Analysis Model

Overall Objectives

- Three primary objectives
  - To describe what the customer requires
  - To establish a basis for the creation of a software design
  - To define a set of requirements that can be validated once the software is built
- All elements of an analysis model are directly traceable to parts of the design model, and some parts overlap
- Provides the software designer with a representation of information, function, and behavior
  - This is later translated into architectural, interface, class/data and component-level designs

Goals of Analysis Modeling

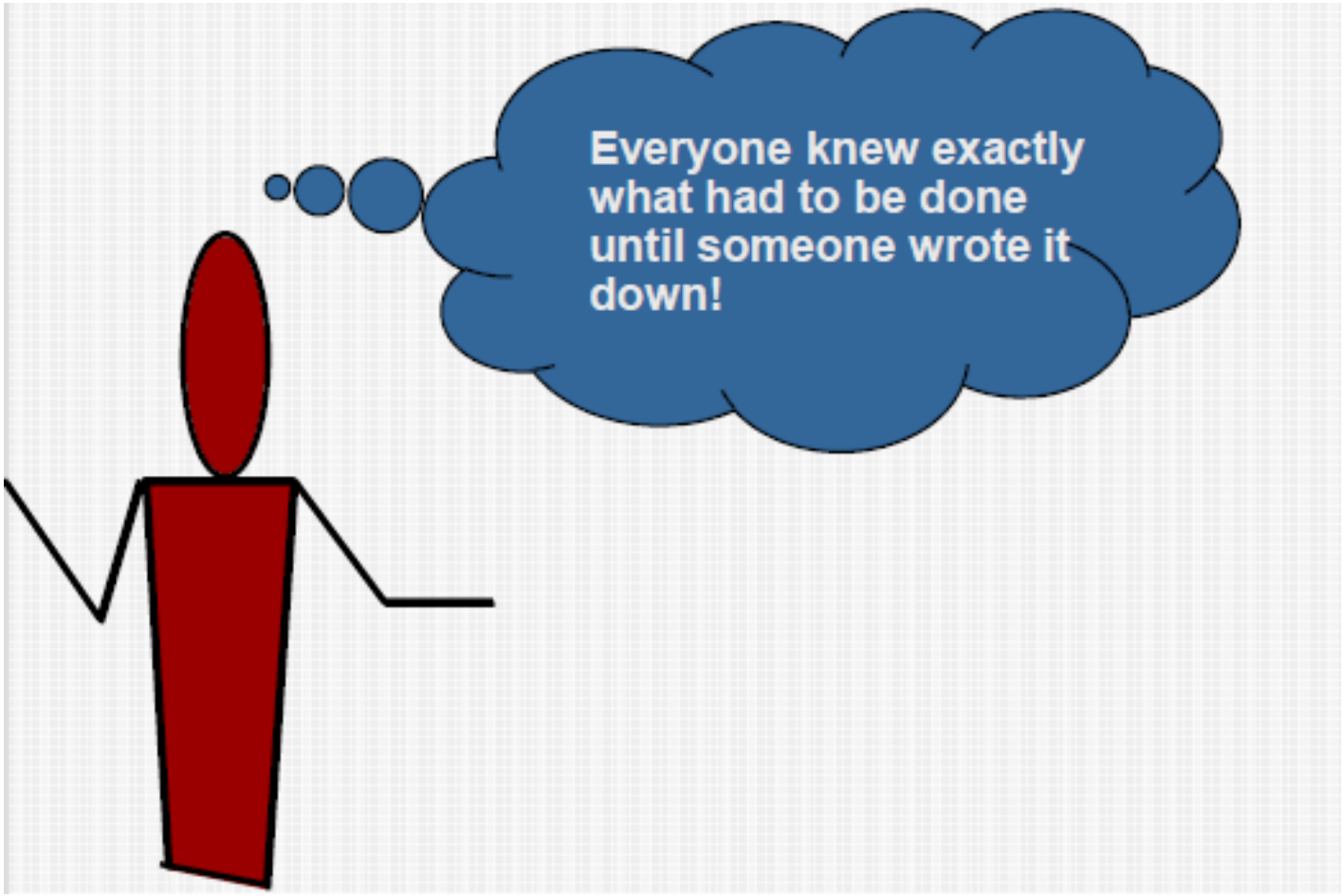
- Provide first technical representation of a system
- Is easy to understand and maintain
- Partition the system to deal with the problem of size
- Uses graphics whenever possible
- Differentiates between essential information versus implementation information
- Helps in the tracking and evaluation of interfaces
- Provides tools other than narrative text to describe software logic and policy

Analysis Rules of Thumb

- The analysis 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 following
  - Information domain, function, and behavior of the system
- The model should delay the consideration of infrastructure and other non-functional models until the design phase
  - First complete the analysis of the problem domain
- The model should minimize coupling throughout the system
  - Reduce the level of interconnectedness among functions and classes
- The model should provide value to all stakeholders
- The model should be kept as simple as can be

– Arlow & Neustadt

Happily misunderstood…



Requirements Engineering Tasks
(Review)

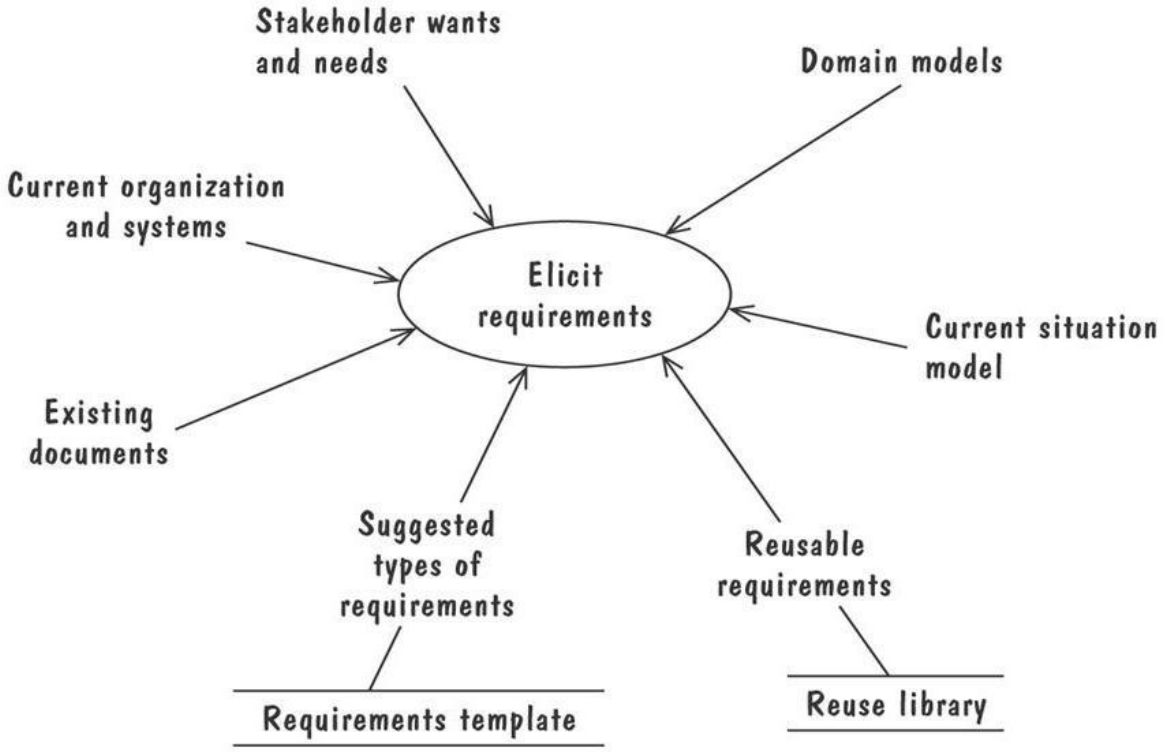
- Inception—ask a set of questions that establish …
  - basic understanding of the problem
  - the people who want a solution
  - the nature of the solution that is desired, and
  - the effectiveness of preliminary communication and collaboration between the customer and the developer
- Elicitation—elicit requirements from all stakeholders
- Elaboration—create an analysis model that identifies data, function and behavioral requirements
- Negotiation—agree on a deliverable system that is realistic for developers and customers

Requirements Engineering Tasks (contd)
(Review)

- Specification—can be any one (or more) of the following:
  - A written document
  - A set of graphical models
  - A formal mathematical model
  - A collection of user scenarios (use-cases)
  - A prototype
- Validation—a review mechanism that looks for
  - errors in content or interpretation
  - areas where clarification may be required
  - missing information
  - inconsistencies (a major problem when large products or systems are engineered)
  - conflicting or unrealistic (unachievable) requirements.
- Requirements management

Requirements Elicitation
Means of Eliciting Requirements

The Volere requirements process model suggests some additional sources for requirements



