UNDERSTANDING REQUIREMENTS
Module 2- Chapter 1

Dr. Vinod Kumar P
Associate Professor
Dept of CSE-Data Science
ATMECE

Introduction to requirement engineering

- The process of collecting the software requirement from the client then understand, evaluate and document it is called as **requirement engineering**.
- Requirement engineering constructs a bridge for design and construction.

Requirements Engineering Task

Requirement engineering consists of seven different tasks as follow:

1. Inception

- Inception is a task where the requirement engineering asks a set of questions to establish a software process.
- In this task, it understands the problem and evaluates with the proper solution.
- It collaborates with the relationship between the customer and the developer.
- The developer and customer decide the overall scope and the nature of the question.

2. Elicitation

Elicitation means to find the requirements from anybody.

The requirements are difficult because the following problems occur in elicitation.

Problem of scope: The customer give the unnecessary technical detail rather than clarity of the overall system objective.

Problem of understanding: Poor understanding between the customer and the developer regarding various aspect of the project like capability, limitation of the computing environment.

Problem of volatility: In this problem, the requirements change from time to time and it is difficult while developing the project**.**

3. Elaboration

In this task, the information taken from user during inception and elicitation are expanded and refined in elaboration.

Its main task is developing pure model of software using functions, feature and constraints of a software.

4. Negotiation

In negotiation task, a software engineer decides the how will the project be achieved with limited business resources.

To create rough guesses of development and access the impact of the requirement on the project cost and delivery time.

5. Specification

In this task, the requirement engineer constructs a final work product.

The work product is in the form of software requirement specification.

In this task, formalize the requirement of the proposed software such as informative, functional and behavioral.

The requirement are formalize in both graphical and textual formats.

6. Validation

- The work product is built as an output of the requirement engineering and that is accessed for the quality through a validation step.
- The formal technical reviews from the software engineer, customer and other stakeholders helps for the primary requirements validation mechanism.

7. Requirement management

- It is a set of activities that help the project team to identify, control and track the requirements and changes can be made to the requirements at any time of the ongoing project.
- These tasks start with the identification and assign a unique identifier to each of the requirement.
- After finalizing the requirement traceability table is developed.
- The examples of traceability table are the features, sources, dependencies, subsystems and interface of the requirement.

ESTABLISHING THE GROUNDWORK

The steps required to establish the groundwork for an understanding of software requirements, to get the project started in a way that will keep it moving forward toward a successful solution.

Identifying Stakeholders: Stakeholder is "anyone who benefits in a direct or indirect way from the system which is being developed." The usual stakeholders are: business operations managers, product managers, marketing people, internal and external customers, end users,consultants, product engineers, software engineers, support and maintenance engineers.

Recognizing Multiple Viewpoints: Because many different stakeholders exist, the requirements of the system will be explored from many different points of view. Each of these constituencies will contribute information to the requirements engineering process.

Should categorize all stakeholder information in a way that will allow decision makers to choose an internally consistent set of requirements for the system

Working toward Collaboration: If five stakeholders are involved in a software project, you may have five different opinions about the proper set of requirements. Customers must collaborate among themselves and with software engineering practitioners if a successful system is to result.

Asking the First Questions: Questions asked at the inception of the project should be "context free. The first set of context-free questions focuses on the customer and other stakeholders, the overall project goals and benefits. You might ask:

- 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?

The next set of questions enables you to gain a better understanding of the problem and allows the customer to voice his or her perceptions about a solution:

- How would you characterize "good" output that would be generated by a successful solution?
- What problem(s) will this solution address?

The final set of questions focuses on the effectiveness of the communication activity itself.

- 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 questions?
- Can anyone else provide additional information?
- Should I be asking you anything else?

Eliciting Requirements

- Eliciting requirement helps the user for collecting the requirement

Eliciting requirement steps are as follows:

1. Collaborative requirements gathering

- Gathering the requirements by conducting the meetings between developer and customer.
- Fix the rules for preparation and participation.
- The main motive is to identify the problem, give the solutions for the elements, negotiate the different approaches and specify the primary set of solution requirements in an environment which is valuable for achieving goal.

2. Quality Function Deployment (QFD)

- In this technique, translate the customer need into the technical requirement for the software.
- QFD system designs a software according to the demands of the customer.

3. Usage scenarios

- Till the software team does not understand how the features and function are used by the end users it is difficult to move technical activities.
- To achieve above problem the software team produces a set of structure that identify the usage for the software.
- This structure is called as 'Use Cases'.

4. Elicitation work product

- The work product created as a result of requirement elicitation that is depending on the size of the system or product to be built.
- The work product consists of a statement need, feasibility, statement scope for the system.
- It also consists of a list of users participate in the requirement elicitation.

Developing Use cases

- In software and systems engineering, a use case is a list of actions or event steps, typically defining the interactions between a role (known as an actor) and a system, to achieve a goal.
- The actor can be a human, an external system, or time.
- A use case describes a way in which a real-world actor interacts with the system.

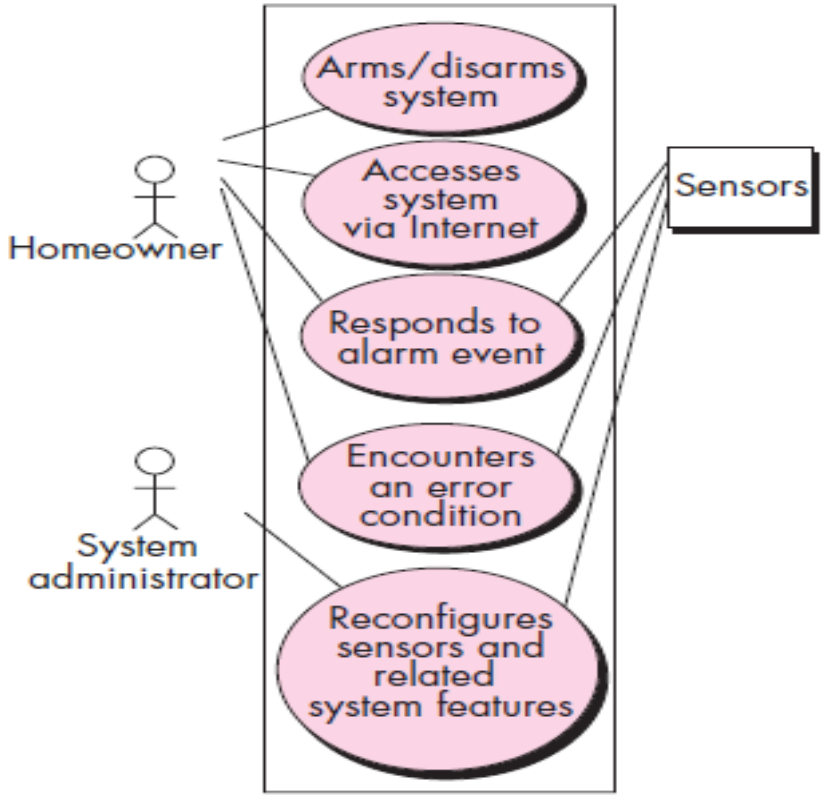
The advantages of Use cases includes:

- The list of goal names provides the shortest summary of what the system will offer
- It gives an overview of the roles of each and every component in the system. It will help us in defining the role of users, administrators etc.
- It provides solutions and answers to many questions that might pop up if we start a project unplanned.

