## LESSON 9

## Requirement Analysis Concepts and Modelling

**Learning Outcomes:**

After successful completion of this lesson, you should be able to:

* Explain the requirement analysis concepts in relation to requirements models.
* Discuss the elements of requirement models.
* Identify the types and elements of the requirement models.
* Discuss the approaches to requirement analysis.

.**Course Materials:**

**Requirements Analysis and Requirement Models**

Requirements analysis results in the specification of software’s operational characteristics, indicates software’s interface with other system elements, and establishes constraints that software must meet.

Requirements analysis allows elaborating on basic requirements established during the inception, elicitation, and negotiation tasks that are part of requirements engineering.

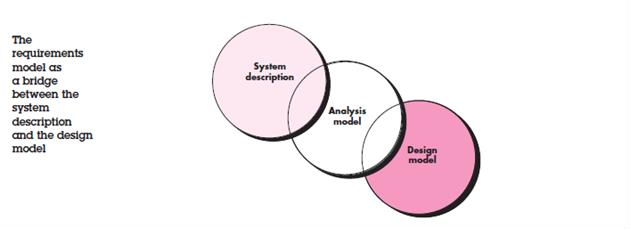
Requirement models provide a software designer with information that can be translated to architectural, interface, and component-level designs.

The requirements model (and the software requirements specification) provides the developer and the customer with the means to assess quality once software is built.

**The Requirement Models Objectives**

The requirements model must achieve 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.



**Requirements Modeling Approaches**

**Structured Analysis**

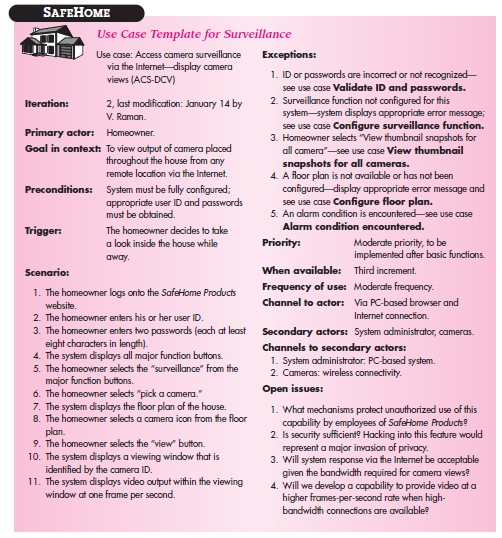
Considers data and the processes that transform the data as separate entities. Data objects are modelled in a way that defines their attributes and relationships. Processes that manipulate data objects are modeled in a manner that shows how they transform data as data objects flow through the system.

**Object-Oriented Analysis**

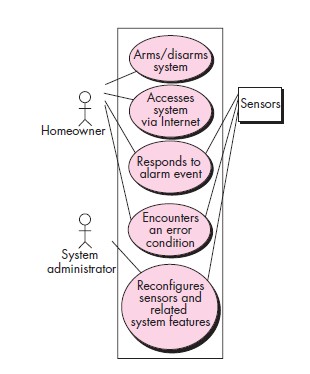
Focuses on the definition of classes and the manner in which they collaborate with one another to effect customer requirements. UML and the Unified Process are predominantly object oriented.

**Types of Requirement Models**

* **Scenario-based models of requirements from the point of view of various system “actors”.**
* Scenario-based models depict software requirements from the user’s point of view.
* The use case—a narrative or template-driven description of an interaction between an actor and the software—is the primary modeling element. See use case template and diagram below.

