes that use the data or control item and how it is used (e.g., input to the process, output from the process, as a store, as an external entity.

Data dictionary

- *Content description*—a notation for representing content.
- *Supplementary information*—other information about data types, preset values (if known), restrictions or limitations, and so forth.

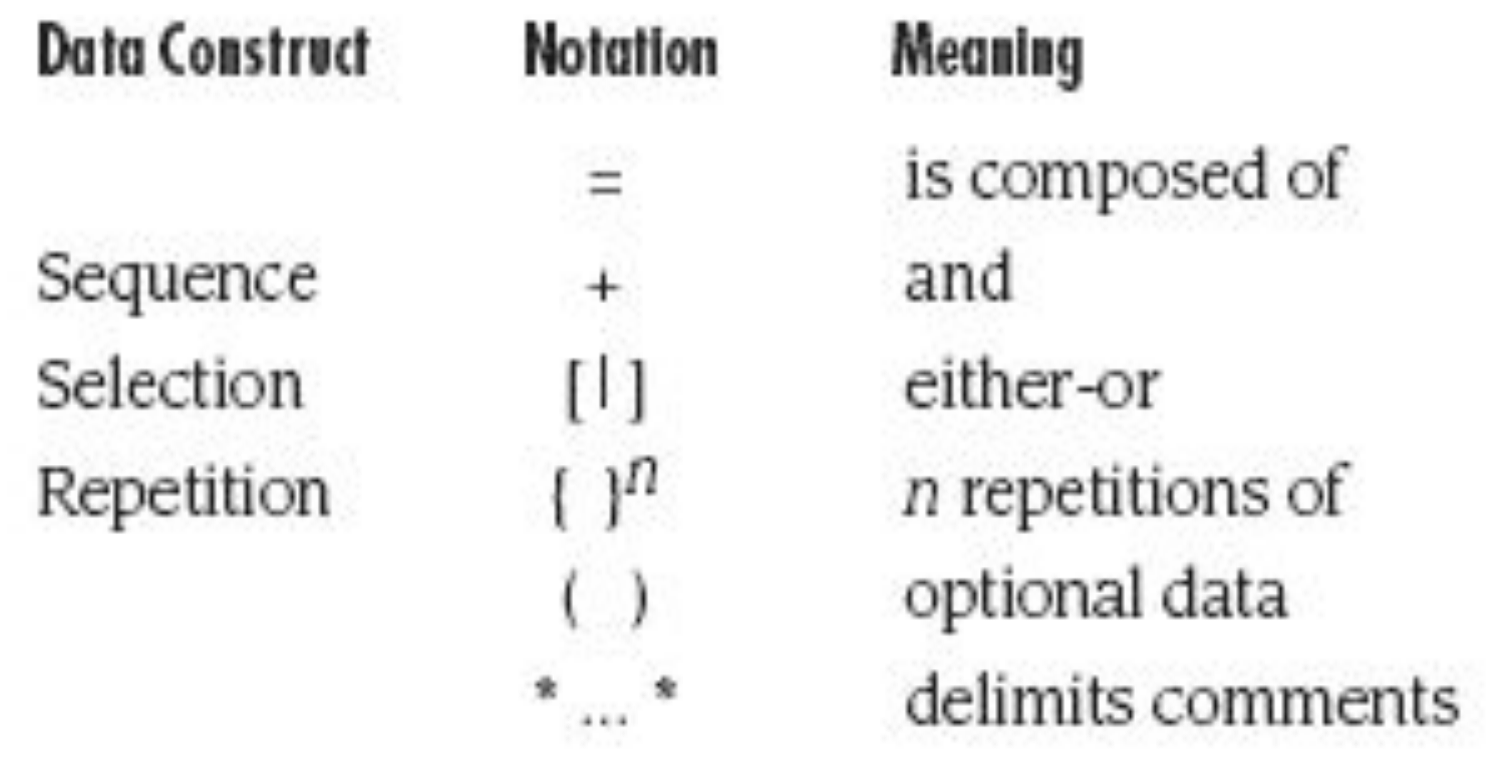
Data dictionary

- Once a data object or control item name and its aliases are entered into the data dictionary, consistency in naming can be enforced. That is, if an analysis team member decides to name a newly derived data item **xyz,** but **xyz** is already in the dictionary, the CASE tool supporting the dictionary posts a warning to indicate duplicate names.
- “Where-used/how-used” information is recorded automatically from the flow models. When a dictionary entry is created, the CASE tool scans DFDs and CFDs to determine which processes use the data or control information and how it is used.