Parts of Use Cases

- **Use Case**: What is the main objective of this use case. For eg. Adding a software component, adding certain functionality etc.
- **Primary Actor**: Who will have the access to this use case. In the above examples, administrators will have the access.
- **Scope**: Scope of the use case
- **Level**: At what level the implementation of the use case be.
- **Flow**: What will be the work flow of the use case.
- Some other things that can be included in the use cases are:
- Preconditions
- Post conditions
- Brief course of action
- Time Period

UML use case diagram for *SafeHome* home security function



Building the Requirement Model

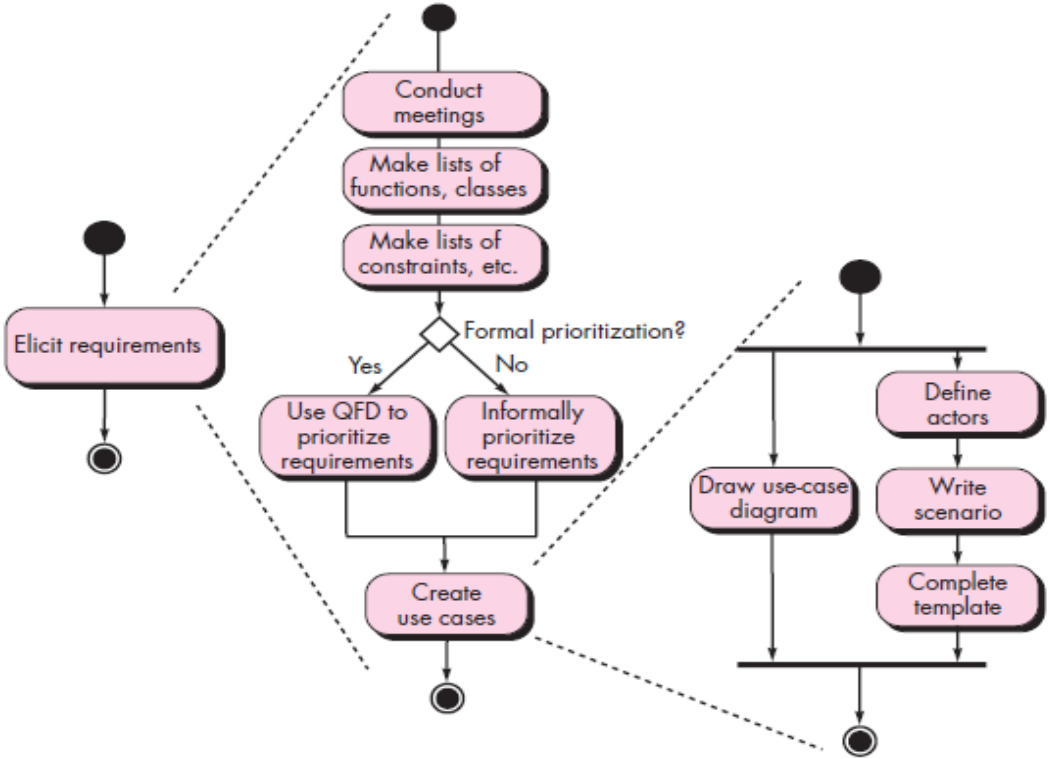
The intent of the analysis model is to provide a description of the required informational, functional, and behavioral domains for a computer-based system.

The model changes dynamically.

Elements of the Requirements Model: There are many different ways to look at the requirements for a computer-based system.

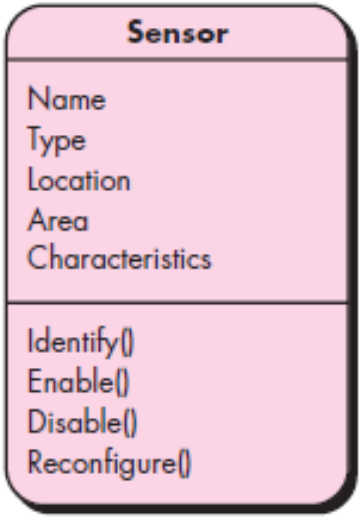
1. **Scenario-based elements.**

- The system is described from the user's point of view using a scenario-based approach.
- Scenario-based elements of the requirements model are often the first part of the model that is developed. Three levels of elaboration are shown


Conduct meetings
Make lists of functions, classes
Make lists of constraints, etc.
Formal prioritization?
Yes
No
Use QFD to prioritize requirements
Informally prioritize requirements
Elicit requirements
Create use cases
Define actors
Draw use-case diagram
Write scenario
Complete template

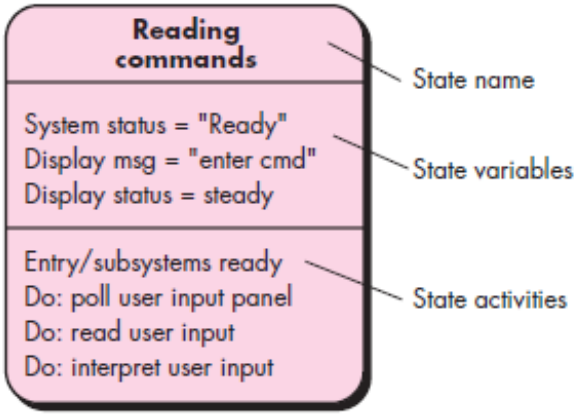
2. Class-based elements.

- Each usage scenario implies a set of objects that are manipulated as an actor interacts with the system.
- These objects are categorized into classes—a collection of things that have similar attributes and common behaviors. Example shown below



3. Behavioral elements.

- The behavior of a computer-based system can have a profound effect on the design that is chosen and the implementation approach that is applied.
- Therefore, the requirements model must provide modeling elements that depict behavior.
- The state diagram is one method for representing the behavior of a system.



4. Flow-oriented elements

- Information is transformed as it flows through a computer-based system.
- The system accepts input in a variety of forms, applies functions to transform it, and produces output in a variety of forms.
- The transform(s) may comprise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system.