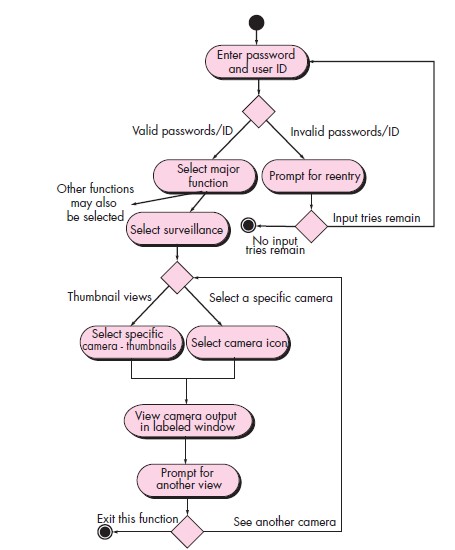


Use Case Diagram

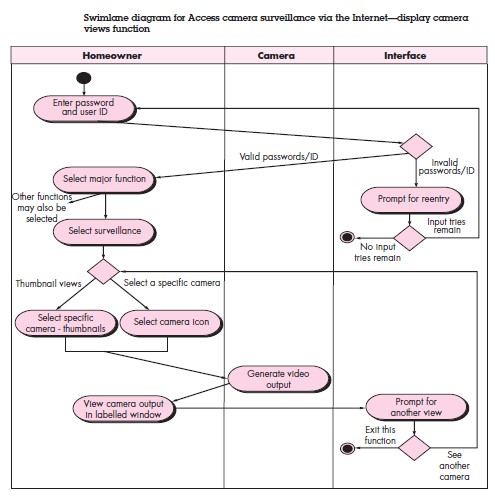


* Derived during requirements elicitation, the use case defines the keys steps for a specific function or interaction.
* The degree of use-case formality and detail varies, but the end result provides necessary input to all other analysis modeling activities.
* Scenarios can also be described using an activity diagram—a flowchartlike graphical representation that depicts the processing flow within a specific scenario. Sample activity diagram for Access Camera Surveillance via the Internet- display camera views function

Activity diagram for Access Camera Surveillance via the Internet- display camera views function



* Swimlane diagrams illustrate how the processing flow is allocated to various actors or classes.



* **Data models that depict the information domain for the problem.**

If software requirements include the need to create, extend, or interface with a database or if complex data structures must be constructed and manipulated, the software team may choose to create a data model as part of overall requirements modeling.

The entity-relationship diagram (ERD) addresses these issues and represents all data objects that are entered, stored, transformed, and produced within an application.

Data modeling is used to describe the information space that will be constructed or manipulated by the software.

Data modeling begins by representing data objects—composite information that must be understood by the software.

The attributes of each data object are identified and relationships between data objects are described.

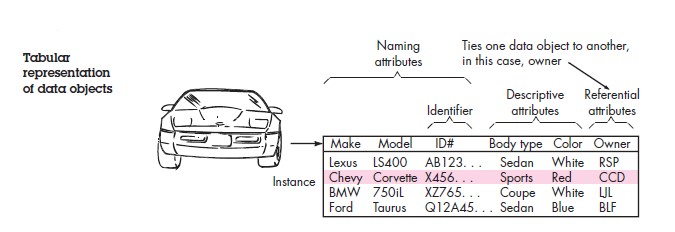
**Data Object**

Data object is a representation of composite information that must be understood by software.

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) or event (e.g., an alarm), a role (e.g., salesperson), an organizational un it (e.g., accounting department), a place (e.g., a warehouse), or a structure (e.g., a file). For example, a person or a car can be viewed as a data object in the sense that either can be defined in terms of a set of attributes

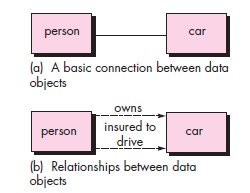
**Data Attributes**

Data attributes define the properties of a data object and take on one of three different characteristics. They can be used to (1) name an instance of the data object, (2) describe the instance, or (3) make reference to another instance in another table



**Relationships**

Data objects are connected to one another in different ways. Relationships indicate the manner in which data objects are connected to one another.



**Entity-Relationship Diagrams**

The object-relationship pair is the cornerstone of the data model. These pairs can be represented graphically using the entity-relationship diagram (ERD).

The ERD was originally proposed by Peter Chen for the design of relational database systems and has been extended by others.

A set of primary components is identified for the ERD: data objects, attributes, relationships, and various type indicators. The primary purpose of the ERD is to represent data objects and their relationships.

Rudimentary ERD notation has already been introduced. Data objects are represented by a labelled rectangle. Relationships are indicated with a labeled line connecting objects. In some variations of the ERD, the connecting line contains a diamond that is labeled with the relationship. Connections between data objects and relationships are established using a variety of special symbols that indicate cardinality and modality.

* **Class-oriented models that represent object-oriented classes (attributes and operations) and the manner in which classes collaborate to achieve system requirements.**

Class-based modeling represents the objects that the system will manipulate, the operations (also called methods or services) that will be applied to the objects to effect the manipulation, relationships (some hierarchical) between the objects, and the collaborations that occur between the classes that are defined.

The elements of a class-based model include classes and objects, attributes, operations, class responsibility-collaborator (CRC) models, collaboration diagrams, and packages.

Class-based modeling uses information derived from scenario-based and data modeling elements to identify analysis classes.