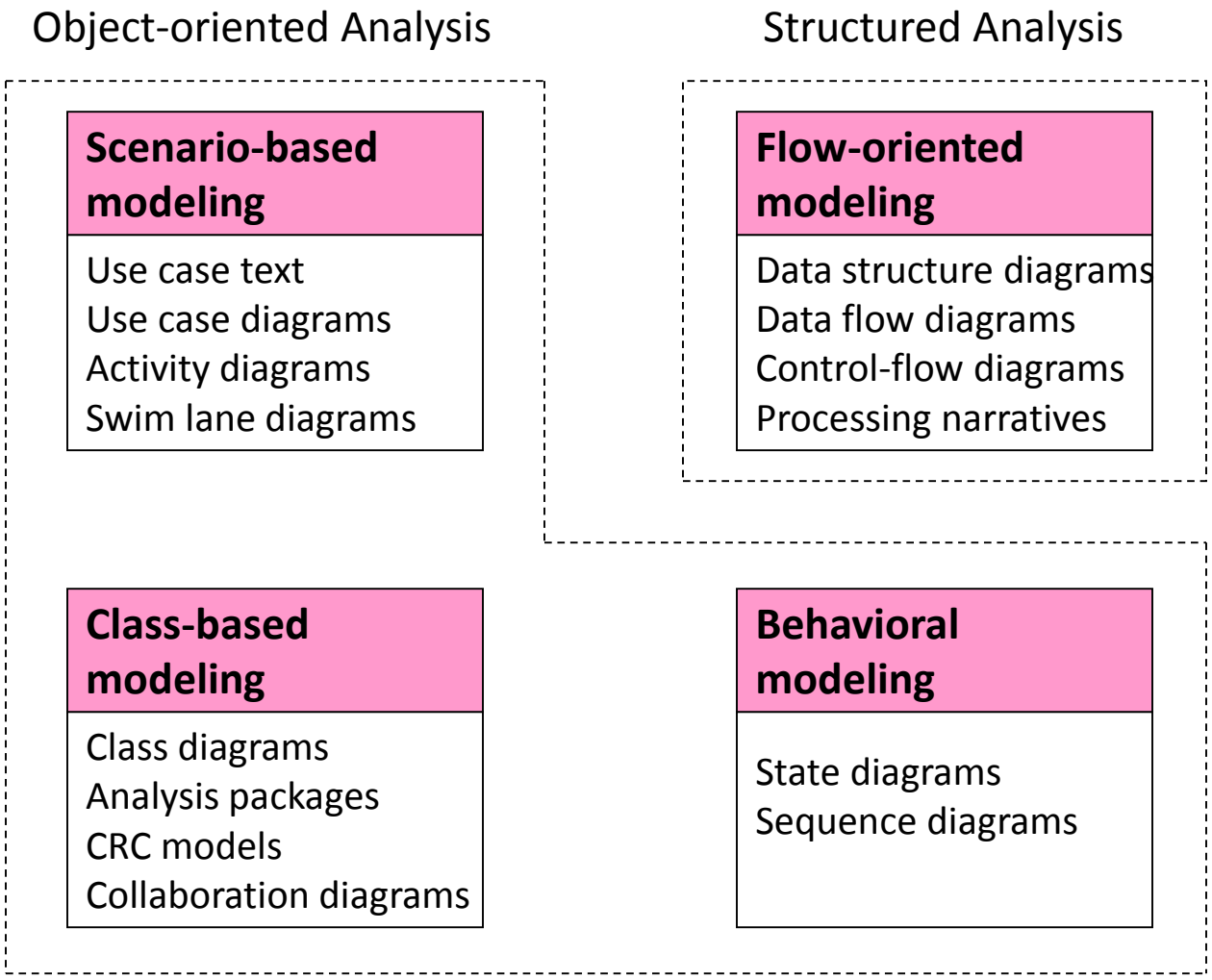
Analysis Modeling Approaches

- Structured analysis
  - Considers data and the processes that transform the data as separate entities
  - Data is modeled in terms of only attributes and relationships (but no operations)
  - Processes are modeled to show the 1) input data, 2) the transformation that occurs on that data, and 3) the resulting output data
- Object-oriented analysis
  - Focuses on the definition of classes and the manner in which they collaborate with one another to fulfill customer requirements

A Set of Models

- **Scenario-based modeling** – represents the system from the user's point of view
- **Flow-oriented modeling** – provides an indication of how data objects are transformed by a set of processing functions
- **Class-based modeling** – defines objects, attributes, and relationships
- **Behavioral modeling** – depicts the states of the classes and the impact of events on these states

Elements of the Analysis Model



Scenario-Based Modeling

Use-Cases

- A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an "actor"—a person or device that interacts with the software in some way
- Each scenario answers the following questions:
  - Who is the primary actor, the secondary actor (s)?
  - What are the actor's goals?
  - What preconditions should exist before the story begins?
  - What main tasks or functions are performed by the actor?
  - What extensions might be considered as the story is described?
  - What variations in the actor's interaction are possible?
  - What system information will the actor acquire, produce, or change?
  - Will the actor have to inform the system about changes in the external environment?
  - What information does the actor desire from the system?
  - Does the actor wish to be informed about unexpected changes?

Use case—detailed example (Pressman)

Example: “SAFEHOME” system (Pressman)

Use case name: ***InitiateMonitoring***

Participating actors: homeowner, technicians, sensors

Flow of events (homeowner):

--Homeowner wants to set the system when the homeowner leaves house or remains in house
--Homeowner observes control panel
--Homeowner enters password
--Homeowner selects “stay” or “away”
--Homeowner observes that read alarm light has come on, indicating the system is armed

Use detailed example (Pressman)--continued

Entry condition(s)

Homeowner decides to set control panel

Exit condition(s)

Control panel is *not ready*; homeowner must check all sensors and reset them if necessary

Control panel indicates incorrect password (one beep)—homeowner enters correct password

Password not recognized—must contact monitoring and response subsystem to reprogram password

Stay selected: control panel beeps twice and lights *stay* light; perimeter sensors are activated

Away selected: control panel beeps three times and lights away light; all sensors are activated

Use case—detailed example (Pressman)

Quality requirements:

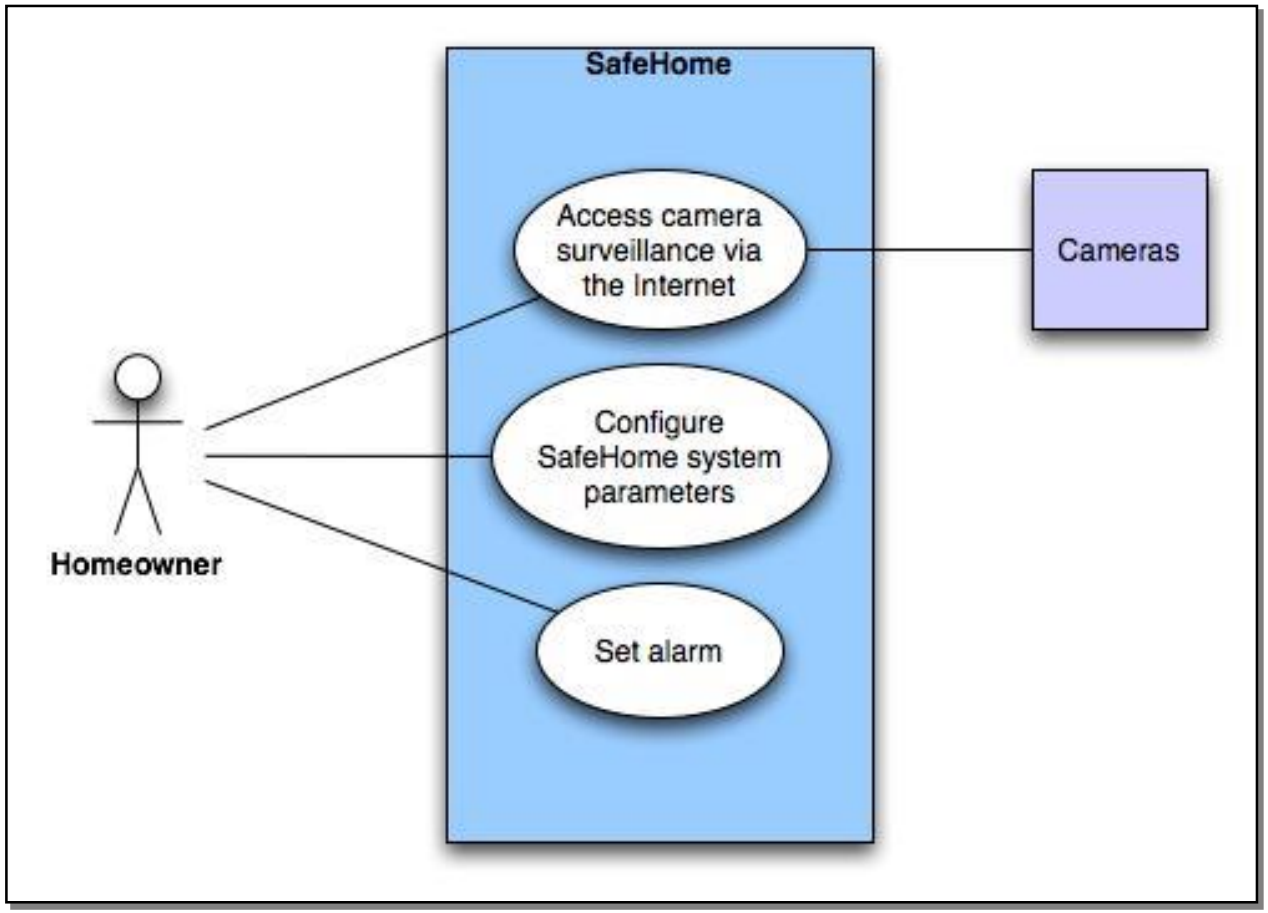
Control panel may display additional text messages

time the homeowner has to enter the password from the time the first key is pressed