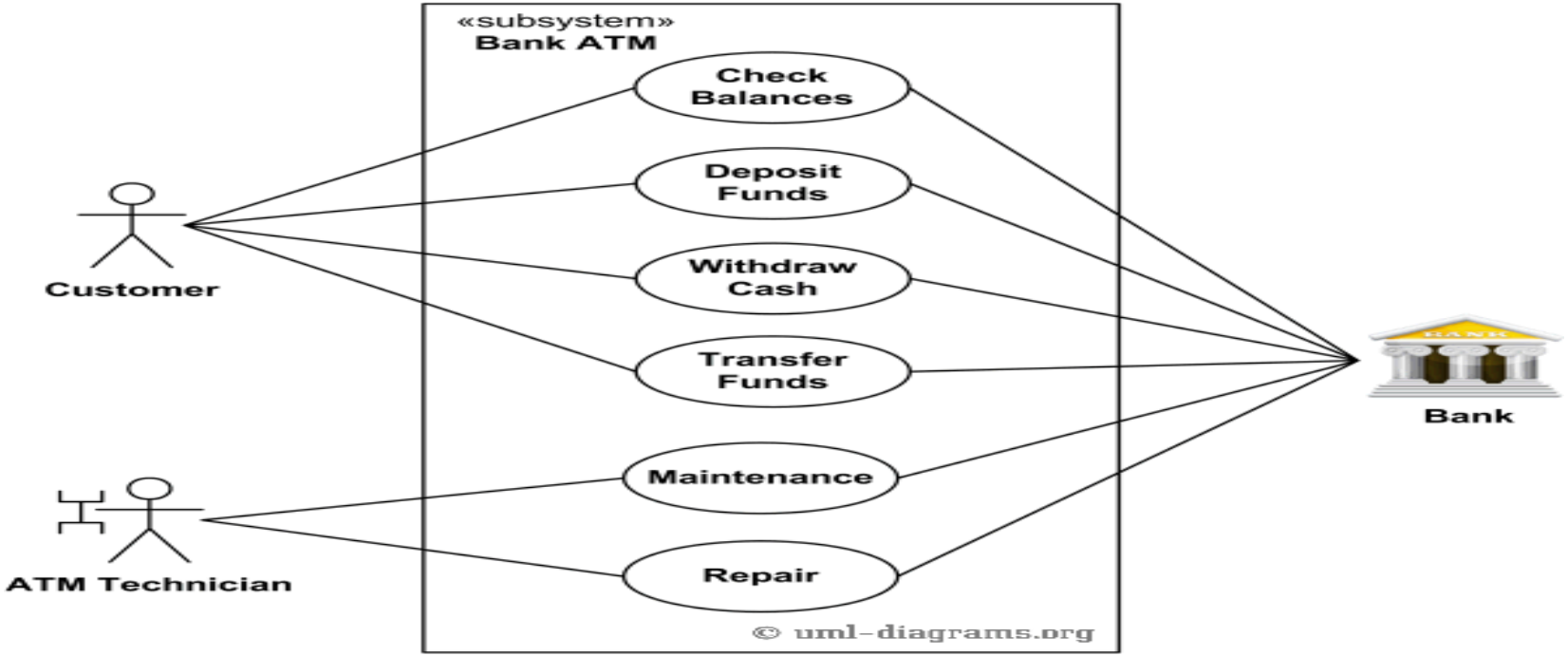
Analysis Patterns:

- Anyone who has done requirements engineering on more than a few software projects , notice that certain problems reoccur across all projects.
- These analysis patterns suggest solutions (e.g., a class, a function, a behavior) within the domain that can be reused when modeling many apps. Analysis patterns are integrated into the analysis model by pattern name. They are also stored in a repository so that requirements engineers can search and apply them.

Use Case

- Use-cases are a scenario based technique in the UML which identify the actors in an interaction and which describe the interaction itself.
- A set of use cases should describe all possible interactions with the system.
- High-level graphical model supplemented by more detailed tabular description .
- Sequence diagrams may be used to add detail to use cases by showing the sequence of event processing in the system.

Use case


«subsystem»
Bank ATM
Check Balances
Deposit Funds
Withdraw Cash
Transfer Funds
Maintenance
Repair
Customer
ATM Technician
Bank
© uml-diagrams.org

Negotiating Requirement

Agree on a deliverable system that is realistic for developers and customers

- SW team & other project stakeholders negotiate the priority, availability, and cost of each requirement
- The Process are :

– Identify the key stakeholders

These are the people who will be involved in the negotiation

– Determine each of the stakeholders "win conditions"

Win conditions are not always obvious

– Negotiate

Work toward a set of requirements that lead to "win-win"

The Art of Negotiation

- Recognize that it is not competition
- Map out a strategy
- Listen actively
- Focus on the other party's interests
- Don't let it get personal
- Be creative
- Be ready to commit

Validation

- Examine the specification to ensure that SW requirement is not ambiguous, consistent, error free etc .
- 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.

- Validation: “Am I building the right product?” checking a work product against higher-level work products or authorities that frame this particular product. – Requirements are validated by stakeholders

. Verification: “Am I building the product right?” checking a work product against some standards and conditions imposed on this type of product and the process of its development. – Requirements are verified by the analysts mainly

REQUIREMENTS MODELLING SCENARIOS

Module 2- Chapter 2

What is Requirements modeling?

- Requirements modeling uses a combination of **text** and **diagrammatic** forms to depict requirements in a way that is relatively easy to understand.

Why is it important?

- To validate software requirements, you need to examine them from a number of different points of view.
- In this chapter we'll consider requirements modeling from three different perspectives: **scenario based** models, **data(information)** models, and **class-based** models.
- Each represents requirements in a different "dimension," thereby increasing the probability that errors will be found, that inconsistency will surface, and that omissions will be uncovered.

REQUIREMENTS ANALYSIS

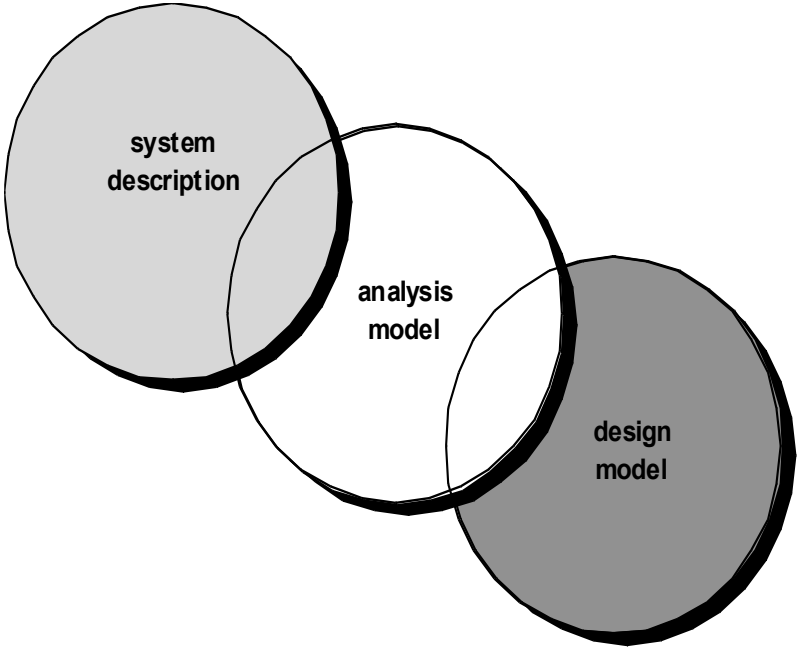
Requirements analysis results in the specification of software's operational characteristics,indicates software's interface with other system elements, and establishes constraints that software must meet.