Data dictionary

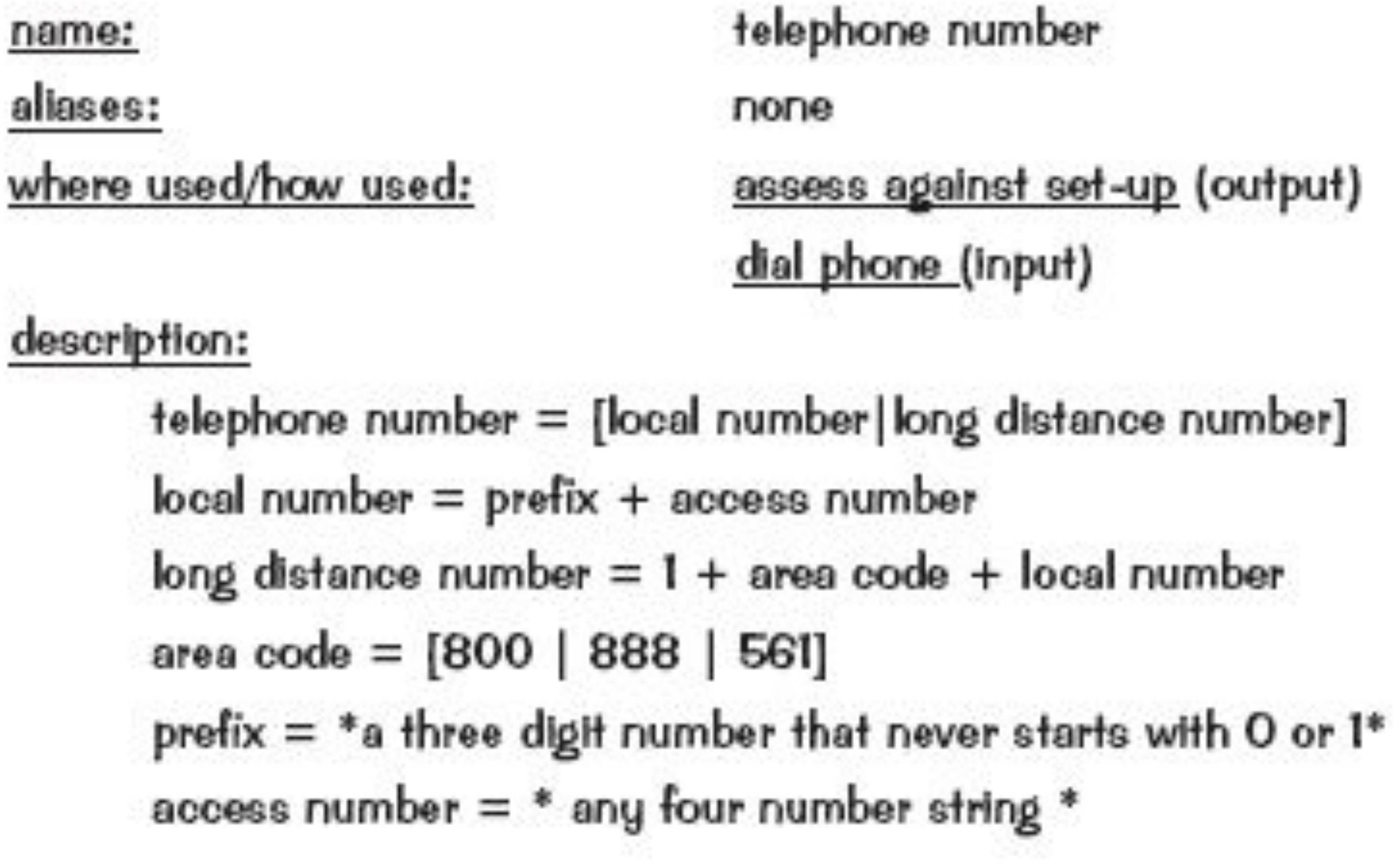
- For large projects, it is often quite difficult to determine the impact of a change. Many a software engineer has asked, "Where is this data object used? What else will have to change if we modify it? What will the overall impact of the change be?" Because the data dictionary can be treated as a database,

The notation used to develop a content description is noted in the following table:



Data Construct	Notation	Meaning
	=	is composed of
Sequence	+	and
Selection	[|]	either-or
Repetition	$\{ \}^n$	*n* repetitions of
	( )	optional data
	* ... *	delimits comments

Data dictionary example



name:	telephone number
aliases:	none
where used/how used:	assess against set-up (output)
	dial phone (input)

description:

telephone number = [local number|long distance number]

local number = prefix + access number

long distance number = 1 + area code + local number

area code = [800 | 888 | 561]

prefix = *a three digit number that never starts with 0 or 1*

access number = * any four number string *