Ability to activate the system without the use of a password or with an abbreviated password

Ability to deactivate the system before it actually activates

Use-case Diagram

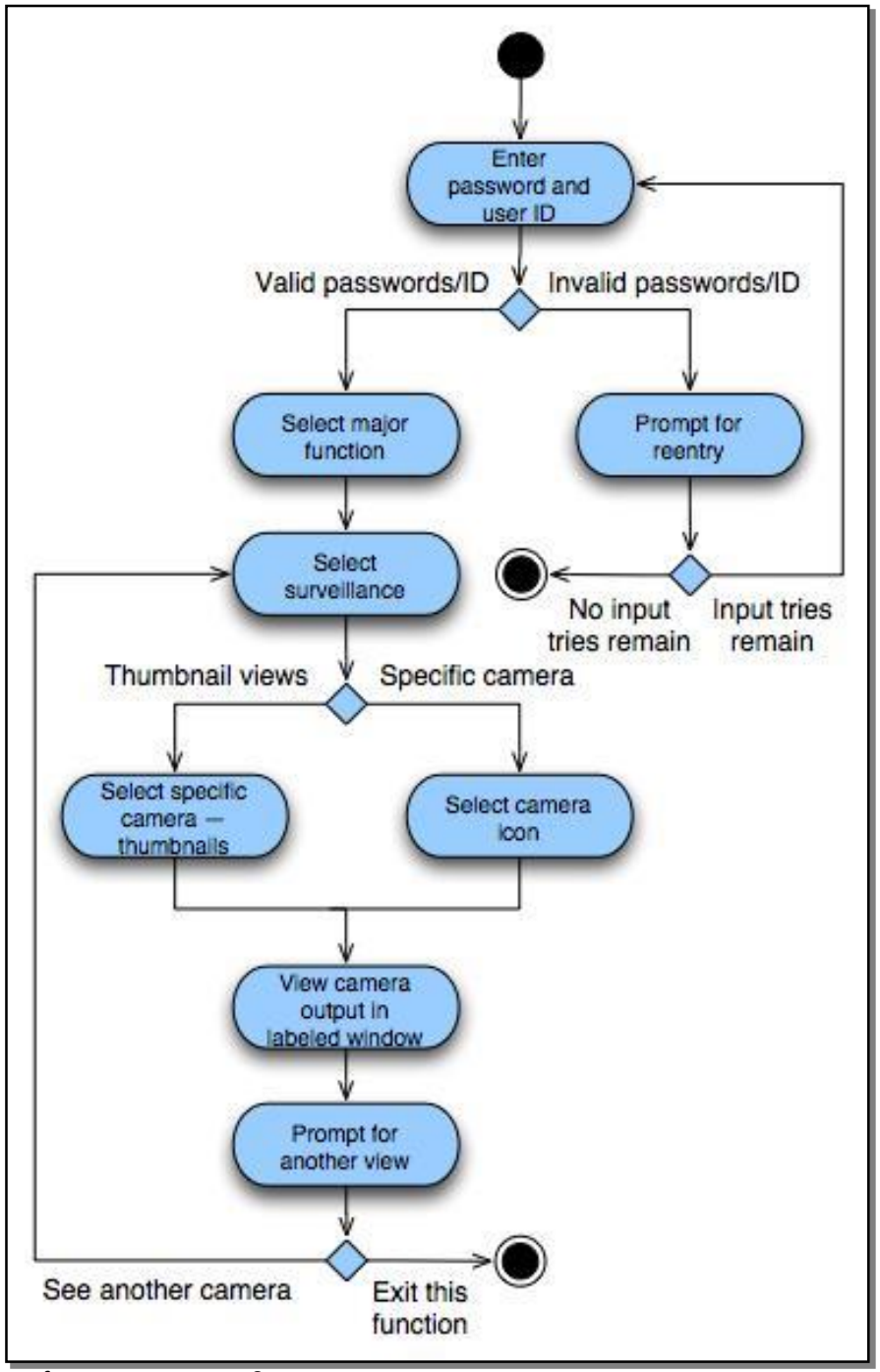


Use-case diagram for surveillance function

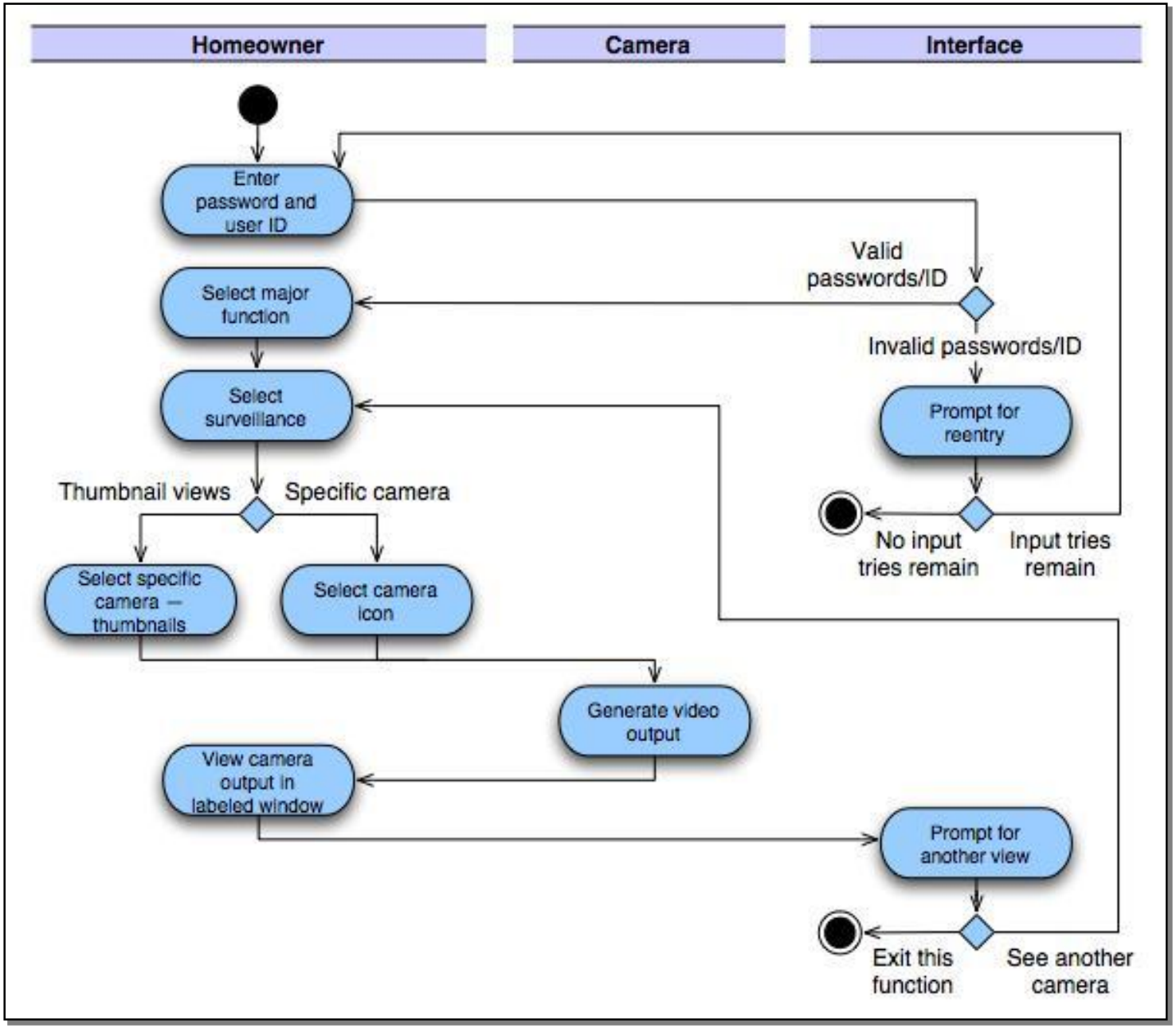
Alternative Actions

- Can the actor take some other action at this point?
- Is it possible that the actor will encounter some error condition at this point?
- Is it possible that the actor will encounter behavior invoked by some event outside the actor's control?