- ✓ Objectives
- ✓ Analysis Rules of Thumb
- ✓ Domain Analysis
- ✓ Requirement Modeling Approaches

A Bridge

The analysis model bridges the gap between a system-level description that describes overall system or business functionality and a software design that describes the software's application architecture, user interface, and component-level structure.

This relationship is illustrated in Figure .



Overall Objectives and Philosophy:

Throughout requirements modeling, the primary focus is on what, not how.

The requirements model must achieve three primary objectives:
(1) to describe what the customer requires
(2) to establish a basis for the creation of a software design
(3) to define a set of requirements that can be validated once the software is built.

Analysis Rules of Thumb

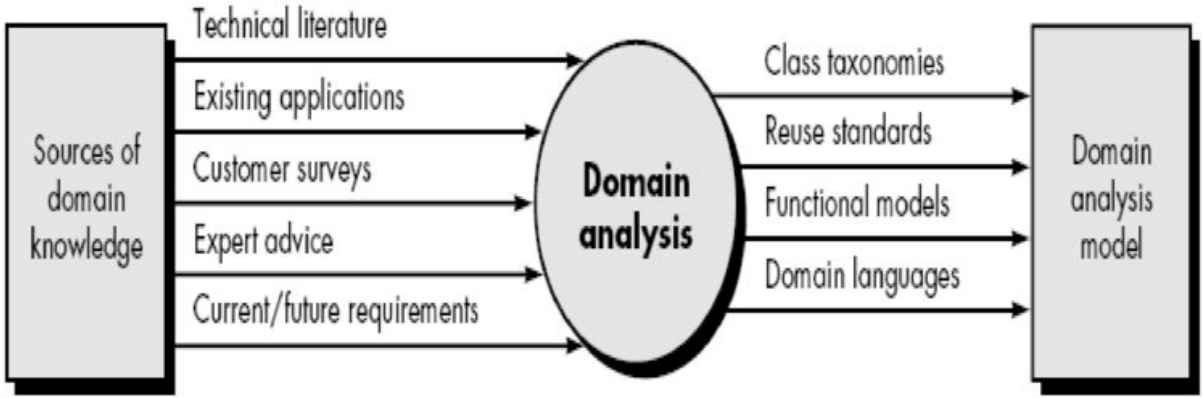
- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high.
- Each element of the analysis model should add to an overall understanding of software requirements and provide insight into the information domain, function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

Domain Analysis

As part of Requirement Analysis, this is done for software reuse:

- Can I use these designs for similar applications in the same domain?
- Define the domain to be investigated.
- Collect a representative sample of applications in the domain.
- Analyze each application in the sample.
- Develop an analysis model for the objects.

Input and Output of domain analysis

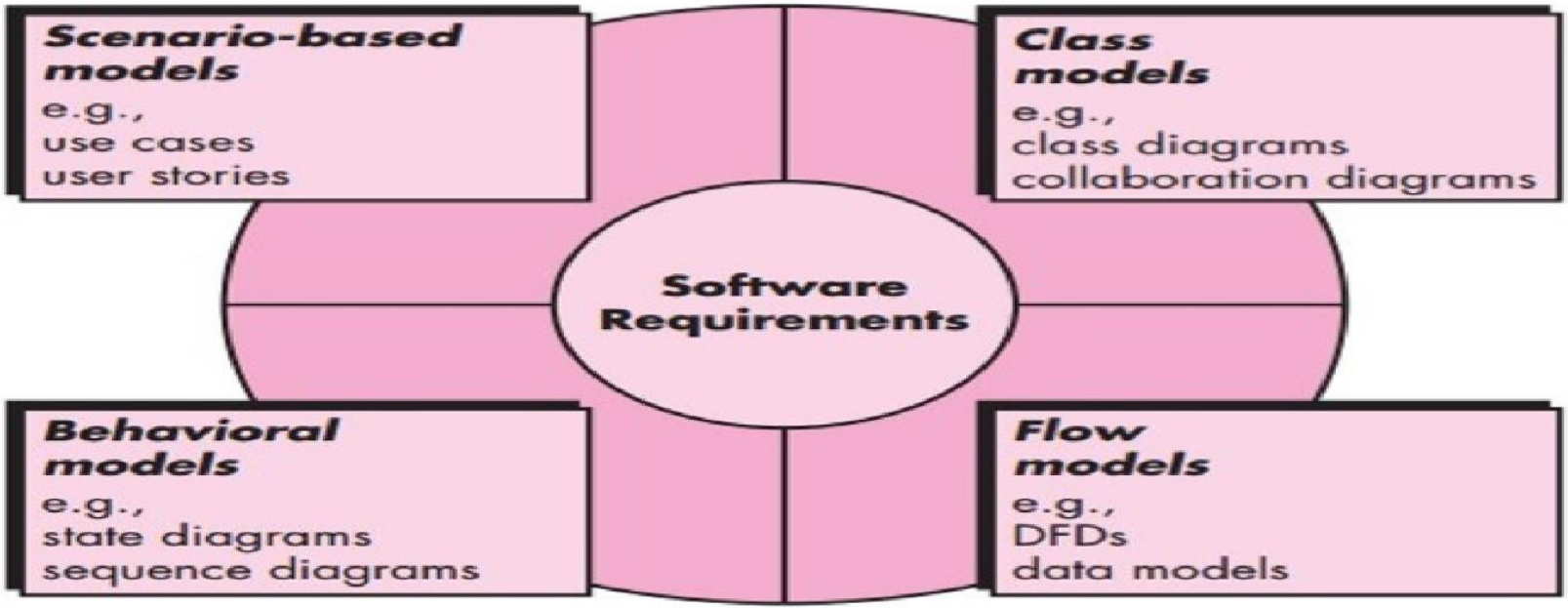


Requirements Modeling Approaches

- **Structured analysis**, considers data and the processes that transform the data as separate entities.
- Data objects 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.
- A second approach to analysis modeling, called **object-oriented analysis**, focuses on the definition of classes and the manner in which they collaborate with one another. **UML** and the Unified Process are predominantly object oriented.
- Each element of the requirements model (Next Figure) presents the problem from a different point of view.

Requirements Modeling Approaches

Elements of the analysis model



- **Scenario-based** elements depict how the user interacts with the system and the specific sequence of activities that occur as the software is used.
- **Class-based** elements model the objects that the system will manipulate, the operations that will be applied to the objects to effect the manipulation, relationships between the objects, and the collaborations that occur between the classes that are defined.
- **Behavioral elements** depict how external events change the state of the system or the classes that reside within it.
- Finally, **flow oriented** elements represent the system as an information transform, depicting how data objects are transformed as they flow through various system functions.