A grammatical parse may be used to extract candidate classes, attributes, and operations from text-based narratives.

Criteria for the definition of a class are defined. A set of class-responsibility collaborator index cards can be used to define relationships between classes.

In addition, a variety of UML modeling notation can be applied to define hierarchies, relationships, associations, aggregations, and dependencies among classes.

Analysis packages are used to categorize and group classes in a manner that makes them more manageable for large systems.

**Identifying Analysis Classes**

* Examining the usage scenarios developed as part of the requirements model
* Performing a “grammatical parse” on the use cases developed for the system to be built.
* Classes are determined by underlining each noun or noun phrase and entering it into a simple table.
* Synonyms should be noted.
  + If the class (noun) is required to implement a solution, then it is part of the solution space
  + if a class is necessary only to describe a solution, it is part of the problem space.

**Sample Class Categorization for Analysis Class**

* External entities (e.g., other systems, devices, people) that produce or consume information to be used by a computer-based system.
* 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 of the system.
* Structures (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects.

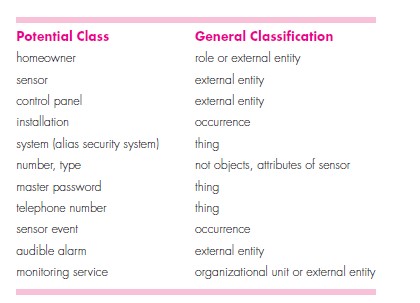
**How to Define Classes**

Grammatical parsing (nouns are underlined, verbs italicized) is done for a processing narrative for the SafeHome security function.

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 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 . . .

Extracting the nouns, we can propose a number of potential classes:



**Six Selection Characteristics that should be Used to Consider Each Potential Class for Inclusion in The Analysis Model:**

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.

**Specifying Attributes**

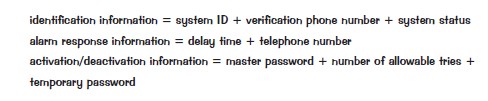
Attributes describe a class that has been selected for inclusion in the requirements model.

To develop a meaningful set of attributes for an analysis class, you should study each use case and select those “things” that reasonably “belong” to the class. In addition, the following question should be answered for each class**: “What data items (composite and/or elementary) fully define this class in the context of the problem at hand?”**

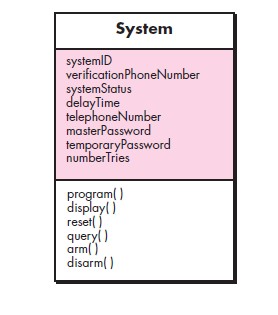
Illustration:

System class defined for SafeHome.

A homeowner can configure the security function to reflect sensor information, alarm response information, activation/deactivation information, identification information, and so forth. We can represent these composite data items in the following manner:



**Class Diagram for the System Class**



**Defining Operations**

Operations define the behavior of an object.

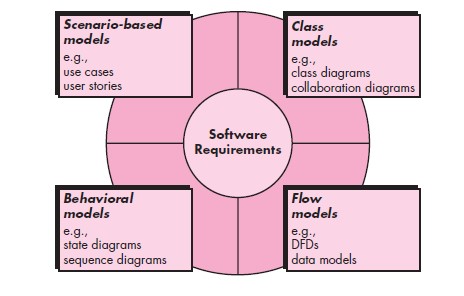
It is divided into four broad categories:

* operations that manipulate data in some way (e.g., adding, deleting, reformatting, selecting)
* operations that perform a computation
* operations that inquire about the state of an object
* operations that monitor an object for the occurrence of a controlling event.

* Flow-oriented models that represent the functional elements of the system and how they transform data as it moves through the system.

* Behavioral models that depict how the software behaves as a consequence of external “events”.

**Elements of Analysis Model**



**Read:**

Chapter 6 Software Engineering, Practitioner Approach 7th Edition Roger S. Pressman Ph.D.

**Activities/Assessment:**

1. What are the objectives of requirements models?
2. What are the steps to identify analysis classes?
3. What are the types of requirements models?

**Answer:**

1. 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.
2. Examine the usage scenarios developed as part of the requirements model, Perform a

“grammatical parse” on the use cases developed for the system to be built, Underline each noun or noun phrase and entering it into a simple table, Note the synonyms.

1. Scenario-based, class models, behavioural models and flow models

**Problem:**

What different points of view can be used to describe the requirements model?

Is it possible to begin coding immediately after an analysis model has been created? Explain your answer and then argue the counterpoint.