Activity diagram for Access camera surveillance—display camera views function



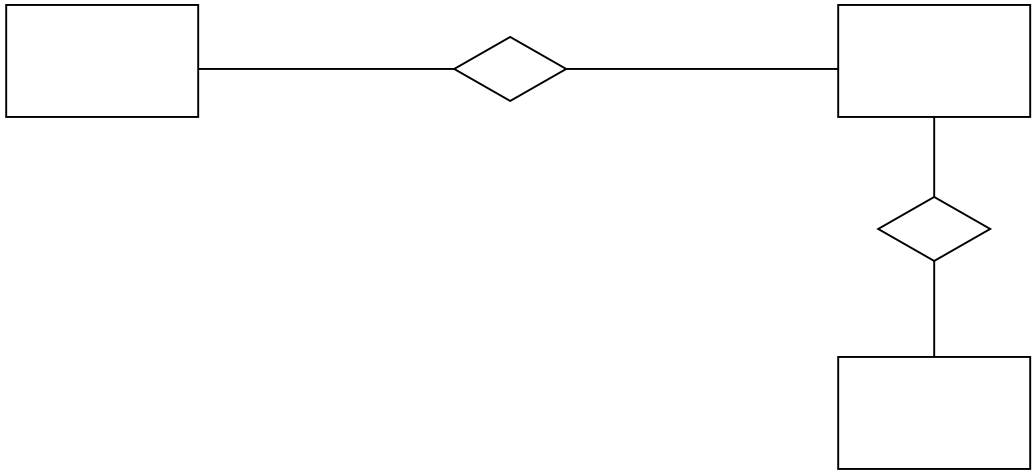
Swimlane diagram



Flow-Oriented Modeling

Data Modeling

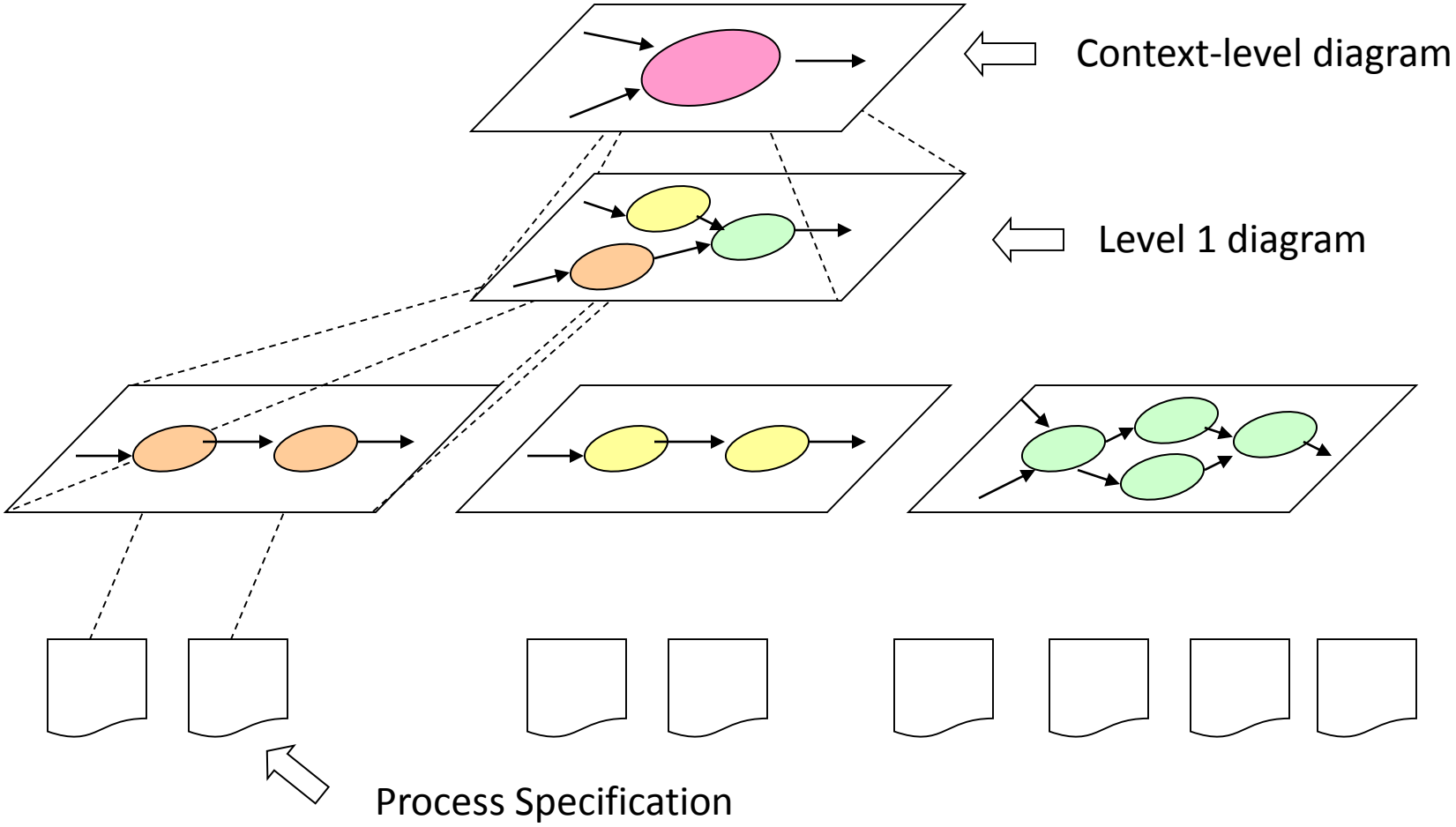
- Identify the following items
  - Data objects (Entities)
  - Data attributes
  - Relationships
  - Cardinality (number of occurrences)



Data Flow and Control Flow

- Data Flow Diagram
  - Depicts how input is transformed into output as data objects move through a system
- Process Specification
  - Describes data flow processing at the lowest level of refinement in the data flow diagrams
- Control Flow Diagram
  - A **control flow diagram** (**CFD**) is a diagram to describe the control flow of a process or program (at various levels of abstraction).

Diagram Layering and Process Refinement



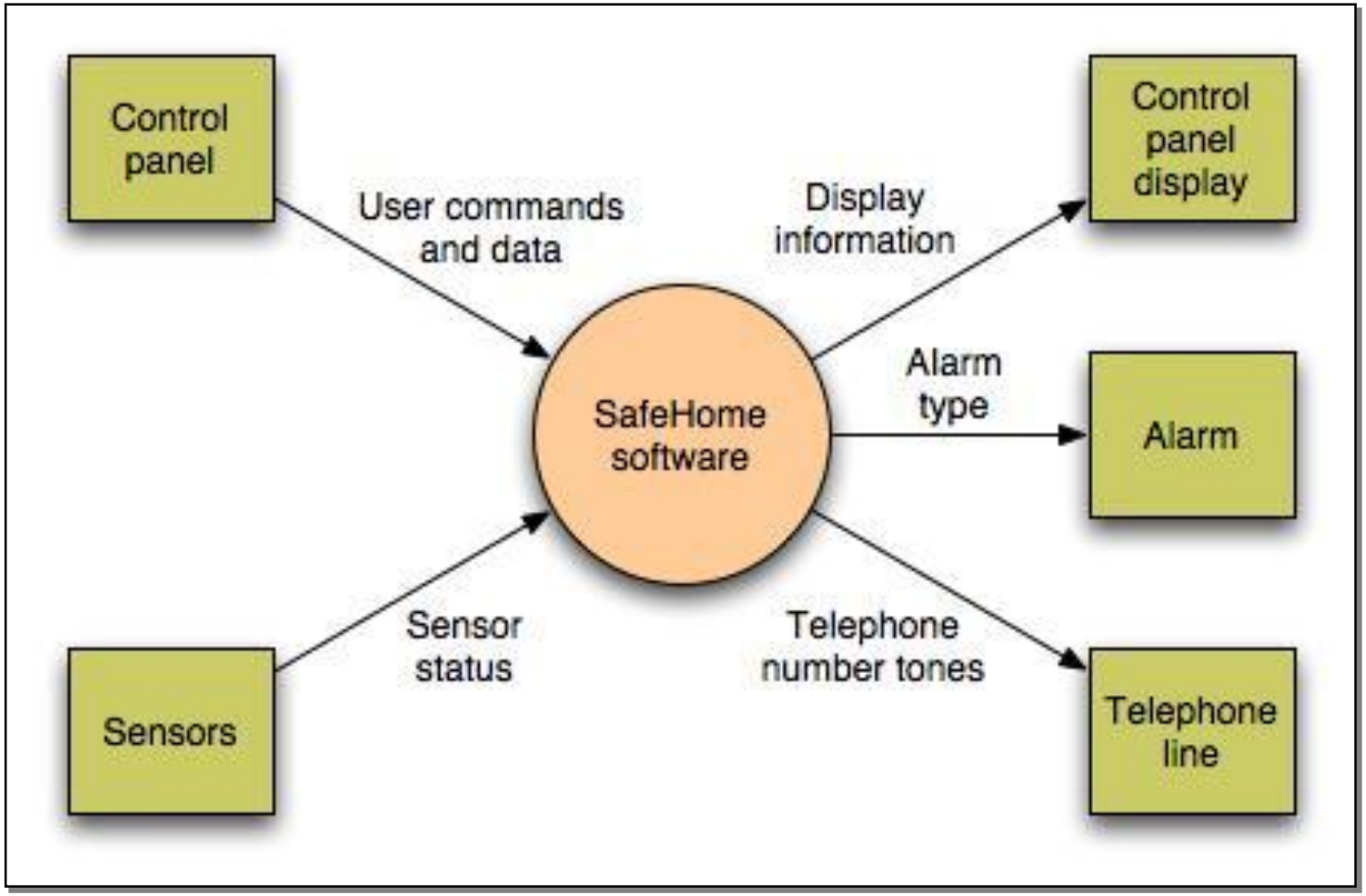
Guidelines

- Depict the system as single bubble in level 0.
- Carefully note primary input and output.
- Refine by isolating candidate processes and their associated data objects and data stores.
- Label all elements with meaningful names.
- Maintain information conformity between levels.
- Refine one bubble at a time.