Analysis modeling leads to the derivation of each of these modeling elements

Scenario Based model

- Although the success of a computer-based system or product is measured in many ways, user satisfaction resides at the top of the list.
- If you understand how end users (and other actors) want to interact with a system, your software team will be better able to properly characterize requirements and build meaningful analysis and design models.
- Hence, requirements modeling with UML begins with the creation of scenarios in the form of use cases, activity 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 (use-cases)." Ivar Jacobson

(1) What should we write about?

(2) How much should we write about it?

(3) How detailed should we make our description?

(4) How should we organize the description?

What to Write About?

- Inception and elicitation—provide you with the information you'll need to begin writing use cases.
- Requirements gathering meetings and other requirements engineering mechanisms are used.
- To begin developing a set of use cases, list the functions or activities performed by a specific actor.

How Much to Write About?

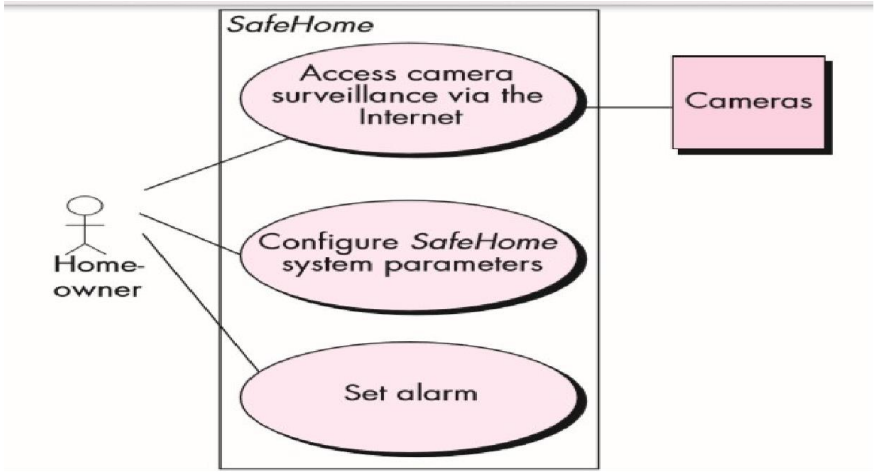
- As further conversations with the stakeholders progress, the requirements gathering team develops use cases for each of the functions noted.
- In general, use cases are written first in an informal narrative fashion.
- If more formality is required, the same use case is rewritten using a structured format similar to the one proposed.

Refining a Preliminary Use Case:

- What are the main tasks or functions that are performed by the actor?
- 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?

Writing a Formal Use Case:

The informal use cases presented are sometimes sufficient for requirements modeling. However, when a use case involves a critical activity or describes a complex set of steps with a significant number of exceptions, a more formal approach may be desirable.