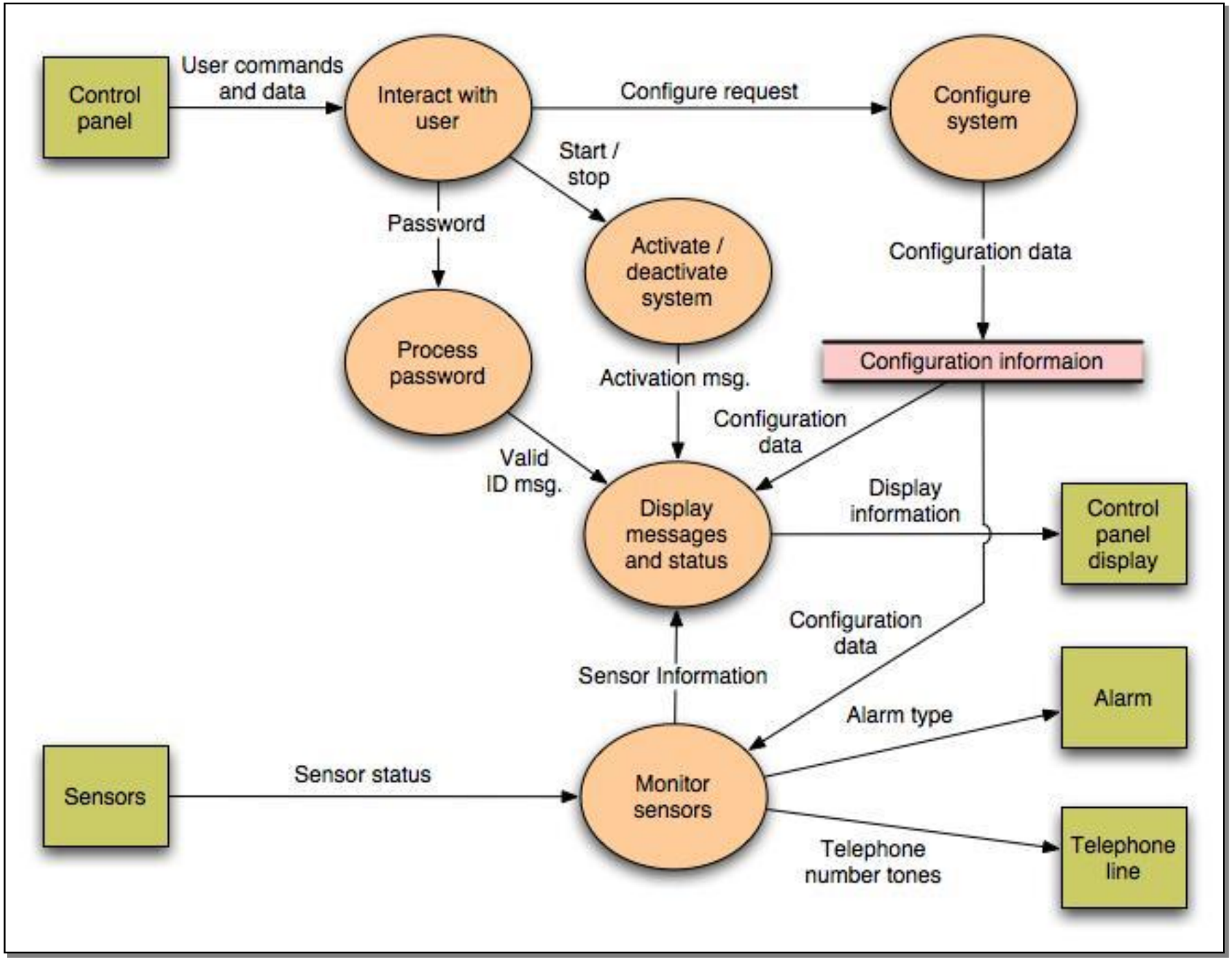
Grammatical Parse

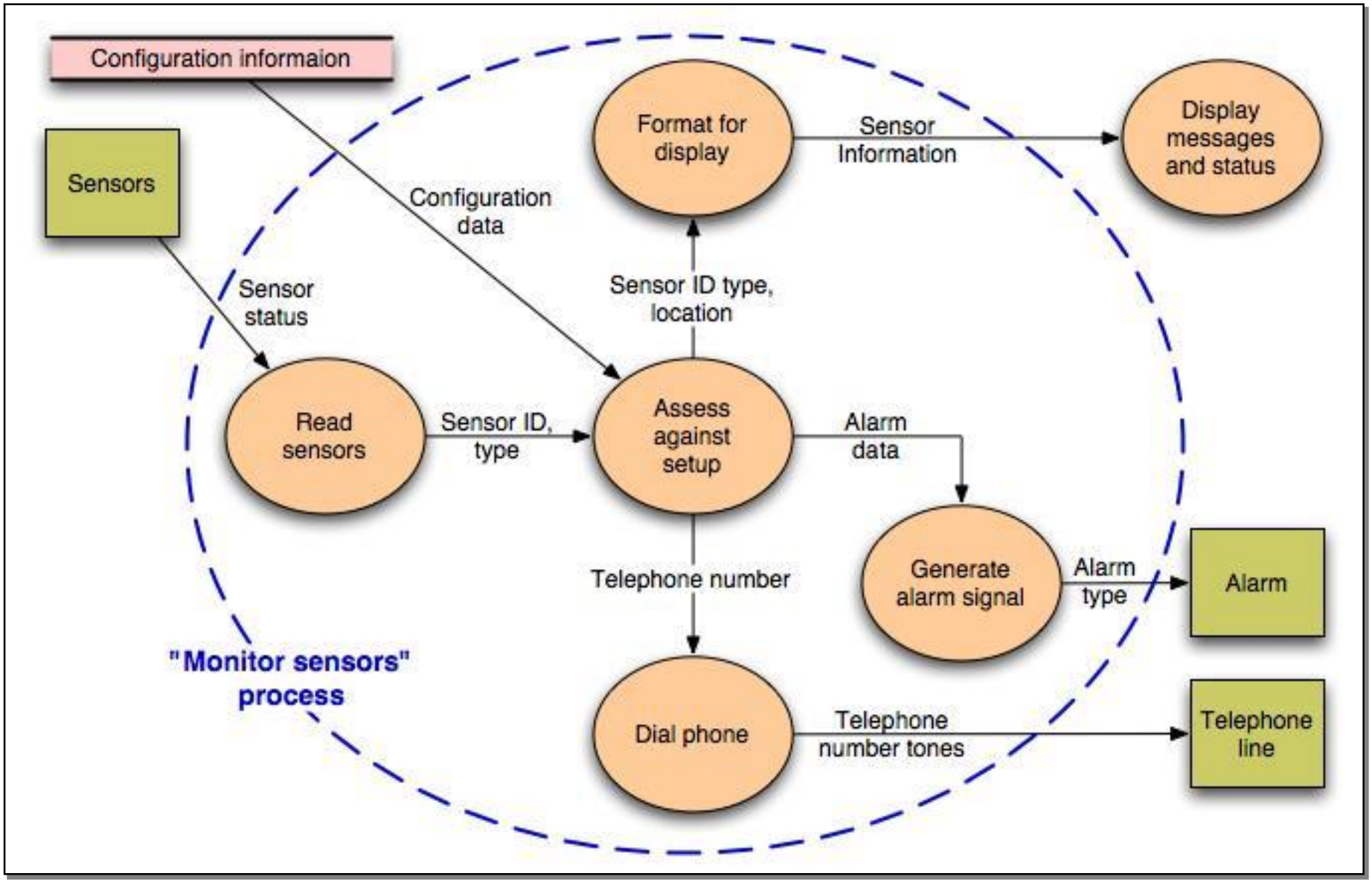
- The ***SafeHome*** **security function** enables the **homeowner** to configure the **security system** when it is installed, monitors all **sensors** connected to the security system, and interacts with the homeowner through the **Internet**, a **PC**, or a **control panel**.
- During **installation**, 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(s)** are input for dialing when a **sensor event** occurs.
- When a sensor event is recognized, the software invokes an audible **alarm** attached to the system. After a **delay time** that is specified by the homeowner during system configuration activities, the software *dials* a telephone number of a **monitoring service**, provides information about the **location**, reporting the nature of the event that has been detected. The telephone number will be redialed every 20 seconds until a **telephone connection** is obtained.
- The homeowner receives **security information** via a control panel, the PC, or a browser, collectively called an **interface**. The interface displays prompting **messages** and system **status information** on the control panel, the PC, or the browser window. Homeowner interaction takes the following form…

Data Flow Diagram



Context-level DFD for *SafeHome* security function


Control panel
User commands and data
Interact with user
Configure request
Configure system
Start / stop
Password
Activate / deactivate system
Configuration data
Process password
Configuration informaion
Activation msg.
Configuration data
Valid ID msg.
Display messages and status
Display information
Control panel display
Configuration data
Sensor Information
Alarm
Alarm type
Sensors
Sensor status
Monitor sensors
Telephone number tones
Telephone line



Level 2 DFD that refines the monitor sensors process

Class-Based Modeling

Class-Based Modeling

Class-based modeling represents:

- objects that the system will manipulate
- operations (also called 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 include classes and objects, attributes, operations, CRC models, collaboration diagrams and packages.

Manifestations of Analysis Classes

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

Potential Classes

1. Retained information. The potential class will be useful during analysis only if information about it must be remembered so that the system can function.

2. Needed services. The potential class must have a set of identifiable operations that can change the value of its attributes in some way.

3. Multiple attributes. During requirement analysis, the focus should be on "major" information; a class with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another class during the analysis activity.

4. Common attributes. A set of attributes can be defined for the potential class and these attributes apply to all instances of the class.

5. Common operations. A set of operations can be defined for the potential class and these operations apply to all instances of the class.

6. Essential requirements. External entities that appear in the problem space and produce or consume information essential to the operation of any solution for the system will almost always be defined as classes in the requirements model.

Grammatical Parse

- The ***SafeHome*** **security function** enables the **homeowner** to configure the **security system** when it is 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(s)** are input for dialing when a **sensor event** occurs.
- When a sensor event is recognized, the software invokes an audible **alarm** attached to the system. After a **delay time** that is specified by the homeowner during system configuration activities, the software *dials* a telephone number of a **monitoring service**, provides information about the **location**, reporting the nature of the event that has been detected. The telephone number will be redialed every 20 seconds until a **telephone connection** is obtained.
- The homeowner receives **security information** via a control panel, the PC, or a browser, collectively called an **interface**. The interface displays prompting **messages** and system **status information** on the control panel, the PC, or the browser window. Homeowner interaction takes the following form…

Identifying Classes

Potential class	Classification	Accept / Reject
homeowner	role; external entity	reject: 1, 2 fail
sensor	external entity	accept
control panel	external entity	accept
installation	occurrence	reject
(security) system	thing	accept
number, type	not objects, attributes	reject: 3 fails
master password	thing	reject: 3 fails
telephone number	thing	reject: 3 fails
sensor event	occurrence	accept
audible alarm	external entity	accept: 1 fails
monitoring service	organizational unit; ee	reject: 1, 2 fail

Object class identification
(as per Sommerville)

- Identifying object classes is often a difficult part of object oriented design.
- There is no 'magic formula' for object identification. It relies on the skill, experience, and domain knowledge of system designers.
- Object identification is an iterative process. You are unlikely to get it right first time.

Class Types

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 work from start to finish. That is, controller classes can be designed to manage

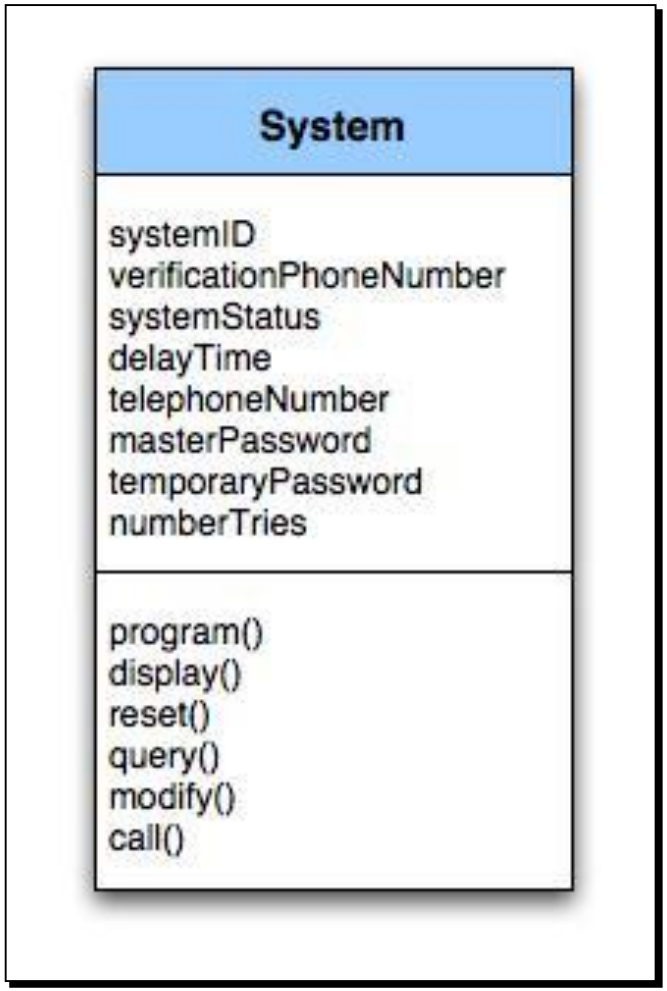
–the creation or update of entity objects;

– 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.

Class Diagram



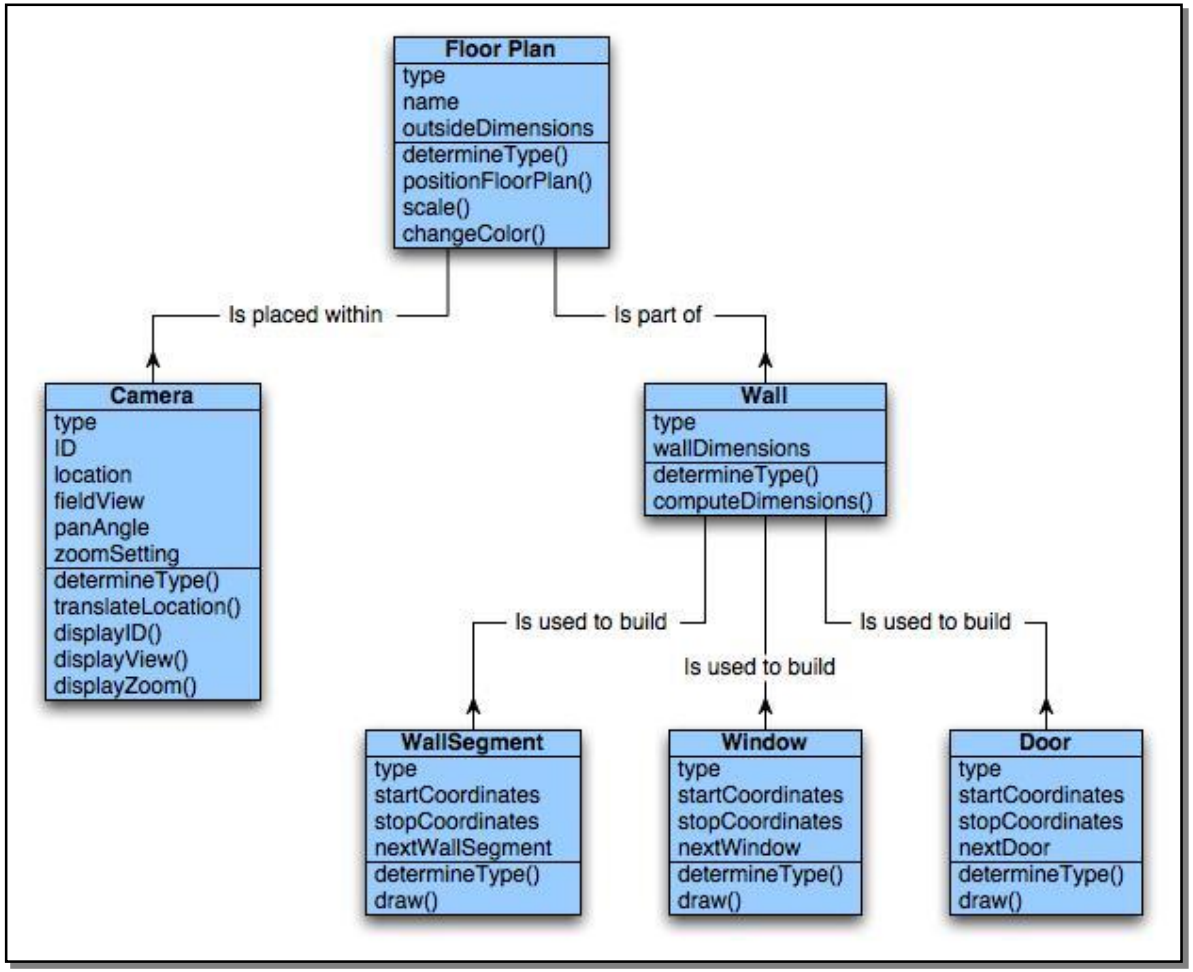
Attributes : Describe a class in the context of problem space. To identify attributes, we study each use case and select "things" that reasonably "belong" to the class