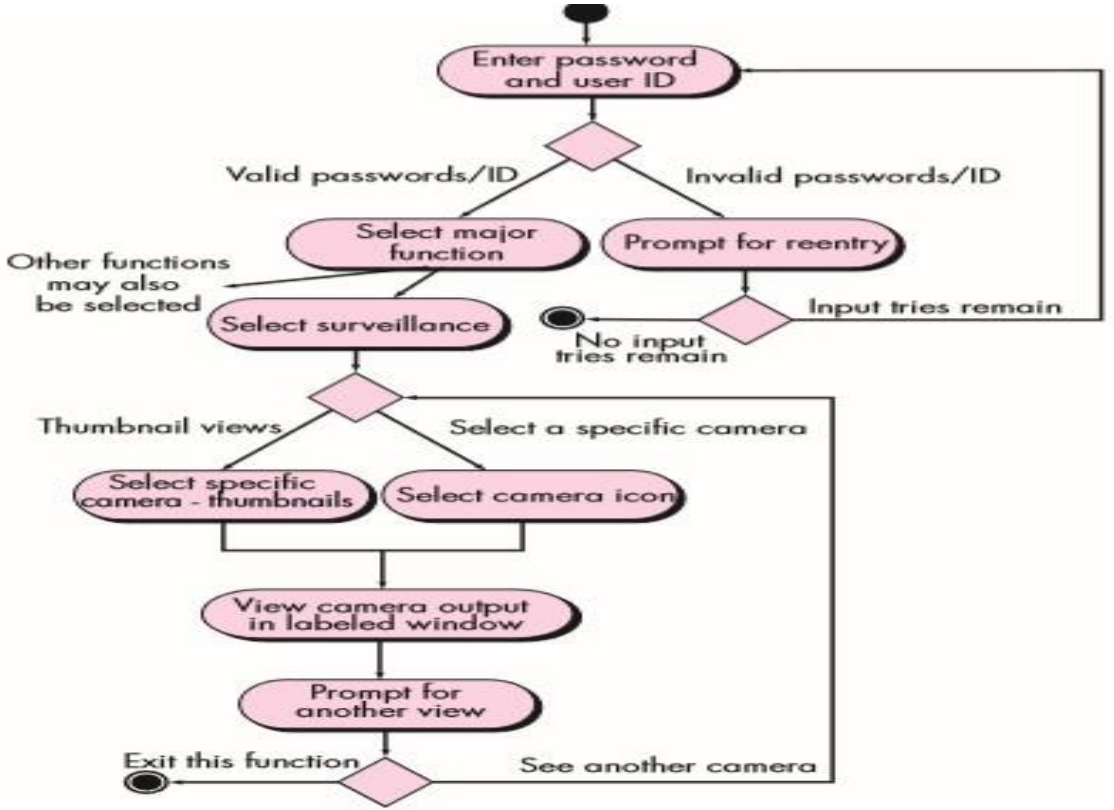
UML MODELS THAT SUPPLEMENT THE USE CASE

1. Developing an Activity Diagram:

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

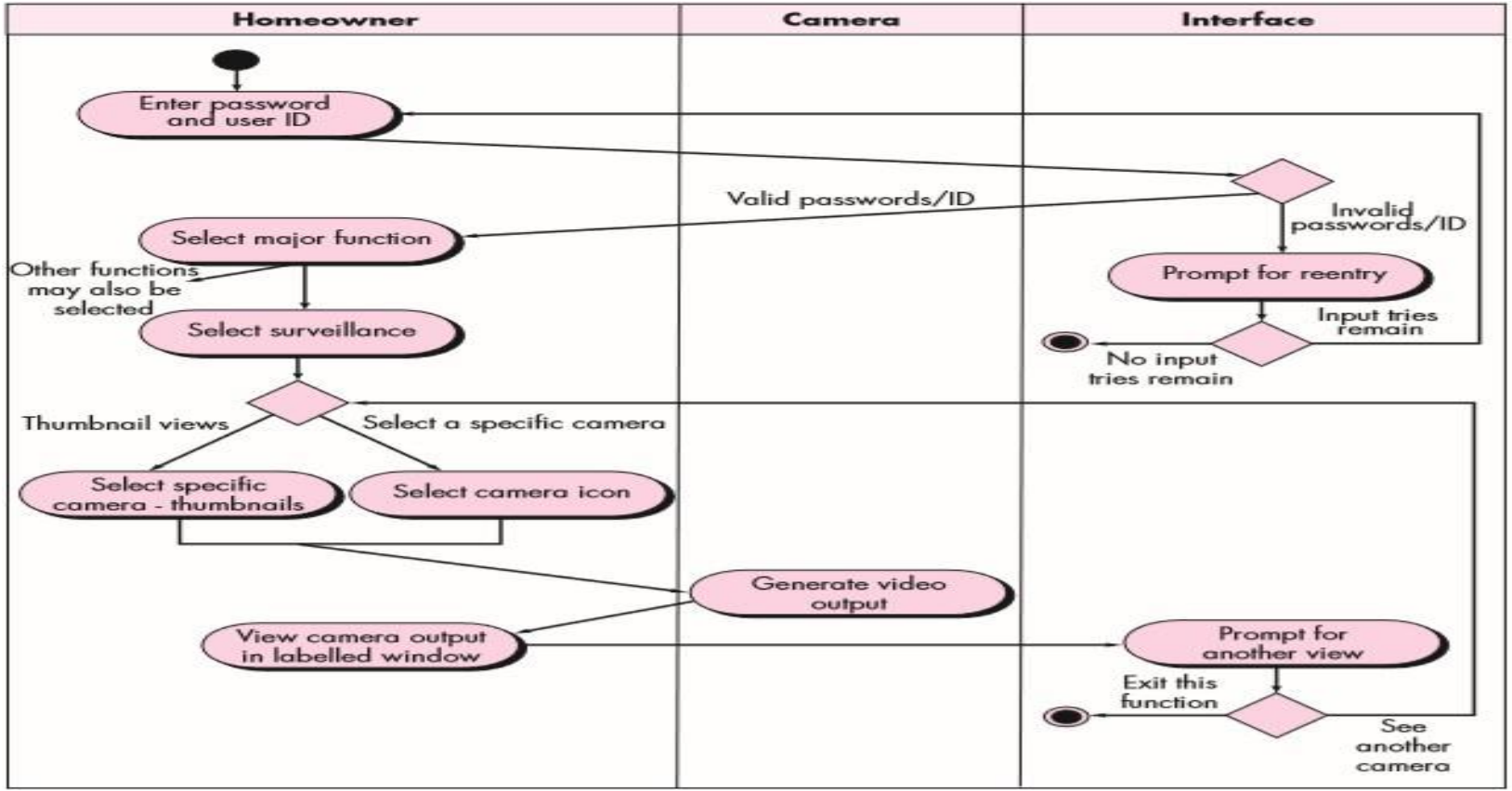
- Similar to the flowchart, an activity diagram uses rounded **rectangles** to imply a specific system function, **arrows** to represent flow through the system, decision **diamonds** to depict a branching decision and solid **horizontal lines** to indicate that parallel activities are occurring.

Activity diagram for Access camera surveillance via the Internet—display camera views function.



2. Swimlane Diagrams:

- The UML swimlane diagram is a useful variation of the activity diagram.
- Allows you to represent the flow of activities 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.
- Responsibilities are represented as parallel segments that divide the diagram vertically, like the lanes in a swimming pool.


Homeowner
Camera
Interface
Enter password and user ID
Valid passwords/ID
Invalid passwords/ID
Select major function
Other functions may also be selected
Prompt for reentry
Input tries remain
Select surveillance
No input tries remain
Thumbnail views
Select a specific camera
Select specific camera - thumbnails
Select camera icon
Generate video output
View camera output in labelled window
Prompt for another view
Exit this function
See another camera

DATA MODELING CONCEPTS

- If software requirements include the need to create, extend, or interface with a database or if complex data structures must be constructed and manipulated, then data model is used as part of overall requirements modeling.
- A software engineer or analyst defines all data objects that are processed within the system, the relationships between the data objects, and other information that is pertinent to the relationships.
- 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
    - Attributes
    - Relationship

- A **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 unit** (e.g., accounting department), **a place** (e.g., a warehouse), or **a structure** (e.g., a file)
- The **description** of the data object incorporates the data object and all of its **attributes**.
- A data object **encapsulates** data only—there is no reference within a data object to operations that act on the data.

Data Modeling

- examines data objects independently of processing
- focuses attention on the data domain
- creates a model at the customer's level of abstraction
- indicates how data objects relate to one another

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

(3) make reference to another instance in another table.

What is a Relationship?

- Data objects are connected to one another in different ways.
  - A connection is established between **person** and **car** because the two objects are related.
    - A person *owns* a car
    - A person *is insured to drive* a car
- The relationships *owns* and *insured to drive* define the relevant connections between **person** and **car.**
- Several instances of a relationship can exist
- Objects can be related in many different ways

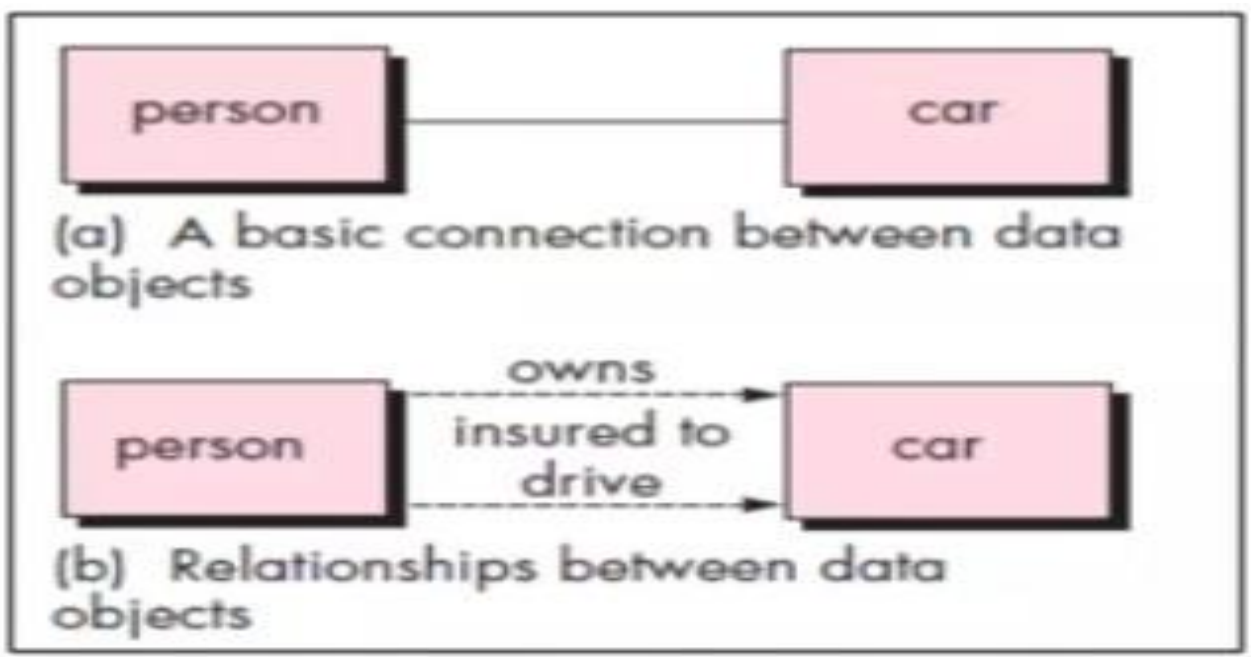


Figure. Relationships between data objects

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.