Operations : Define the behavior of a class. Can be divided into four broad categories:

1) That manipulate data in some way
2) That perform computation
3) That inquire about the state of object
4) That monitor object for a controlling event

Class diagram for the system class

Class Diagram



Class diagram for FloorPlan

CRC Modeling

Class: FloorPlan	
Description	
Responsibility	**Collaborator**
Defines floor plan name/type	
Manages floor plan positioning	
Scales floor plan for display	
Incorporates walls, doors, windows	Wall
Shows position of video cameras	Camera

A CRC model index card for FloorPlan class

Class Responsibilities

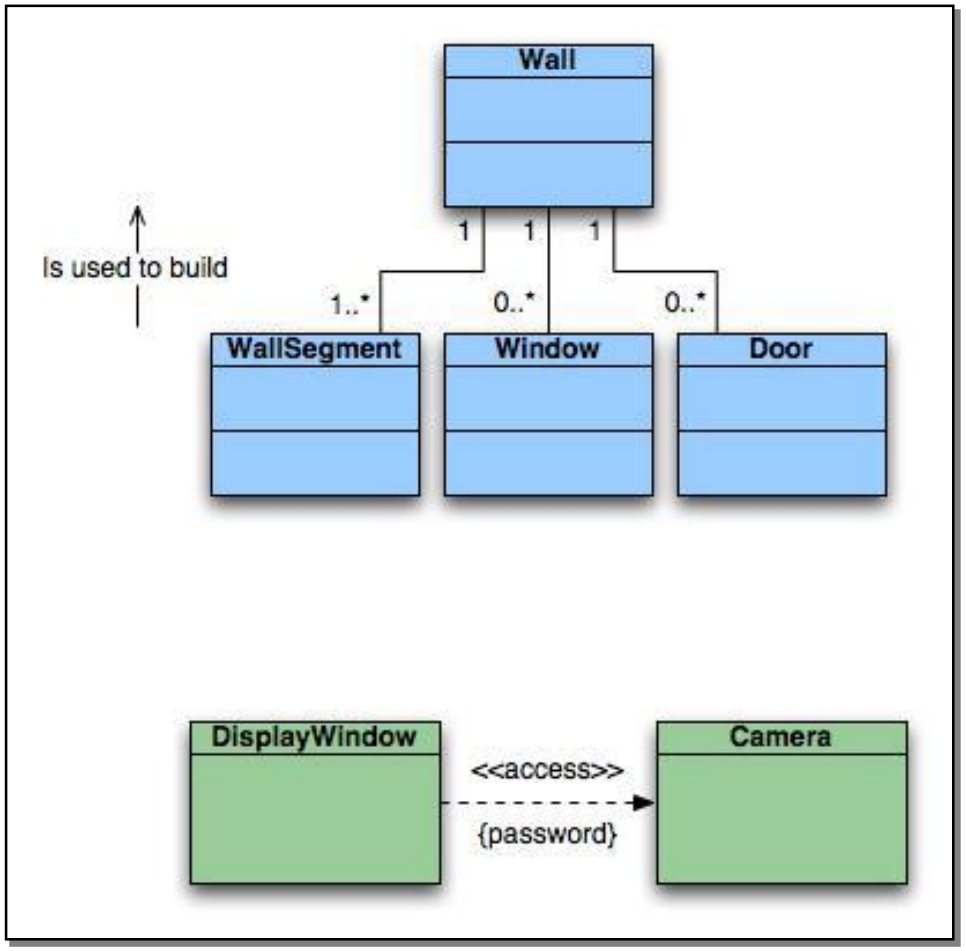
- Distribute system intelligence across classes.
- State each responsibility as generally as possible.
- Put information and the behavior related to it in the same class.
- Localize information about one thing rather than distributing it across multiple classes.
- Share responsibilities among related classes, when appropriate.

– Wirfs-Brock, et al.

Class Collaborations

- Relationships between classes:
  - is-part-of — used when classes are part of an aggregate class.
  - has-knowledge-of — used when one class must acquire information from another class.
  - depends-on — used in all other cases.

Association & Dependency
in Class Diagrams



Top: Example for Association with Multiplicity

Bottom: Example for Dependency

Behavioral Modeling

Creating a Behavioral Model

1) Identify events found within the use cases and implied by the attributes in the class diagrams
2) Build a state diagram for each class, and if useful, for the whole software system

Identifying Events in Use Cases

- An event occurs whenever an actor and the system exchange information
- An event is NOT the information that is exchanged, but rather the fact that information has been exchanged
- Some events have an explicit impact on the flow of control, while others do not
  - An example is the reading of a data item from the user versus comparing the data item to some possible value

Identifying Events

- A use-case is examined for *points of information exchange*.
- The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password stored in the system. If the password in incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further action.

The States of a System

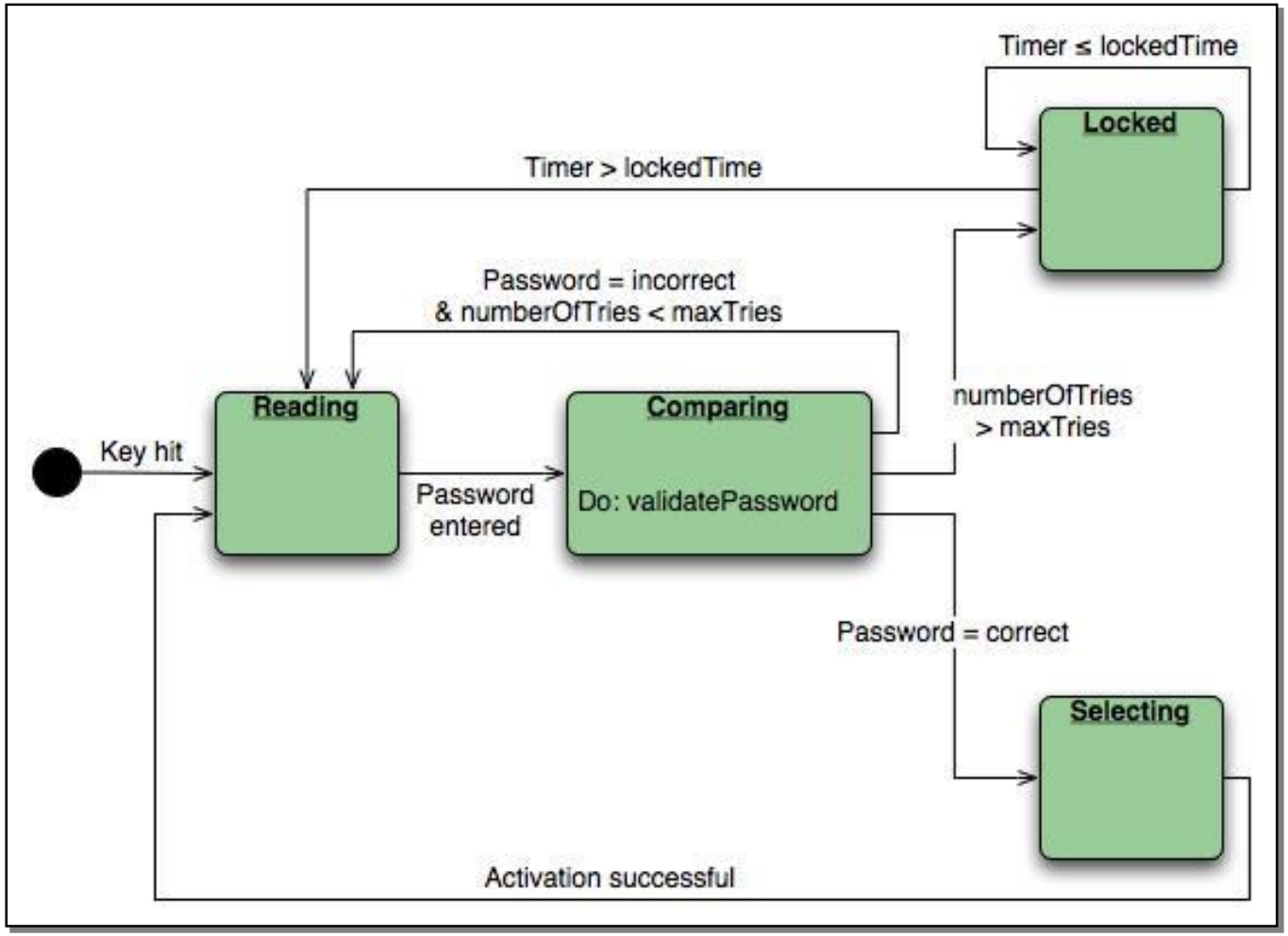
State—a set of observable circumstances that characterizes the behaviour of a system at a given time