Identifying Analysis Classes

- Examining the usage scenarios developed as part of the requirements model and perform a "grammatical parse".
  - 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; otherwise, if a class is necessary only to describe a solution, it is part of the problem space.

But what should we look for once all of the nouns have been isolated?

Manifestations of Analysis Classes

- Analysis classes manifest themselves in one of the following ways:

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 (to be included in Analysis Model)

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

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

- 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.

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

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

- 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.

Defining Attributes

- Attributes describe a class that has been selected for inclusion in the analysis model.
- Build two different classes for professional baseball players.
  - For Playing Statistics software: name, position, batting average, fielding percentage, years played, and games played might be relevant.
  - For Pension Fund software: average salary, credit toward full vesting, pension plan options chosen, mailing address, and the like.

Defining Operations

- Do a grammatical parse of a processing narrative and look at the verbs.
- Operations can be divided into four broad categories:

(1)operations that manipulate data in some way (e.g., adding, deleting, reformatting, selecting)

(2) operations that perform a computation

(3) operations that inquire about the state of an object, and

(4) operations that monitor an object for the occurrence of a controlling event.

CRC Models

- Class-responsibility-collaborator (CRC) modeling provides a simple means for identifying and organizing the classes that are relevant to system or product requirements.

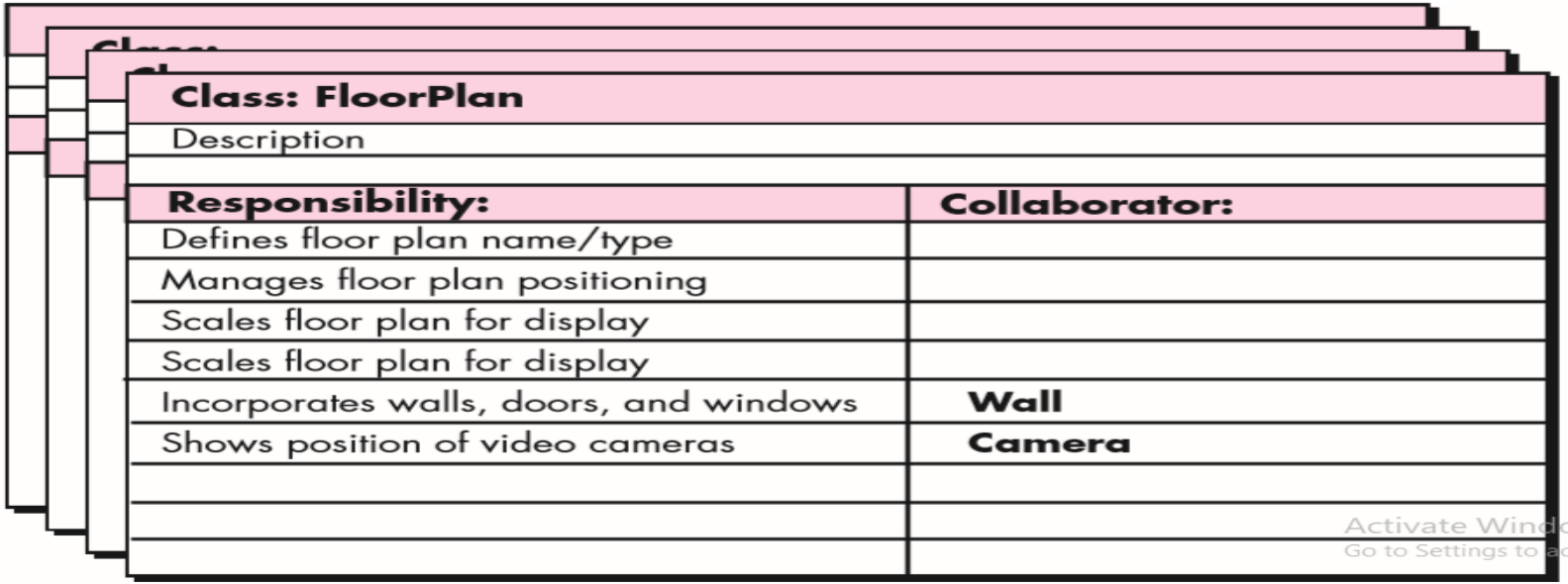
Ambler describes CRC modeling in the following way:

A CRC model is really a collection of standard index cards that represent classes.

The cards are divided into three sections. Along the top of the card you write the name of the class.

In the body of the card you list the class responsibilities on the left and the collaborators on the right.

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**

Class Types in CRC

- **Entity classes**, also called model or business classes, are extracted directly from the statement of the problem (e.g., Floor Plan 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. 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.

Responsibilities

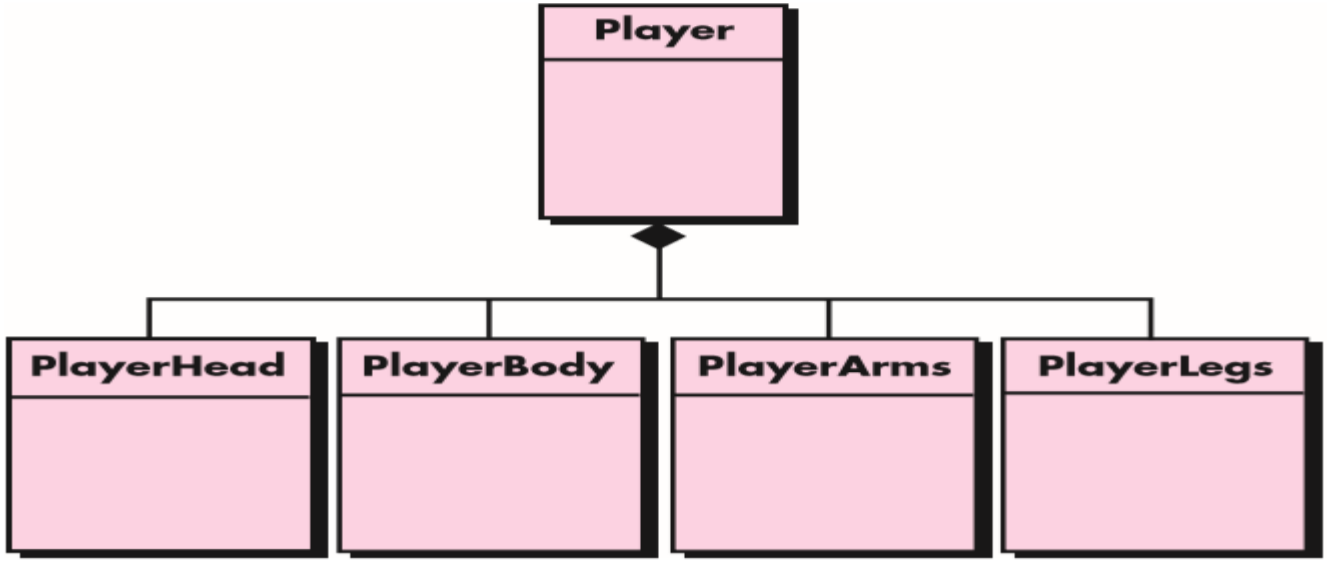
- System intelligence should be distributed across classes to best address the needs of the problem
- Each responsibility should be stated as generally as possible
- Information and the behavior related to it should reside within the same class
- Information about one thing should be localized with a 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, thereby fulfilling a particular responsibility, or
(2) a class can collaborate with other classes.

To help in the identification of collaborators, you can examine three different generic relationships between classes
(1) the is-part-of relationship
(2) the has-knowledge-of relationship, and
(3) the depends-upon relationship.

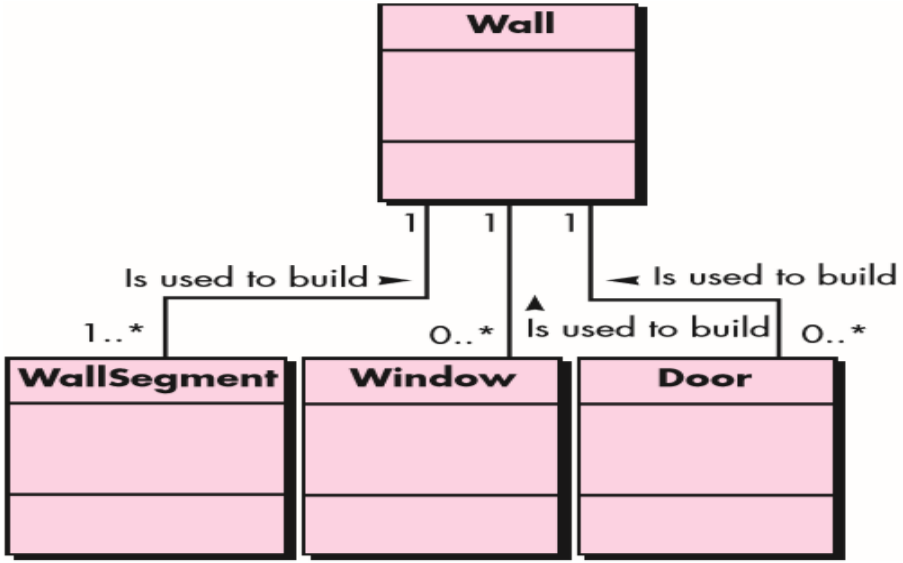
A composite Aggregate class



Associations and Dependencies

- 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.
- 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

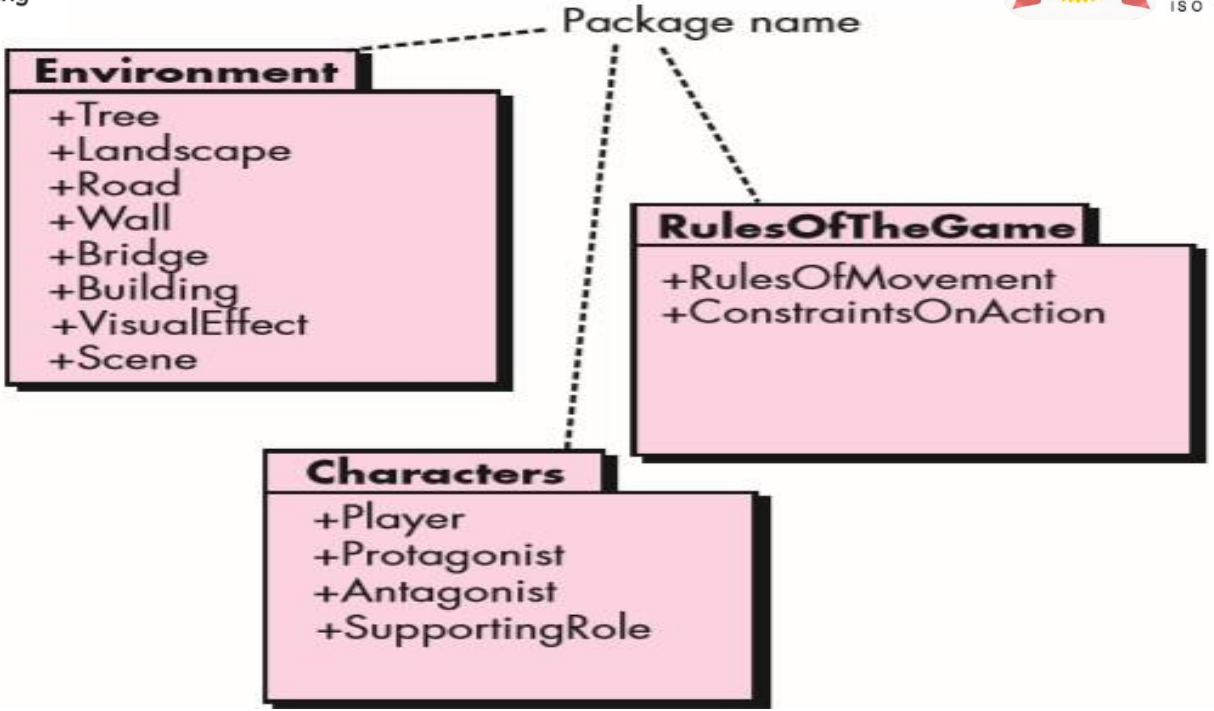
Multiplicity



Dependencies

Analysis Packages

- Various elements of the analysis model (e.g., use-cases, analysis classes) are categorized in a manner that packages them as a grouping.
- The plus sign preceding the analysis class name in each package indicates that the classes have public visibility and are therefore accessible from other packages.
- Other symbols can precede an element within a package.
- A minus sign indicates that an element is hidden from all other packages and a # symbol indicates that an element is accessible only to packages contained within a given package.


Package name
Environment
+Tree
+Landscape
+Road
+Wall
+Bridge
+Building
+VisualEffect
+Scene
RulesOfTheGame
+RulesOfMovement
+ConstraintsOnAction
Characters
+Player
+Protagonist
+Antagonist
+SupportingRole