State transition—the movement from one state to another

Event—an occurrence that causes the system to exhibit some predictable form of behavior

Action—process that occurs as a consequence of making a transition

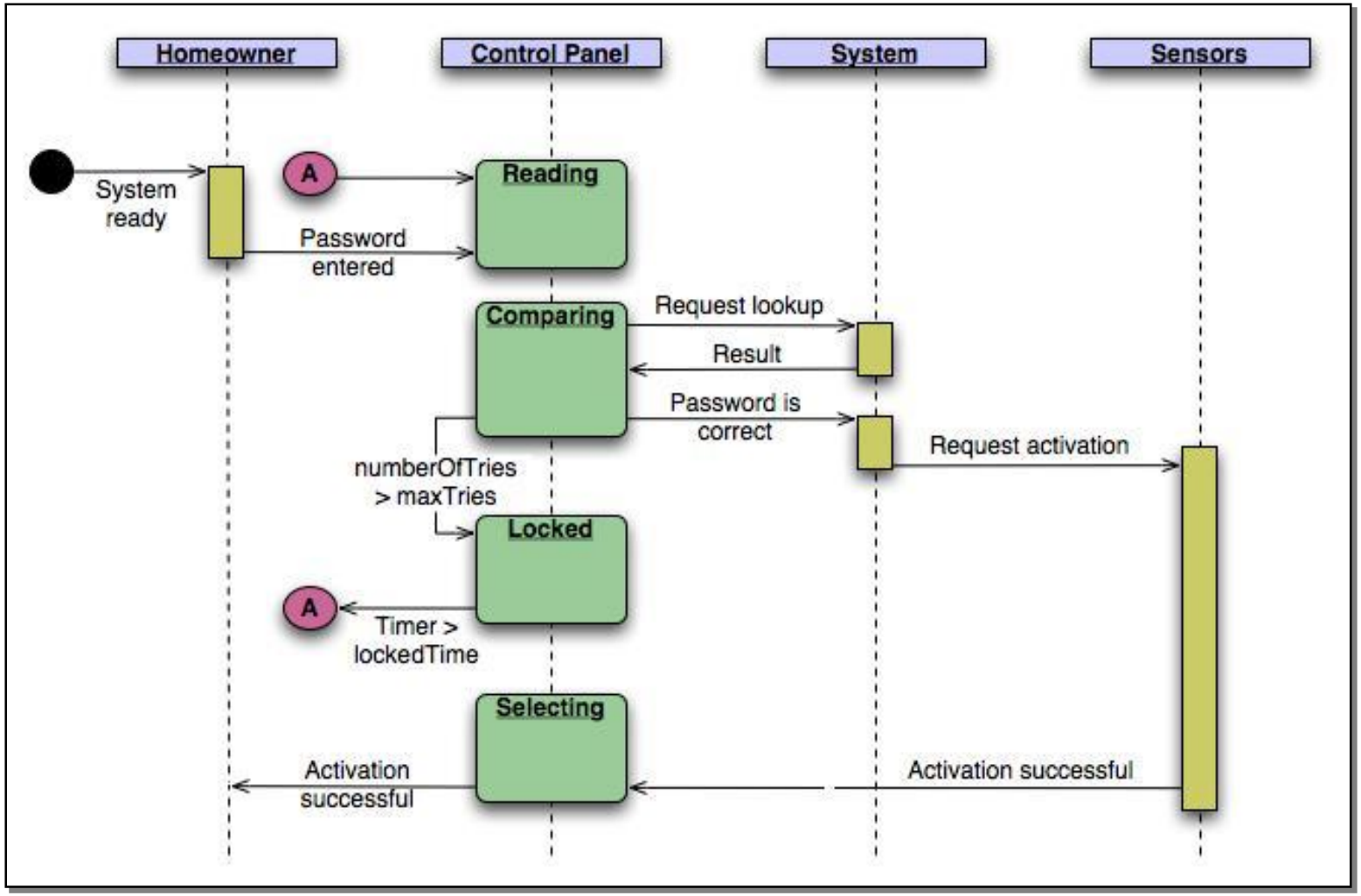
State Diagram



State diagram for the ControlPanel class

A variant of

Sequence Diagram



Sequence diagram (partial) for the *SafeHome* security function

Requirements Modeling for WebApps

Content Analysis. The full spectrum of content to be provided by the WebApp is identified, including text, graphics and images, video, and audio data. Data modeling can be used to identify and describe each of the data objects.

Interaction Analysis. The manner in which the user interacts with the WebApp is described in detail. Use-cases can be developed to provide detailed descriptions of this interaction.

Functional Analysis. The usage scenarios (use-cases) created as part of interaction analysis define the operations that will be applied to WebApp content and imply other processing functions. All operations and functions are described in detail.

Navigation Model. Defines the overall navigation strategy for the WebApp

Configuration Analysis. The environment and infrastructure in which the WebApp resides are described in detail.

When Do We Perform Analysis?

In some Web situations, analysis and design merge. However, an explicit analysis activity occurs when …

- the WebApp to be built is large and/or complex
- the number of stakeholders is large
- the number of Web engineers and other contributors is large
- the goals and objectives (determined during formulation) for the WebApp will effect the business' bottom line
- the success of the WebApp will have a strong bearing on the success of the business

The Content Model

Content objects are extracted from use-cases

- examine the scenario description for direct and indirect references to content

Attributes of each content object are identified

The relationships among content objects and/or the hierarchy of content maintained by a WebApp

- Relationships—entity-relationship diagram or UML
- Hierarchy—data tree or UML

The Interaction Model

- Composed of four elements:
  - use-cases
  - sequence diagrams
  - state diagrams
  - a user interface prototype

The Functional Model

The functional model addresses two processing elements of the WebApp

- user observable functionality that is delivered by the WebApp to end-users
- the operations contained within analysis classes that implement behaviors associated with the class.

An activity diagram can be used to represent processing flow

The Configuration Model

Server-side

- Server hardware and operating system environment must be specified
- Interoperability considerations on the server-side must be considered
- Appropriate interfaces, communication protocols and related collaborative information must be specified

Client-side

- Browser configuration issues must be identified
- Testing requirements should be defined

Navigation Modeling-I

- Should certain elements be easier to reach (require fewer navigation steps) than others? What is the priority for presentation?
- Should certain elements be emphasized to force users to navigate in their direction?
- How should navigation errors be handled?
- Should navigation to related groups of elements be given priority over navigation to a specific element.
- Should navigation be accomplished via links, via search based access, or by some other means?
- Should certain elements be presented to users based on the context of previous navigation actions?
- Should a navigation log be maintained for users?

Navigation Modeling-II

- Should a full navigation map or menu (as opposed to a single “back” link or directed pointer) be available at every point in a user’s interaction?
- Should navigation design be driven by the most commonly expected user behaviors or by the perceived importance of the defined WebApp elements?
- Can a user “store” his previous navigation through the WebApp to expedite future usage?
- For which user category should optimal navigation be designed?
- How should links external to the WebApp be handled? overlaying the existing browser window? as a new browser window? as a separate frame?

Domain requirements

(as per Sommerville)

- The system's operational domain imposes requirements on the system.
  - For example, a train control system has to take into account the braking characteristics in different weather conditions.
- Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

Train protection system
(as per Sommerville)

- This is a domain requirement for a train protection system:
- The deceleration of the train shall be computed as:
  - Dtrain = Dcontrol + Dgradient
  - where Dgradient is 9.81ms2 * compensated gradient/alpha and where the values of 9.81ms2 /alpha are known for different types of train.
- It is difficult for a non-specialist to understand the implications of this and how it interacts with other requirements.

Domain requirements problems
(as per Sommerville)

- Understandability
  - Requirements are expressed in the language of the application domain;
  - This is often not understood by software engineers developing the system.
- Implicitness
  - Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

Analysis Patterns

•Software patterns are a mechanism for capturing domain knowledge in a way that allows it to be reapplied when a new problem is encountered

— domain knowledge can be applied to a new problem within the same application domain

—the domain knowledge captured by a pattern can be applied by analogy to a completely different application domain.

• The original author of an analysis pattern does not “create” the pattern, but rather, *discovers it as* requirements engineering work is being conducted.

• Once the pattern has been discovered, it is documented

The structure of a requirements document

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

The structure of a requirements document

Chapter	Description
System requirements specification	This should describe the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Ways to specify system requirements

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

Summary

- The requirements model is the first technical representation of a system.
- Requirements modeling process uses a combination of text and diagrams to represent software requirements (data, function, and behavior) in an understandable way.
- Software engineers build requirements models using requirements elicited from customers. Building analysis models helps to make it easier to uncover requirement inconsistencies and omissions.
- Developer insights into software requirements grows in direct proportion to the number of different representations used in modeling
- Requirements modeling work products must be reviewed for correctness, completeness, consistency, and relevancy to stakeholder needs.